



IDL

Advanced Math and Statistics

© 2018 Harris Geospatial Solutions, Inc.

Legal and Copyright Notices

The IDL® and ENVI® software programs and the accompanying procedures, functions, and documentation described herein are sold under license agreements. Their use, duplication, and disclosure are subject to the restrictions stated in the license agreement. Harris Geospatial Solutions, Inc. reserves the right to make changes to this document at any time and without notice.

Limitation of Warranty

Harris Geospatial Solutions makes no warranties, either express or implied, as to any matter not expressly set forth in the license agreement, including without limitation the condition of the software, merchantability, or fitness for any particular purpose. Harris Geospatial Solutions shall not be liable for any direct, consequential, or any other damages, suffered by the licensed user or any other users resulting from the use of the software packages or the software documentation.

Permission to Reproduce Manuals

If you are a licensed user of Harris Geospatial Solutions software, Harris Geospatial Solutions grants you a limited, nontransferable license to reproduce its software's manuals provided such copies are for your use only and are not sold or distributed to third parties. All such copies must contain the title page and Harris Geospatial Solutions copyright notice.

Export Control Information

Harris Geospatial Solutions Software and its associated technology are subject to U.S. export controls including the United States Export Administration Regulations. The licensed user is responsible for ensuring compliance with all applicable U.S. export control laws and regulations. These laws include restrictions on destinations, end users and end use.

Copyright and Trademark Notices

IDL® and ENVI® are registered trademarks of Harris Corporation.

Esri®, ArcGIS®, ArcView®, and ArcInfo® are registered trademarks of Esri.

Adobe Illustrator® and Adobe PDF® Print Engine are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries.

Macintosh® is a registered trademark of Apple Inc., registered in the U.S. and other countries.

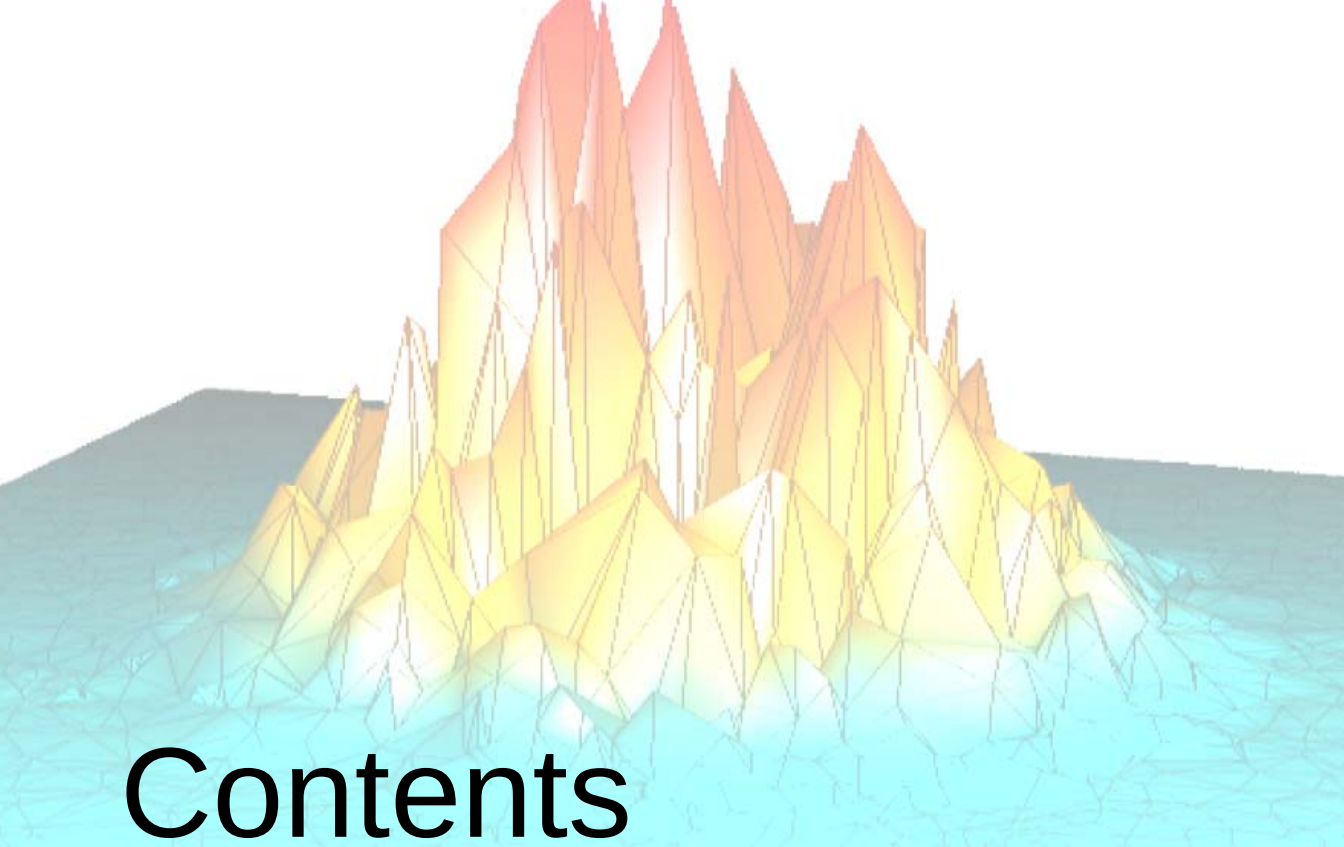
PowerPoint®, PowerPoint icon and Windows® are registered trademarks of Microsoft Corporation in the United States and/or other countries.

UNIX® is a registered trademark of The Open Group.

FLAASH® and QUAC® are registered trademarks of Spectral Sciences, Inc.

Other trademarks and registered trademarks are the property of the respective trademark holders.

© 2018 Harris Geospatial Solutions, Inc.



Contents

Chapter 1	
Preface	13
About IDL Advanced Math and Stats	14
Licensing	14
Using the IDL Advanced Math and Stats Documentation	15
Rows versus Columns	15
Error Handling	17
Underflow and Overflow	17
Missing Values	17
Errors in User Code	17
Chapter 2	
Functional List of	
IMSL Routines	21
Routines for Linear Systems	23
Routines for Eigensystem Analysis	25

Routines for Interpolation and Approximation	26
Routines for Quadrature	27
Routines for Differential Equations	28
Routines for Transforms	29
Routines for Nonlinear Equations	30
Routines for Optimization	31
Routines for Special Functions	32
Routines for Basic Statistics and Random Number Generators	34
Routines for Regression	35
Routines for Correlation and Covariance	37
Routines for Analysis of Variance	38
Routines for Categorical and Discrete Data Analysis	39
Routines for Nonparametric Statistics	40
Routines for Goodness of Fit	41
Routines for Time Series and Forecasting	42
Routines for Multivariate Analysis	43
Routines for Survival Analysis	44
Routines for Probability Distribution Functions and Inverses	45
Routines for Random Number Generation	46
Utility Routines for Math and Statistics	48

Chapter 3

Alphabetical Listing of IMSL Routines	49

Part I: Mathematics Routines

Chapter 4

Linear Systems	63
Overview: Linear Systems	64
Linear Systems Routines	75
IMSL_INV	77
IMSL_LUSOL	79
IMSL_LUFAC	85
IMSL_CHSOL	89
IMSL_CHFAC	93
IMSL_QRSOL	96
IMSL_QRFAC	100
IMSL_SVDCOMP	104

IMSL_CHNND SOL	108
IMSL_CHNND FAC	112
IMSL_LINLSQ	116
IMSL_SP_LUSOL	122
IMSL_SP_LUFAC	128
IMSL_SP_BDSOL	135
IMSL_SP_BDFAC	139
IMSL_SP_PDSOL	143
IMSL_SP_PDFAC	148
IMSL_SP_BDPDSOL	152
IMSL_SP_BDPDFAC	155
IMSL_SP_GMRES	159
IMSL_SP_CG	163
IMSL_SP_MVMUL	167
 Chapter 5	
Eigensystem Analysis	173
Overview: Eigensystem Analysis	174
Eigensystem Routines	177
IMSL_EIG	178
IMSL_EIGSYMGEN	183
IMSL_GENEIG	186
 Chapter 6	
Interpolation and Approximation	191
Overview: Interpolation and Approximation	192
Interpolation and Approximation Routines	199
IMSL_CSINTERP	200
IMSL_CSSHAPE	205
IMSL_BSINTERP	210
IMSL_BSKNOTS	219
IMSL_SPVALUE	224
IMSL_SPINTEG	230
IMSL_FCNLSQ	234
IMSL_BSLSQ	238
IMSL_CONLSQ	248
IMSL_CSSMOOTH	254

IMSL_SMOOTHDATA1D	258
IMSL_SCAT2DINTERP	262
IMSL_RADBF	266
IMSL_RADBE	277

Chapter 7

Quadrature 279

Overview: Quadrature	280
Quadrature Routines	283
IMSL_INTFCN	284
IMSL_INTFCNHYPER	315
IMSL_INTFCN_QMC	319
IMSL_GQUAD	322
IMSL_FCN_DERIV	326

Chapter 8

Differential Equations 329

Overview: Differential Equations	330
Differential Equations Routines	332
IMSL_ODE	333
IMSL_PDE_MOL	351
IMSL_POISSON2D	366

Chapter 9

Transforms 373

Overview: Transforms	374
Transforms Routines	376
IMSL_FFTCOMP	377
IMSL_FFTINIT	387
IMSL_CONVOL1D	390
IMSL_CORR1D	395
IMSL_LAPLACE_INV	398

Chapter 10

Nonlinear Equations 407

Overview: Nonlinear Equations	408
Nonlinear Equations Routines	409
IMSL_ZEROPOLY	410

IMSL_ZEROFCN	413
IMSL_ZEROSYS	418

Chapter 11

Optimization 421

Overview: Optimization	422
Optimization Routines	424
IMSL_FMIN	425
IMSL_FMINV	433
IMSL_NLINLSQ	441
IMSL_LINPROG	449
IMSL_QUADPROG	454
IMSL_MINCONGEN	458
IMSL_CONSTRAINED_NLP	465

Chapter 12

Special Functions 473

Overview: Special Functions	474
Special Functions Routines	475
IMSL_ERF	477
IMSL_ERFC	480
IMSL_BETA	484
IMSL_LNBETA	487
IMSL_BETAI	489
IMSL_LNGAMMA	491
IMSL_GAMMA_ADV	493
IMSL_GAMMAI	495
IMSL_BESSI	498
IMSL_BESSJ	500
IMSL_BESSK	502
IMSL_BESSY	504
IMSL_BESSI_EXP	506
IMSL_BESSK_EXP	508
IMSL_ELK	510
IMSL_ELE	512
IMSL_ELRF	514
IMSL_ELRD	516

IMSL_ELRLJ	518
IMSL_ELRC	520
IMSL_FRESNEL_COSINE	522
IMSL_FRESNEL_SINE	524
IMSL_AIRY_AI	526
IMSL_AIRY_BI	528
IMSL_KELVIN_BER0	531
IMSL_KELVIN_BEI0	533
IMSL_KELVIN_KER0	535
IMSL_KELVIN_KEI0	537

Part II: Statistics Routines

Chapter 13

Basic Statistics 541

Overview: Basic Statistics	542
Basic Statistics Routines	543
IMSL_SIMPLESTAT	544
IMSL_NORM1SAMP	550
IMSL_NORM2SAMP	555
IMSL_FREQTABLE	563
IMSL_SORTDATA	570
IMSL_RANKS	577

Chapter 14

Regression 585

Overview: Regression	586
Regression Routines	599
IMSL_REGRESSORS	600
IMSL_MULTIREGRESS	607
IMSL_MULTIPREDICT	622
IMSL_ALLBEST	630
IMSL_STEPWISE	639
IMSL_POLYREGRESS	649
IMSL_POLYPREDICT	657
IMSL_NONLINREGRESS	665
IMSL_HYPOTH_PARTIAL	675

IMSL_HYPOTH_SCPH	681
IMSL_HYPOTH_TEST	686
IMSL_NONLINOPT	694
IMSL_LNORMREGRESS	703

Chapter 15

Correlation and Covariance 719

Overview: Correlation and Covariance	720
Correlation and Covariance Routines	721
IMSL_COVARIANCES	722
IMSL_PARTIAL_COV	728
IMSL_POOLED_COV	734
IMSL_ROBUST_COV	738

Chapter 16

Analysis of Variance 747

Overview: Analysis of Variance	748
Analysis of Variance Routines	749
IMSL_ANOVA1	750
IMSL_ANOVAFACT	760
IMSL_MULTICOMP	769
IMSL_ANOVANESTED	772
IMSL_ANOVABALANCED	781

Chapter 17

Categorical and Discrete Data Analysis 793

Overview: Categorical and Discrete Data Analysis	794
Categorical and Discrete Data Analysis Routines	795
IMSL_CONTINGENCY	796
IMSL_EXACT_ENUM	809
IMSL_EXACT_NETWORK	812
IMSL_CAT_GLM	817

Chapter 18

Nonparametric Statistics 831

Overview	832
Nonparametric Statistics Routines	833
IMSL_SIGNTEST	834

IMSL_WILCOXON	837
IMSL_NCTRENDS	846
IMSL_CSTRENDS	849
IMSL_TIE_STATS	855
IMSL_KW_TEST	857
IMSL_FRIEDMANS_TEST	860
IMSL_COCHRANQ	865
IMSL_KTRENDS	868

Chapter 19

Goodness of Fit 873

Overview: Goodness of Fit	874
Goodness of Fit Routines	875
IMSL_CHISQTEST	876
IMSL_NORMALITY	882
IMSL_KOLMOGOROV1	886
IMSL_KOLMOGOROV2	889
IMSL_MVAR_NORMALITY	892
IMSL_RANDOMNESS_TEST	897

Chapter 20

Time Series and Forecasting 909

Overview: Time Series and Forecasting	910
Time Series and Forecasting Routines	912
IMSL_ARMA	913
IMSL_DIFFERENCE	929
IMSL_BOXCOXTRANS	935
IMSL_AUTOCORRELATION	940
IMSL_PARTIAL_AC	945
IMSL_LACK_OF_FIT	948
IMSL_GARCH	952
IMSL_KALMAN	957

Chapter 21

Multivariate Analysis 967

Overview: Multivariate Analysis	968
Multivariate Analysis Routines	970

IMSL_K_MEANS	971
IMSL_PRINC_COMP	976
IMSL_FACTOR_ANALYSIS	981
IMSL_DISCR_ANALYSIS	992

Chapter 22

Survival Analysis 1003

Overview: Survival Analysis	1004
Survival Analysis Routines	1005
IMSL_SURVIVAL_GLM	1006

Chapter 23

Probability Distribution Functions and Inverses 1029

Overview: Probability Distribution Functions and Inverses	1030
Probability Distribution Functions and Inverses Routines	1031
IMSL_NORMALCDF	1032
IMSL_BINORMALCDF	1035
IMSL_CHISQCDF	1038
IMSL_FCDF	1043
IMSL_TCDF	1046
IMSL_GAMMACDF	1050
IMSL_BETACDF	1053
IMSL_BINOMIALCDF	1056
IMSL_BINOMIALPDF	1058
IMSL_HYPERGEOCDF	1060
IMSL_POISSONCDF	1063

Chapter 24

Random Number Generation 1065

Overview: Random Number Generation	1066
Random Number Generation Routines	1069
IMSL_RANDOMOPT	1071
IMSL_RANDOM_TABLE	1076
IMSL_RANDOM	1080
IMSL_RANDOM_NPP	1100
IMSL_RANDOM_ORDER	1104
IMSL_RAND_TABLE_2WAY	1107

IMSL_RANDOM_ORTH_MAT	1109
IMSL_RANDOM_SAMPLE	1112
IMSL_RANDOM_FROM_DATA	1115
IMSL_CONT_TABLE	1118
IMSL_RANDOM_GEN_CONT	1120
IMSL_DISCR_TABLE	1123
IMSL_RANDOM_GEN_DISCR	1127
IMSL_RANDOM_ARMA	1131
IMSL_FAURE_INIT	1136
IMSL_FAURE_NEXT_PT	1140

Chapter 25

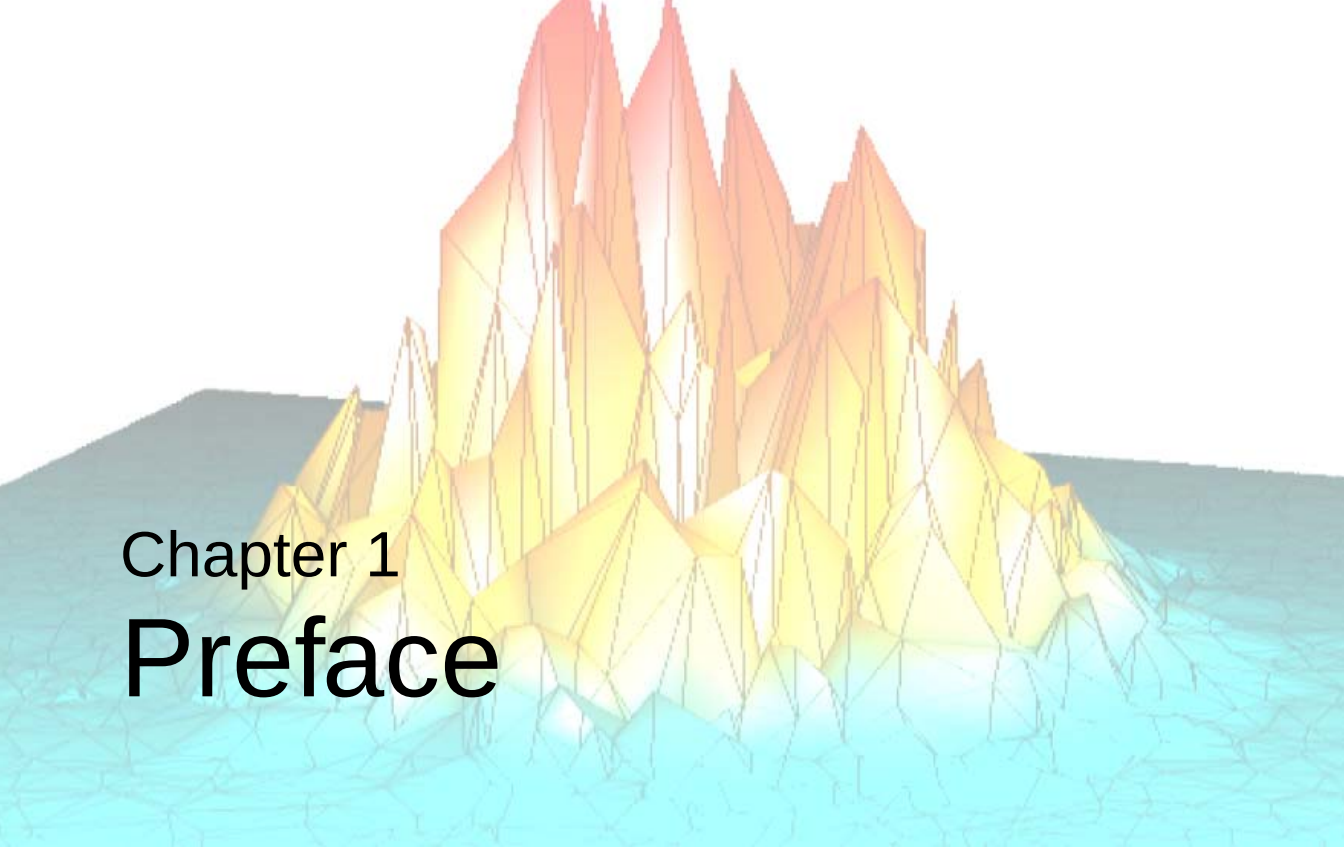
Math and Statistics Utilities 1145

Overview: Math and Statistics Utilities	1146
Math and Statistics Utilities Routines	1147
IMSL_DAYSTODATE	1148
IMSL_DATETODAYS	1150
IMSL_CONSTANT	1152
IMSL_MACHINE	1158
IMSL_STATDATA	1163
IMSL_BINOMIALCOEF	1166
IMSL_NORM	1168
IMSL_MATRIX_NORM	1171
PM	1176
RM	1178

Appendix A

References 1181

Index 1195



Chapter 1

Preface

This section contains the following topics:

About IDL Advanced Math and Stats	14	Error Handling	17
Using the IDL Advanced Math and Stats Documentation	15		

About IDL Advanced Math and Stats

The IDL Advanced Math and Stats module combines the power of IDL with the IMSL C Numerical Library provided by Visual Numerics, Inc. The addition of the IMSL library gives IDL users access to an extensive and powerful set of mathematical and statistical analysis routines via the standard IDL programmer's interface.

If you have used the IMSL libraries when creating C or FORTRAN applications, much of the functionality in IDL Advanced Math and Stats will be familiar. But because the IMSL functionality is exposed via an IDL interface, no linking or compiling is required. Use the IMSL routines as you would any other IDL function or procedure.

IDL Advanced Math and Stats provides a subset of the full IMSL C Numerical Library version 5. See [Chapter 2, “Functional List of IMSL Routines”](#) for a complete listing of the included routines.

Licensing

IDL Advanced Math and Stats is a separately licensed IDL module. IDL applications that incorporate IMSL functionality will not function if the IDL Advanced Math and Stats license is not present; this means that if you distribute an application that uses IMSL functionality, the end-users of your application must also have an IDL Advanced Math and Stats license. For information on runtime licensing of IMSL functionality, contact your sales representative.

Using the IDL Advanced Math and Stats Documentation

The chapters of the *IDL Advanced Math and Stats* guide group routines with similar computational or analytical capabilities. To locate the appropriate function for a given problem, refer to [Chapter 2, “Functional List of IMSL Routines”](#). If you know the name of the routine you wish to use, consult [Chapter 3, “Alphabetical Listing of IMSL Routines”](#) to locate the routine’s documentation.

Each chapter of the *IDL Advanced Math and Stats* provides an overview of the functionality described in that chapter, along with information on the types of problems addressed by that functionality.

Rows versus Columns

IDL Advanced Math and Stats uses the standard linear algebraic convention for two-dimensional arrays: “row” refers to the first index of the array and “column” refers to the second. So for a 2D array A , $A(i,j)$ is the element in row i and column j . The [PM](#) procedure makes this easy to visualize:

```
a = INTARR( 4, 8 ) & a(2,5) = 1 & PM, a
```

IDL Prints:

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0
```

Note that this is the opposite of the standard image processing convention used in most IDL documentation, where “column” refers to the first index of the array and “row” refers to the second. Using the standard IDL PRINT procedure, the above array would look like this:

```
a = INTARR( 4, 8 ) & a(2,5) = 1 & PRINT, a
```

IDL Prints:

```
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
0 0 1 0
0 0 0 0
```

0 0 0 0

For additional information, see [“Columns, Rows, and Array Majority”](#) (Chapter 15, *Application Programming*).

References

References are listed alphabetically by author in [Appendix A, “References”](#).

Error Handling

IDL Advanced Math and Stats uses IDL's built-in error handling mechanisms for most errors. This section describes areas in which IDL Advanced Math and Stats may provide a greater level of control than IDL itself.

Underflow and Overflow

In most cases, IDL Advanced Math and Stats routines are written so that computations are not affected by underflow, provided the system (hardware or software) replaces an underflow with the value zero. Normally, system error messages indicating underflow can be ignored.

IDL Advanced Math and Stats routines also are written to avoid overflow. A program that produces system error messages indicating overflow should be examined for programming errors such as incorrect input data, mismatch of parameter types, or improper dimensions.

In many cases, the documentation for a function points out common pitfalls that can lead to failure of the algorithm.

Missing Values

Some IDL Advanced Math and Stats routines allow input data to contain missing values. These routines recognize as a missing value the special floating-point value referred to as “Not a Number” or NaN. The actual value varies on different computers, but it can be obtained by reference to the [IMSL_MACHINE](#) function.

The manner in which missing values are treated depends on the individual function as described in the documentation for that function.

For more information on special floating-point values (including NaN), see “[Math Errors](#)” (Chapter 8, *Application Programming*).

Errors in User Code

IDL Advanced Math and Stats functions attempt to detect user errors and handle them in a way that provides as much information to the user as possible. In addition to the basic IDL error-handling facility, five levels of *Informational Error* severity are recognized. The error levels are described in [Table 1-1](#).

Error Levels and Default Actions

The IMSL numerical library categorizes library errors with one of five severity levels:

Type	Meaning
Note	A <i>note</i> is issued to indicate the possibility of a trivial error or simply to provide information about the computations. A note does not update !ERROR_STATE.
Alert	An <i>alert</i> indicates that the user should be advised about conditions that arise during computation. Underflow errors are generally categorized as alerts. An alert does not update !ERROR_STATE.
Warning	A <i>warning</i> indicates the existence of a condition that may require corrective action by the user or calling routine. A warning error may be issued because the results are accurate to only a few decimal places, because some of the output may be erroneous but most of the output is correct, or because some assumptions underlying the analysis technique are violated. Often no corrective action is necessary and the condition can be ignored. A warning does not update !ERROR_STATE.
Fatal	A <i>fatal</i> error indicates the existence of a condition that may be serious. In most cases, the user or calling routine must take corrective action to recover. A fatal error updates !ERROR_STATE.
Terminal	A <i>terminal</i> error is serious. It usually is the result of an incorrect specification, such as specifying a negative number as the number of equations. Terminal errors may also be caused by various programming errors that are impossible to diagnose correctly within the IMSL library. If a terminal error occurs, first check that the arguments passed to the routine are in the correct order and have the correct data types. A terminal error updates !ERROR_STATE.

Table 1-1: Error levels generated by the IMSL numerical library

Although the IDL Advanced Math and Stats module does not allow users to directly manipulate how these errors are interpreted internally, you can control which errors are printed to the IDL output log. All informational error messages are printed by default. Setting the system variable !QUIET to a nonzero value suppresses output of Notes, Alerts, and Warnings. Fatal and Terminal errors always halt execution of the IDL program and change the value of !ERROR_STATE.

Handling Errors in IMSL Routines

When a fatal or terminal error occurs in an IMSL routine, the value of `!ERROR_STATE` is updated to reflect the fact that the error occurred. If you have implemented a `CATCH` block to handle errors in your own routine, you can use the value of `!ERROR_STATE` to determine which fatal or terminal error occurred in the IMSL library.

To determine whether the most recent error was generated by the IMSL library, inspect the `NAME` field of the `!ERROR_STATE` structure. Errors generated by the IMSL library will populate the `NAME` field with the string:

```
IDL_M_IMSL_LIBRARYERROR
```

If the error was generated in the IMSL library, inspect the `MSG` field of the `!ERROR_STATE` structure for information on which specific fatal or terminal error occurred. For example, attempting to invert a matrix in which every element is zero will generate a fatal error with the following message:

```
IMSL Error: IMSL_INV: Fatal error: MATH_SINGULAR_MATRIX: The input
matrix is singular.
```

You could, for example, use the following code fragment to test for this particular error:

```
IF (STRPOS(!error_state.msg, 'MATH_SINGULAR_MATRIX') GE 0) THEN $
  BEGIN
    Error handling code here...
  ENDIF
```




Chapter 2

Functional List of IMSL Routines

This chapter contains a list of IMSL routines included in the IDL Advanced Math and Stats package, categorized by functional categories:

- [“Routines for Linear Systems”](#) on page 23
- [“Routines for Eigensystem Analysis”](#) on page 25
- [“Routines for Interpolation and Approximation”](#) on page 26
- [“Routines for Quadrature”](#) on page 27
- [“Routines for Differential Equations”](#) on page 28
- [“Routines for Transforms”](#) on page 29
- [“Routines for Nonlinear Equations”](#) on page 30
- [“Routines for Optimization”](#) on page 31
- [“Routines for Special Functions”](#) on page 32
- [“Routines for Basic Statistics and Random Number Generators”](#) on page 34
- [“Routines for Regression”](#) on page 35

- “Routines for Correlation and Covariance” on page 37
- “Routines for Analysis of Variance” on page 38
- “Routines for Categorical and Discrete Data Analysis” on page 39
- “Routines for Nonparametric Statistics” on page 40
- “Routines for Goodness of Fit” on page 41
- “Routines for Time Series and Forecasting” on page 42
- “Routines for Multivariate Analysis” on page 43
- “Routines for Survival Analysis” on page 44
- “Routines for Probability Distribution Functions and Inverses” on page 45
- “Routines for Random Number Generation” on page 46
- “Utility Routines for Math and Statistics” on page 48

Routines for Linear Systems

See [Chapter 4, “Linear Systems”](#) or select a link below.

Matrix Inversion

[IMSL_INV](#)—General matrix inversion.

Linear Equations with Full Matrices

[IMSL_LUSOL](#)—Systems involving general matrices.

[IMSL_LUFAC](#)— LU factorization of general matrices.

[IMSL_CHSOL](#)—Systems involving symmetric positive definite matrices.

[IMSL_CHFAC](#)—Factorization of symmetric positive definite matrices.

Linear Least Squares with Full Matrices

[IMSL_QRSOL](#)—Least-squares solution.

[IMSL_QRFAC](#)—Least-squares factorization.

[IMSL_SVDCOMP](#)—Singular Value Decomposition (SVD) and generalized inverse.

[IMSL_CHNND SOL](#)—Solve and generalized inverse for positive semidefinite matrices.

[IMSL_CHN NDFAC](#)—Factor and generalized inverse for positive semidefinite matrices.

[IMSL_LINLSQ](#)—Linear constraints.

Sparse Matrices

[IMSL_SP_LUSOL](#)—Solve a sparse system of linear equations $Ax = b$.

[IMSL_SP_LUFAC](#)—Compute an LU factorization of a sparse matrix stored in either coordinate format or CSC format.

[IMSL_SP_BDSOL](#)—Solve a general band system of linear equations $Ax = b$.

[IMSL_SP_BDFAC](#)—Compute the LU factorization of a matrix stored in band storage mode.

IMSL_SP_PDSOL—Solve a sparse symmetric positive definite system of linear equations $Ax = b$.

IMSL_SP_PDFAC—Compute a factorization of a sparse symmetric positive definite system of linear equations $Ax = b$.

IMSL_SP_BDPDSOL—Solve a symmetric positive definite system of linear equations $Ax = b$ in band symmetric storage mode.

IMSL_SP_BDPDFAC—Compute the $R^T R$ Cholesky factorization of symmetric positive definite matrix, A , in band symmetric storage mode.

IMSL_SP_GMRES—Solve a linear system $Ax = b$ using the restarted generalized minimum residual (GMRES) method.

IMSL_SP_CG—Solve a real symmetric definite linear system using a conjugate gradient method.

IMSL_SP_MVMUL—Compute a matrix-vector product involving a sparse matrix and a dense vector.

Routines for Eigensystem Analysis

See [Chapter 5, “Eigensystem Analysis”](#) or select a link below.

Linear Eigensystem Problems

[IMSL_EIG](#)—General and symmetric matrices.

Generalized Eigensystem Problems

[IMSL_EIGSYMGGEN](#)—Real symmetric matrices and B positive definite.

[IMSL_GENEIG](#)—General eigenexpansion of $Ax=\lambda Bx$.

Routines for Interpolation and Approximation

See [Chapter 6, “Interpolation and Approximation”](#) or select a link below.

Cubic Spline Interpolation

[IMSL_CSINTERP](#)—Derivative end conditions.

[IMSL_CSSHAPE](#)—Shape preserving.

B-spline Interpolation

[IMSL_BSINTERP](#)—One-dimensional and two-dimensional interpolation.

[IMSL_BSKNOTS](#)—Knot sequence given interpolation data.

B-spline and Cubic Spline Evaluation and Integration

[IMSL_SPVALUE](#)—Evaluation and differentiation.

[IMSL_SPINTEG](#)—Integration.

Least-squares Approximation and Smoothing

[IMSL_FCNLSQ](#)—General functions.

[IMSL_BSLSQ](#)—Splines with fixed knots.

[IMSL_CONLSQ](#)—Constrained spline fit.

[IMSL_CSSMOOTH](#)—Cubic-smoothing spline.

[IMSL_SMOOTHDATA1D](#)—Smooth one-dimensional data by error detection.

Scattered Data Interpolation

[IMSL_SCAT2DINTERP](#)—Akima’s surface-fitting method.

[IMSL_RADBF](#)—Computes a fit using radial-basis functions.

[IMSL_RADBE](#)—Evaluates a radial-basis fit.

Routines for Quadrature

See [Chapter 7, “Quadrature”](#) or select a link below.

Univariate and Bivariate Quadrature

[IMSL_INTFCN](#)—Integration of a user-defined univariate or bivariate function.

Arbitrary Dimension Quadrature

[IMSL_INTFCNHYPER](#)—Iterated integral on a hyper-rectangle.

[IMSL_INTFCN_QMC](#)—Integrates a function on a hyper-rectangle using a Quasi Monte Carlo method.

Gauss Quadrature

[IMSL_GQUAD](#)—Gauss quadrature formulas.

Differentiation

[IMSL_FCN_DERIV](#)—First, second, or third derivative of a function.

Routines for Differential Equations

See [Chapter 8, “Differential Equations”](#) or select a link below.

[IMSL_ODE](#)—Adams-Gear or Runge-Kutta method.

[IMSL_PDE_MOL](#)—Solves a system of partial differential equations using the method of lines.

[IMSL_POISSON2D](#)—Solves Poisson’s or Helmholtz’s equation on a two-dimensional rectangle.

Routines for Transforms

See [Chapter 9, “Transforms”](#) or select a link below.

[IMSL_FFTCOMP](#)—Real or complex FFT.

[IMSL_FFTINIT](#)—Real or complex FFT initialization.

[IMSL_CONVOL1D](#)—Compute discrete convolution.

[IMSL_CORR1D](#)—Compute discrete correlation.

[IMSL_LAPLACE_INV](#)—Approximate inverse Laplace transform of a complex function.

Routines for Nonlinear Equations

See [Chapter 10, “Nonlinear Equations”](#) or select a link below.

Zeros of a Polynomial

[IMSL_ZEROPOLY](#)—Real or complex coefficients.

Zeros of a Function

[IMSL_ZEROFCN](#)—Real zeros of a function.

Root of a System of Equations

[IMSL_ZEROSYS](#)—Powell’s hybrid method.

Routines for Optimization

See [Chapter 11, “Optimization”](#) or select a link below.

Unconstrained Minimization

[IMSL_FMIN](#)—(Univariate Function) Using function and possibly first derivative values.

[IMSL_FMINV](#)—(Multivariate Function) Using quasi-Newton method.

[IMSL_NLINLSQ](#)—(Nonlinear Least Squares) Using Levenberg-Marquardt algorithm.

Linearly Constrained Minimization

[IMSL_LINPROG](#)—Dense linear programming.

[IMSL_QUADPROG](#)—Quadratic programming.

Nonlinearly Constrained Minimization

[IMSL_MINCONGEN](#)—Minimize a general objective function.

[IMSL_CONSTRAINED_NLP](#)—Using a sequential equality constrained quadratic programming method.

Routines for Special Functions

See [Chapter 12, “Special Functions”](#) or select a link below.

Error Functions

[IMSL_ERF](#)—Error function.

[IMSL_ERFC](#)—Complementary error function.

[IMSL_BETA](#)—Beta function.

[IMSL_LNBETA](#)—Logarithmic beta function.

[IMSL_BETAI](#)—Incomplete beta function.

Gamma Functions

[IMSL_LNGAMMA](#)—Logarithmic gamma function.

[IMSL_GAMMA_ADV](#)—Real gamma function.

[IMSL_GAMMAI](#)—Incomplete gamma function.

Bessel Functions with Real Order and Complex Argument

[IMSL_BESSI](#)—Modified Bessel function of the first kind.

[IMSL_BESSJ](#)—Bessel function of the first kind.

[IMSL_BESSK](#)—Modified Bessel function of the second kind.

[IMSL_BESSY](#)—Bessel function of the second kind.

[IMSL_BESSI_EXP](#)—Bessel function $e^{-|x|}I_0(x)$, Bessel function $e^{-|x|}I_1(x)$.

[IMSL_BESSK_EXP](#)—Bessel function $e^xK_0(x)$, Bessel function $e^xK_1(x)$.

Elliptic Integrals

[IMSL_ELK](#)—Complete elliptic integral of the first kind.

[IMSL_ELE](#)—Complete elliptic integral of the second kind.

[IMSL_ELRF](#)—Carlson’s elliptic integral of the first kind.

[IMSL_ELRD](#)—Carlson’s elliptic integral of the second kind.

[IMSL_ELRJ](#)—Carlson's elliptic integral of the third kind.

[IMSL_ELRC](#)—Special case of Carlson's elliptic integral.

Fresnel Integrals

[IMSL_FRESNEL_COSINE](#)—Cosine Fresnel integral.

[IMSL_FRESNEL_SINE](#)—Sine Fresnel integral.

Airy Functions

[IMSL_AIRY_AI](#)—Airy function, and derivative of the Airy function.

[IMSL_AIRY_BI](#)—Airy function of the second kind, and derivative of the Airy function of the second kind.

Kelvin Functions

[IMSL_KELVIN_BER0](#)—Kelvin function *ber* of the first kind, order 0, and derivative of the Kelvin function *ber*.

[IMSL_KELVIN_BEI0](#)—Kelvin function *bei* of the first kind, order 0, and derivative of the Kelvin function *bei*.

[IMSL_KELVIN_KER0](#)—Kelvin function *ker* of the second kind, order 0, and derivative of the Kelvin function *ker*.

[IMSL_KELVIN_KEI0](#)—Kelvin function *kei* of the second kind, order 0 and derivative of the Kelvin function *kei*.

Routines for Basic Statistics and Random Number Generators

See [Chapter 13, “Basic Statistics”](#) or select a link below.

Simple Summary Statistics

[IMSL_SIMPLESTAT](#)—Univariate summary statistics.

[IMSL_NORM1SAMP](#)—Mean and variance inference for a single normal population.

[IMSL_NORM2SAMP](#)—Inferences for two normal populations.

Tabulate, Sort, and Rank

[IMSL_FREQTABLE](#)—Tallies observations into a one-way frequency table.

[IMSL_SORTDATA](#)—Sorts data with options to tally cases into a multiway frequency table.

[IMSL_RANKS](#)—Ranks, normal scores, or exponential scores.

Routines for Regression

See [Chapter 14, “Regression”](#) or select a link below.

Multiple Linear Regression

[IMSL_REGRESSORS](#)—Generates regressors for a general linear model.

[IMSL_MULTIREGRESS](#)—Fits a multiple linear regression model and optionally produces summary statistics for a regression model.

[IMSL_MULTIPREDICT](#)—Computes predicted values, confidence intervals, and diagnostics.

Variable Selection

[IMSL_ALLBEST](#)—All best regressions.

[IMSL_STEPWISE](#)—Stepwise regression.

Polynomial and Nonlinear Regression

[IMSL_POLYREGRESS](#)—Fits a polynomial regression model.

[IMSL_POLYPREDICT](#)—Computes predicted values, confidence intervals, and diagnostics.

[IMSL_NONLINREGRESS](#)—Fits a nonlinear regression model.

Multivariate Linear Regression—Statistical Inference and Diagnostics

[IMSL_HYPOTH_PARTIAL](#)—Construction of a completely testable hypothesis.

[IMSL_HYPOTH_SCPH](#)—Sums of cross products for a multivariate hypothesis.

[IMSL_HYPOTH_TEST](#)—Tests for the multivariate linear hypothesis.

Polynomial and Nonlinear Regression

[IMSL_NONLINOPT](#)—Fit a nonlinear regression model using Powell's algorithm.

Alternatives to Least Squares Regression

[IMSL_LNORMREGRESS](#)—LAV, Lpnorm, and LMV criteria regression.

Routines for Correlation and Covariance

See [Chapter 15, “Correlation and Covariance”](#) or select a link below.

[IMSL_COVARIANCES](#)—Variance-covariance or correlation matrix.

[IMSL_PARTIAL_COV](#)—Partial correlations and covariances.

[IMSL_POOLED_COV](#)—Pooled covariance matrix.

[IMSL_ROBUST_COV](#)—Robust estimate of covariance matrix.

Routines for Analysis of Variance

See [Chapter 16, “Analysis of Variance”](#) or select a link below.

[IMSL_ANOVA1](#)—Analyzes a one-way classification model.

[IMSL_ANOVAFACT](#)—Analyzes a balanced factorial design with fixed effects.

[IMSL_MULTICOMP](#)—Performs Student-Newman-Keuls multiple comparisons test.

[IMSL_ANOVANESTED](#)—Nested random model.

[IMSL_ANOVABALANCED](#)—Balanced fixed, random, or mixed model.

Routines for Categorical and Discrete Data Analysis

See [Chapter 17, “Categorical and Discrete Data Analysis”](#) or select a link below.

Statistics in the Two-Way Contingency Table

[IMSL_CONTINGENCY](#)—Two-way contingency table analysis.

[IMSL_EXACT_ENUM](#)—Exact probabilities in a table; total enumeration.

[IMSL_EXACT_NETWORK](#)—Exact probabilities in a table.

Generalized Categorical Models

[IMSL_CAT_GLM](#)—Generalized linear models.

Routines for Nonparametric Statistics

See [Chapter 18, “Nonparametric Statistics”](#) or select a link below.

One Sample Tests—Nonparametric Statistics

[IMSL_SIGNTEST](#)—Sign test.

[IMSL_WILCOXON](#)—Wilcoxon rank sum test.

[IMSL_NCTRENDS](#)—Noether’s test for cyclical trend.

[IMSL_CSTRENDS](#)—Cox and Stuarts’ sign test for trends in location and dispersion.

[IMSL_TIE_STATS](#)—Tie statistics.

Two or More Samples Tests—Nonparametric Statistics

[IMSL_KW_TEST](#)—Kruskal-Wallis test.

[IMSL_FRIEDMANS_TEST](#)—Friedman’s test.

[IMSL_COCHRANQ](#)—Cochran’s Q test.

[IMSL_KTRENDS](#)—K-sample trends test.

Routines for Goodness of Fit

See [Chapter 19, “Goodness of Fit”](#) or select a link below.

General Goodness of Fit Tests

[IMSL_CHISQTEST](#)—Chi-squared goodness of fit test.

[IMSL_NORMALITY](#)—Shapiro-Wilk W test for normality.

[IMSL_KOLMOGOROV1](#)—One-sample continuous data Kolmogorov-Smirnov.

[IMSL_KOLMOGOROV2](#)—Two-sample continuous data Kolmogorov-Smirnov.

[IMSL_MVAR_NORMALITY](#)—Mardia’s test for multivariate normality.

Tests for Randomness

[IMSL_RANDOMNESS_TEST](#)—Runs test, Paris-serial test, d^2 test or triplets tests.

Routines for Time Series and Forecasting

See [Chapter 20, “Time Series and Forecasting”](#) or select a link below.

IMSL_ARMA Models

[IMSL_ARMA](#)—Computes least-squares or method-of-moments estimates of parameters and optionally computes forecasts and their associated probability limits.

[IMSL_DIFFERENCE](#)—Performs differencing on a time series.

[IMSL_BOXCOXTRANS](#)—Perform a Box-Cox transformation.

[IMSL_AUTOCORRELATION](#)—Sample autocorrelation function.

[IMSL_PARTIAL_AC](#)—Sample partial autocorrelation function.

[IMSL_LACK_OF_FIT](#)—Lack-of-fit test based on the correlation function.

[IMSL_GARCH](#)—Compute estimates of the parameters of a GARCH(p,q) model.

[IMSL_KALMAN](#)—Performs Kalman filtering and evaluates the likelihood function for the statespace model.

Routines for Multivariate Analysis

See [Chapter 21, “Multivariate Analysis”](#) or select a link below.

- [IMSL_K_MEANS](#)—Performs a K -means (centroid) cluster analysis.
- [IMSL_PRINC_COMP](#)—Computes principal components.
- [IMSL_FACTOR_ANALYSIS](#)—Extracts factor-loading estimates.
- [IMSL_DISCR_ANALYSIS](#)—Perform discriminant function analysis.

Routines for Survival Analysis

See [Chapter 22, “Survival Analysis”](#) or select a link below.

- [IMSL_SURVIVAL_GLM](#)—Analyzes survival data using a generalized linear model and estimates using various parametric modes.

Routines for Probability Distribution Functions and Inverses

See [Chapter 23, “Probability Distribution Functions and Inverses”](#) or select a link below.

[IMSL_NORMALCDF](#)—Normal (Gaussian) distribution function.

[IMSL_BINORMALCDF](#)—Bivariate normal distribution.

[IMSL_CHISQCDF](#)—Chi-squared distribution function.

[IMSL_FCDF](#)— F distribution function.

[IMSL_TCDF](#)—Student’s t distribution function.

[IMSL_GAMMACDF](#)—Gamma distribution function.

[IMSL_BETACDF](#)—Beta distribution function.

[IMSL_BINOMIALCDF](#)—Binomial distribution function.

[IMSL_BINOMIALPDF](#)—Binomial probability function.

[IMSL_HYPERGEOCDF](#)—Hypergeometric distribution function.

[IMSL_POISSONCDF](#)—Poisson distribution function.

Routines for Random Number Generation

See [Chapter 24, “Random Number Generation”](#) or select a link below.

Random Numbers

[IMSL_RANDOMOPT](#)—Retrieves uniform (0, 1) multiplicative, congruential pseudorandom-number generator.

[IMSL_RANDOM_TABLE](#)—Sets or retrieves the current table used in either the shuffled or GFSR random number generator.

[IMSL_RANDOM](#)—Generates pseudorandom numbers.

[IMSL_RANDOM_NPP](#)—Generates pseudorandom numbers from a nonhomogeneous Poisson process.

[IMSL_RANDOM_ORDER](#)—Generates pseudorandom order statistics from a uniform (0, 1) distribution, or optionally from a standard normal distribution.

[IMSL_RAND_TABLE_2WAY](#)—Generates a pseudorandom two-way table.

[IMSL_RAND_ORTH_MAT](#)—Generates a pseudorandom orthogonal matrix or a correlation matrix.

[IMSL_RANDOM_SAMPLE](#)—Generates a simple pseudorandom sample from a finite population.

[IMSL_RAND_FROM_DATA](#)—Generates pseudorandom numbers from a multivariate distribution determined from a given sample.

[IMSL_CONT_TABLE](#)—Sets up table to generate pseudorandom numbers from a general continuous distribution.

[IMSL_RAND_GEN_CONT](#)—Generates pseudorandom numbers from a general continuous distribution.

[IMSL_DISCR_TABLE](#)—Sets up table to generate pseudorandom numbers from a general discrete distribution.

[IMSL_RAND_GEN_DISCR](#)—Generates pseudorandom numbers from a general discrete distribution using an alias method or optionally a table lookup method.

Stochastic Processes

[IMSL_RANDOM_ARMA](#)—Generate pseudorandom IMSL_ARMA process numbers.

Low-discrepancy Sequences

[IMSL_FAURE_INIT](#)—Initializes the structure used for computing a shuffled Faure sequence.

[IMSL_FAURE_NEXT_PT](#)—Generates a shuffled Faure sequence.

Utility Routines for Math and Statistics

See [Chapter 25, “Math and Statistics Utilities”](#) or select a link below.

Dates

[IMSL_DAYSTODATE](#)—Days since epoch to date.

[IMSL_DATETODAYS](#)—Date to days since epoch.

Constants and Data Sets

[IMSL_CONSTANT](#)—Natural and mathematical constants.

[IMSL_MACHINE](#)—Machine constants.

[IMSL_STATDATA](#)—Commonly analyzed data sets.

Binomial Coefficient

[IMSL_BINOMIALCOEF](#)—Evaluates the binomial coefficient.

Geometry

[IMSL_NORM](#)—Vector norms.

Matrix Norm

[IMSL_MATRIX_NORM](#)—Real coordinate matrix.

Matrix Entry and Display

[PM](#)—Formatted output of arrays using the standard linear algebraic convention: “row” refers to the first index of the array and “column” refers to the second.

[RM](#)—Formatted input of arrays using the standard linear algebraic convention: “row” refers to the first index of the array and “column” refers to the second.



Chapter 3

Alphabetical Listing of IMSL Routines

This chapter contains an alphabetical listing of routines included in the IDL Advanced Math and Stats module.

[“IMSL_AIRY_AI”](#) on page 526—Evaluates the Airy function.

[“IMSL_AIRY_BI”](#) on page 528—Evaluates the Airy function of the second kind.

[“IMSL_ALLBEST”](#) on page 630—Selects the best multiple linear regression models.

[“IMSL_ANOVA1”](#) on page 750—Analyzes one-way classification model.

[“IMSL_ANOVABALANCED”](#) on page 781—Balanced fixed, random, or mixed model.

[“IMSL_ANOVAFACT”](#) on page 760—Analyzes a balanced factorial design with fixed effects.

[“IMSL_ANOVANESTED”](#) on page 772—Nested random mode.

[“IMSL_ARMA”](#) on page 913—Computes method-of-moments or least-squares estimates of parameters for a nonseasonal ARMA model.

[“IMSL_AUTOCORRELATION”](#) on page 940—Sample autocorrelation function.

[“IMSL_BESSI”](#) on page 498—Evaluates a modified Bessel function of the first kind with real order and real or complex parameters.

[“IMSL_BESSI_EXP”](#) on page 506—Evaluates the exponentially scaled modified Bessel function of the first kind of orders zero and one.

[“IMSL_BESSJ”](#) on page 500—Evaluates a Bessel function of the first kind with real order and real or complex parameters.

[“IMSL_BESSK”](#) on page 502—Evaluates a modified Bessel function of the second kind with real order and real or complex parameters.

[“IMSL_BESSK_EXP”](#) on page 508—Evaluates the exponentially scaled modified Bessel function of the third kind of orders zero and one.

[“IMSL_BESSY”](#) on page 504—Evaluates a Bessel function of the second kind with real order and real or complex parameters.

[“IMSL_BETA”](#) on page 484—Evaluates the real beta function $B(x,y)$.

[“IMSL_BETACDF”](#) on page 1053—Evaluates the beta probability distribution function.

[“IMSL_BETAI”](#) on page 489—Evaluates the real incomplete beta function.

[“IMSL_BINOMIALCDF”](#) on page 1056—Evaluates the binomial distribution function.

[“IMSL_BINOMIALCOEF”](#) on page 1166—Evaluate binomial coefficient.

[“IMSL_BINOMIALPDF”](#) on page 1058—Evaluates the binomial probability function.

[“IMSL_BINORMALCDF”](#) on page 1035—Evaluates the bivariate normal distribution function.

[“IMSL_BOXCOXTRANS”](#) on page 935—Perform Box-Cox transformation

[“IMSL_BSINTERP”](#) on page 210—Computes a one- or two-dimensional spline interpolant.

[“IMSL_BSKNOTS”](#) on page 219—Computes the knots for a spline interpolant.

[“IMSL_BSLSQ”](#) on page 238—Computes a one- or two-dimensional, least-squares spline approximation.

[“IMSL_CAT_GLM” on page 817](#)—Generalized linear models.

[“IMSL_CHFAC” on page 93](#)—Computes the Cholesky factor, L , of a real or complex symmetric positive definite matrix A , such that $A = LL^T$.

[“IMSL_CHISQCDF” on page 1038](#)—Evaluates the chi-squared distribution function. Using a keyword, the inverse of the chi-squared distribution can be evaluated.

[“IMSL_CHISQCDF” on page 1038](#)—Evaluates the chi-squared distribution function. Using a keyword, the inverse of the chi-squared distribution can be evaluated.

[“IMSL_CHISQTEST” on page 876](#)—Performs a chi-squared goodness-of-fit test.

[“IMSL_CHNNDFAC” on page 112](#)—Computes the Cholesky factorization of the real matrix A such that $A = R^T R = LL^T$.

[“IMSL_CHNND SOL” on page 108](#)—Solves a real symmetric nonnegative definite system of linear equations $Ax = b$. Computes the solution to $Ax = b$ given the Cholesky factor.

[“IMSL_CHSOL” on page 89](#)—Solves a symmetric positive definite system of real or complex linear equations $Ax = b$.

[“IMSL_COCHRANQ” on page 865](#)—Cochran's Q test.

[“IMSL_CONLSQ” on page 248](#)—Computes a least-squares constrained spline approximation.

[“IMSL_CONSTANT” on page 1152](#)—Returns the value of various mathematical and physical constants.

[“IMSL_CONSTRAINED_NLP” on page 465](#)—Using a sequential equality constrained quadratic programming method.

[“IMSL_CONT_TABLE” on page 1118](#)—Sets up a table to generate pseudorandom numbers from a general continuous distribution.

[“IMSL_CONTINGENCY” on page 796](#)—Performs a chi-squared analysis of a two-way contingency table.

[“IMSL_CONVOLID” on page 390](#)—Computes the discrete convolution of two one dimensional arrays.

[“IMSL_CORRID” on page 395](#)—Compute the discrete correlation of two one-dimensional arrays.

[“IMSL_COVARIANCES” on page 722](#)—Computes the sample variance-covariance or correlation matrix.

[“IMSL_CSINTERP”](#) on page 200—Computes a cubic spline interpolant, specifying various endpoint conditions. The default interpolant satisfies the not-a-knot condition.

[“IMSL_CSSHAPE”](#) on page 205—Computes a shape-preserving cubic spline.

[“IMSL_CSSMOOTH”](#) on page 254—Computes a smooth cubic spline approximation to noisy data by using cross-validation to estimate the smoothing parameter or by directly choosing the smoothing parameter.

[“IMSL_CSTRENDS”](#) on page 849—Cox and Stuarts’ sign test for trends in location and dispersion.

[“IMSL_DATETODAYS”](#) on page 1150—Computes the number of days from January 1, 1900, to the given date.

[“IMSL_DAYSTODATE”](#) on page 1148—Gives the date corresponding to the number of days since January 1, 1900.

[“IMSL_DIFFERENCE”](#) on page 929—Differences a seasonal or nonseasonal time series.

[“IMSL_DISCR_ANALYSIS”](#) on page 992—Perform discriminant function analysis.

[“IMSL_DISCR_TABLE”](#) on page 1123—Sets or retrieves the current table used in either the shuffled or GFSR random number generator

[“IMSL_EIG”](#) on page 178—Computes the eigenexpansion of a real or complex matrix A . If the matrix is known to be symmetric or Hermitian, a keyword can be used to trigger more efficient algorithms.

[“IMSL_EIGSYMGEN”](#) on page 183—Computes the generalized eigenexpansion of a system $Ax = \lambda Bx$. The matrices A and B are real and symmetric, and B is positive definite.

[“IMSL_ELE”](#) on page 512—Evaluates the complete elliptic integral of the second kind $E(x)$.

[“IMSL_ELK”](#) on page 510—Evaluates the complete elliptic integral of the kind $K(x)$.

[“IMSL_ELRC”](#) on page 520—Evaluates an elementary integral from which inverse circular functions, logarithms and inverse hyperbolic functions can be computed.

[“IMSL_ELRD”](#) on page 516—Evaluates Carlson’s elliptic integral of the second kind $R_D(x, y, z)$.

[“IMSL_ELRF”](#) on page 514—Evaluates Carlson’s elliptic integral of the first kind $R_F(x, y, z)$.

[“IMSL_ELRJ”](#) on page 518—Evaluates Carlson’s elliptic integral of the third kind $R_J(x, y, z, r)$.

[“IMSL_ERF”](#) on page 477—Evaluates the real error function $\text{erf}(x)$. Using a keyword, the inverse error function $\text{erf}^{-1}(x)$ can be evaluated.

[“IMSL_ERFC”](#) on page 480—Evaluates the real complementary error function $\text{erfc}(x)$. Using a keyword, the inverse complementary error function $\text{erfc}^{-1}(x)$ can be evaluated.

[“IMSL_EXACT_ENUM”](#) on page 809—Exact probabilities in a table; total enumeration.

[“IMSL_EXACT_NETWORK”](#) on page 812—Exact probabilities in a table.

[“IMSL_FACTOR_ANALYSIS”](#) on page 981—Extracts initial factor-loading estimates in factor analysis.

[“IMSL_FAURE_INIT”](#) on page 1136—Initializes the structure used for computing a shuffled Faure sequence.

[“IMSL_FAURE_NEXT_PT”](#) on page 1140—Generates shuffled Faure sequence.

[“IMSL_FCDF”](#) on page 1043—Evaluates the F distribution function. Using a keyword, the inverse of the F distribution function can be evaluated.

[“IMSL_FCN_DERIV”](#) on page 326—Computes the first, second, or third derivative of a user-supplied function.

[“IMSL_FCNLSQ”](#) on page 234—Computes a least-squares fit using user-supplied functions.

[“IMSL_FFTCOMP”](#) on page 377—Computes discrete Fourier transform of a real or complex sequence. Using keywords, a real-to-complex transform or two-dimensional complex Fourier transform can be computed.

[“IMSL_FFTINIT”](#) on page 387—Computes parameters for a one-dimensional FFT to be used in the `IMSL_FFTCOMP` function with keyword `Init_Params`.

[“IMSL_FMIN”](#) on page 425—Finds the minimum point of a smooth function $f(x)$ of a single variable using function evaluations and, optionally, through both function evaluations and first derivative evaluations.

[“IMSL_FMINV”](#) on page 433—Minimizes a function $f(x)$ of n variables using a quasi-Newton method.

[“IMSL_FREQTABLE”](#) on page 563—Tallies observations into a one-way frequency table.

[“IMSL_FRESNEL_COSINE”](#) on page 522—Evaluates cosine Fresnel integral.

- “[IMSL_FRESNEL_SINE](#)” on page 524—Evaluates sine Fresnel integral.
- “[IMSL_FRIEDMANS_TEST](#)” on page 860—Friedman’s test.
- “[IMSL_GAMMA_ADV](#)” on page 493—Evaluate the real gamma function.
- “[IMSL_GAMMACDF](#)” on page 1050—Evaluates the gamma distribution function.
- “[IMSL_GAMMAI](#)” on page 495—Evaluate incomplete gamma function.
- “[IMSL_GARCH](#)” on page 952—Compute estimates of the parameters of a GARCH(p,q) model
- “[IMSL_GENEIG](#)” on page 186—Computes the generalized eigenexpansion of a system $Ax = \lambda Bx$.
- “[IMSL_GQUAD](#)” on page 322—Computes a Gauss, Gauss-Radau, or Gauss-Lobatto quadrature rule with various classical weight functions.
- “[IMSL_HYPERGEOCDF](#)” on page 1060—Evaluates the hypergeometric distribution function.
- “[IMSL_HYPOTH_PARTIAL](#)” on page 675—Constructs an equivalent completely testable multivariate general linear hypothesis $H\beta U = G$ from a partially testable hypothesis $H_p\beta U = G_p$.
- “[IMSL_HYPOTH_SCPH](#)” on page 681—Computes the matrix of sums of squares and crossproducts for the multivariate general linear hypothesis $H\beta U = G$ given the regression fit.
- “[IMSL_HYPOTH_TEST](#)” on page 686—Performs tests for a multivariate general linear hypothesis $H\beta U = G$ given the hypothesis sums of squares and crossproducts matrix S_H .
- “[IMSL_INTFCN](#)” on page 284—Integrates a user-supplied function using different combinations of keywords and parameters.
- “[IMSL_INTFCN_QMC](#)” on page 319—Integrates a function on a hyper-rectangle using a quasi-Monte Carlo method.
- “[IMSL_INTFCNHYPER](#)” on page 315—Integrates a function on a hyper-rectangle.
- “[IMSL_INV](#)” on page 77—Computes the inverse of a real or complex, square matrix.
- “[IMSL_K_MEANS](#)” on page 971—Performs a K-means (centroid) cluster analysis.
- “[IMSL_KALMAN](#)” on page 957—Performs Kalman filtering and evaluates the likelihood function or the state-space model.
- “[IMSL_KELVIN_BEI0](#)” on page 533—Evaluates the Kelvin function of the first kind, bei , of order zero.

- [“IMSL_KELVIN_BER0”](#) on page 531—Evaluates the Kelvin function of the first kind, ber, of order zero.
- [“IMSL_KELVIN_KEI0”](#) on page 537—Evaluates the Kelvin function of the second kind, kei, of order zero.
- [“IMSL_KELVIN_KER0”](#) on page 535—Evaluates the Kelvin function of the second kind, ker, of order zero.
- [“IMSL_KOLMOGOROV1”](#) on page 886—One-sample continuous data Kolmogorov-Smirnov.
- [“IMSL_KOLMOGOROV2”](#) on page 889—Two-sample continuous data Kolmogorov-Smirnov.
- [“IMSL_KTRENDS”](#) on page 868—K-sample trends test.
- [“IMSL_KW_TEST”](#) on page 857—Kruskal-Wallis test.
- [“IMSL_LACK_OF_FIT”](#) on page 948—Lack-of-fit test based on the correlation function
- [“IMSL_LAPLACE_INV”](#) on page 398—Computes the inverse Laplace transform of a complex function.
- [“IMSL_NLINLSQ”](#) on page 441—Solves a linear least-squares problem with linear constraints.
- [“IMSL_LINLSQ”](#) on page 116—Linear constraints
- [“IMSL_LINPROG”](#) on page 449—Solves a linear programming problem using the revised simplex algorithm.
- [“IMSL_LNBETA”](#) on page 487—Evaluate the log of the real beta function.
- [“IMSL_LNGAMMA”](#) on page 491—Evaluate the logarithm of the absolute value of the gamma function.
- [“IMSL_LNORMREGRESS”](#) on page 703—Fits a multiple linear regression model using criteria other than least squares.
- [“IMSL_LUFAC”](#) on page 85—Computes the LU factorization of a real or complex matrix.
- [“IMSL_LUSOL”](#) on page 79—Solves a general system of real or complex linear equations $Ax = b$.
- [“IMSL_MACHINE”](#) on page 1158—Returns information describing the computer’s arithmetic.

[“IMSL_MATRIX_NORM”](#) on page 1171—Computes various norms of a rectangular matrix, a matrix stored in band format, and a matrix stored in coordinate format.

[“IMSL_MINCONGEN”](#) on page 458—Minimizes a general objective function subject to linear equality/inequality constraints.

[“IMSL_MULTICOMP”](#) on page 769—Performs Student-Newman-Keuls multiple-comparisons test.

[“IMSL_MULTIPREDICT”](#) on page 622—Computes predicted values, confidence intervals, and diagnostics after fitting a regression model.

[“IMSL_MULTIREGRESS”](#) on page 607—Fits a multiple linear regression model using least squares and optionally compute summary statistics for the regression model.

[“IMSL_MVAR_NORMALITY”](#) on page 892—Mardia’s test for multivariate normality.

[“IMSL_NCTRENDS”](#) on page 846—Noether’s test for cyclical trend.

[“IMSL_NLINLSQ”](#) on page 441—Solves a nonlinear least-squares problem using a modified Levenberg-Marquardt algorithm.

[“IMSL_NONLINOPT”](#) on page 694—Fits data to a nonlinear model (possibly with linear constraints) using the successive quadratic programming algorithm (applied to the sum of squared errors, $sse = \sum (y_i - f(x_i; \theta))^2$) and either a finite difference gradient or a user-supplied gradient.

[“IMSL_NONLINREGRESS”](#) on page 665—Fits a nonlinear regression model.

[“IMSL_NORM”](#) on page 1168—Computes various norms of a vector or the difference of two vectors.

[“IMSL_NORMALCDF”](#) on page 1032—Evaluates the standard normal (Gaussian) distribution function. Using a keyword, the inverse of the standard normal (Gaussian) distribution can be evaluated.

[“IMSL_NORM1SAMP”](#) on page 550—Computes statistics for mean and variance inferences using a sample from a normal population.

[“IMSL_NORM2SAMP”](#) on page 555—Computes statistics for mean and variance inferences using samples from two independently normal populations.

[“IMSL_NORMALITY”](#) on page 882—Performs a test for normality.

[“IMSL_ODE”](#) on page 333—Adams-Gear or Runge-Kutta method.

[“IMSL_PARTIAL_AC”](#) on page 945—Sample partial autocorrelation function

[“IMSL_PARTIAL_COV”](#) on page 728—Partial correlations and covariances.

[“IMSL_PDE_MOL”](#) on page 351—Solves a system of partial differential equations of the form $ut = f(x, t, u, u_x, u_{xx})$ using the method of lines. The solution is represented with cubic Hermite polynomials.

[“IMSL_POISSON2D”](#) on page 366—Solves Poisson’s or Helmholtz’s equation on a two-dimensional rectangle using a fast Poisson solver based on the HODIE finite-difference scheme on a uniform mesh.

[“IMSL_POISSONCDF”](#) on page 1063—Evaluates the Poisson distribution function.

[“IMSL_POLYPREDICT”](#) on page 657—Computes predicted values, confidence intervals, and diagnostics after fitting a polynomial regression model.

[“IMSL_POLYREGRESS”](#) on page 649—Performs a polynomial least-squares regression.

[“IMSL_POOLED_COV”](#) on page 734—Pooled covariance matrix.

[“IMSL_PRINC_COMP”](#) on page 976—Computes principal components.

[“IMSL_QRFAC”](#) on page 100—Computes the QR factorization of a real matrix A .

[“IMSL_QRSOL”](#) on page 96—Solves a real linear least-squares problem $Ax = b$.

[“IMSL_QUADPROG”](#) on page 454—Solves a quadratic programming (QP) problem subject to linear equality or inequality constraints.

[“IMSL_RADBE”](#) on page 277—Evaluates a radial-basis fit computed by `IMSL_RADBF`.

[“IMSL_RADBF”](#) on page 266—Computes an approximation to scattered data in R^n for $n \geq 2$ using radial-basis functions.

[“IMSL_RANDOM_FROM_DATA”](#) on page 1115—Generates pseudorandom numbers from multivariate distribution determined from a given sample.

[“IMSL_RANDOM_GEN_CONT”](#) on page 1120—Generates pseudorandom numbers from a general continuous distribution.

[“IMSL_RANDOM_GEN_DISCR”](#) on page 1127—Generates pseudorandom numbers from a general discrete distribution using an alias method or optionally a table lookup method.

[“IMSL_RANDOM_ORTH_MAT”](#) on page 1109—Generates a pseudorandom orthogonal matrix or a correlation matrix

[“IMSL_RANDOM_TABLE_2WAY”](#) on page 1107—Generates a pseudorandom two-way table.

[“IMSL_RANDOM”](#) on page 1080—Generates pseudorandom numbers. The default distribution is a uniform (0, 1) distribution, but many different distributions can be specified through the use of keywords.

[“IMSL_RANDOM_ARMA”](#) on page 1131—Generate pseudorandom IMSL_ARMA process numbers

[“IMSL_RANDOM_NPP”](#) on page 1100—Generates pseudorandom numbers from a nonhomogeneous Poisson process.

[“IMSL_RANDOM_ORDER”](#) on page 1104—Generates pseudorandom order statistics from a standard normal distribution.

[“IMSL_RANDOM_SAMPLE”](#) on page 1112—Generates a simple pseudorandom sample from a finite population

[“IMSL_RANDOM_TABLE”](#) on page 1076—Sets or retrieves the current table used in either the shuffled or GFSR random number generator.

[“IMSL_RANDOMNESS_TEST”](#) on page 897—Runs test, Paris-serial test, d^2 test or triplets tests.

[“IMSL_RANDOMOPT”](#) on page 1071—Uses keywords to set or retrieve the random number seed or to select the uniform (0, 1) multiplicative, congruential pseudorandom-number generator.

[“IMSL_RANKS”](#) on page 577—Computes the ranks, normal scores, or exponential scores for a vector of observations.

[“IMSL_REGRESSORS”](#) on page 600—Generates regressors for a general linear model.

[“IMSL_ROBUST_COV”](#) on page 738—Robust estimate of covariance matrix.

[“IMSL_SCAT2DINTERP”](#) on page 262—Computes a smooth bivariate interpolant to scattered data that is locally a quintic polynomial in two variables.

[“IMSL_SIGNTEST”](#) on page 834—Performs a sign test.

[“IMSL_SIMPLESTAT”](#) on page 544—Computes basic univariate statistics.

[“IMSL_SMOOTHDATA1D”](#) on page 258—Smooth one-dimensional data by error detection.

[“IMSL_SORTDATA”](#) on page 570—Sorts observations by specified keys, with option to tally cases into a multiway frequency table.

[“IMSL_SP_BDFAC”](#) on page 139—Compute the LU factorization of a matrix stored in band storage mode.

“[IMSL_SP_BDPDFAC](#)” on page 155—Compute the $R^T R$ Cholesky factorization of symmetric positive definite matrix, A , in band symmetric storage mode.

“[IMSL_SP_BDPDSOL](#)” on page 152—Solve a symmetric positive definite system of linear equations $Ax = b$ in band symmetric storage mode.

“[IMSL_SP_BDSOL](#)” on page 135—Solve a general band system of linear equations $Ax = b$.

“[IMSL_SP_CG](#)” on page 163—Solve a real symmetric definite linear system using a conjugate gradient method. Using keywords, a preconditioner can be supplied.

“[IMSL_SP_GMRES](#)” on page 159—Solve a linear system $Ax = b$ using the restarted generalized minimum residual (GMRES) method.

“[IMSL_SP_LUFAC](#)” on page 128—Compute an LU factorization of a sparse matrix stored in either coordinate format or CSC format.

“[IMSL_SP_LUSOL](#)” on page 122—Solve a sparse system of linear equations $Ax = b$.

“[IMSL_SP_MVMUL](#)” on page 167—Compute a matrix-vector product involving sparse matrix and a dense vector.

“[IMSL_SP_PDFAC](#)” on page 148—Solve a sparse symmetric positive definite system of linear equations $Ax = b$.

“[IMSL_SP_PDSOL](#)” on page 143—Solve a sparse symmetric positive definite system of linear equations $Ax = b$.

“[IMSL_SPINTEG](#)” on page 230—Computes the integral of a one- or two-dimensional spline.

“[IMSL_SPVALUE](#)” on page 224—Computes values of a spline or values of one of its derivatives.

“[IMSL_STATDATA](#)” on page 1163—Retrieves commonly analyzed data sets.

“[IMSL_STEPWISE](#)” on page 639—Builds multiple linear regression models using forward, backward, or stepwise selection.

“[IMSL_SURVIVAL_GLM](#)” on page 1006—Analyzes survival data using a generalized linear model and estimates using various parametric modes.

“[IMSL_SVDCOMP](#)” on page 104—Computes the singular value decomposition (SVD), $A=USV^T$, of a real or complex rectangular matrix A . An estimate of the rank of A also can be computed.

“[IMSL_TCDF](#)” on page 1046—Evaluates the Student’s t distribution function.

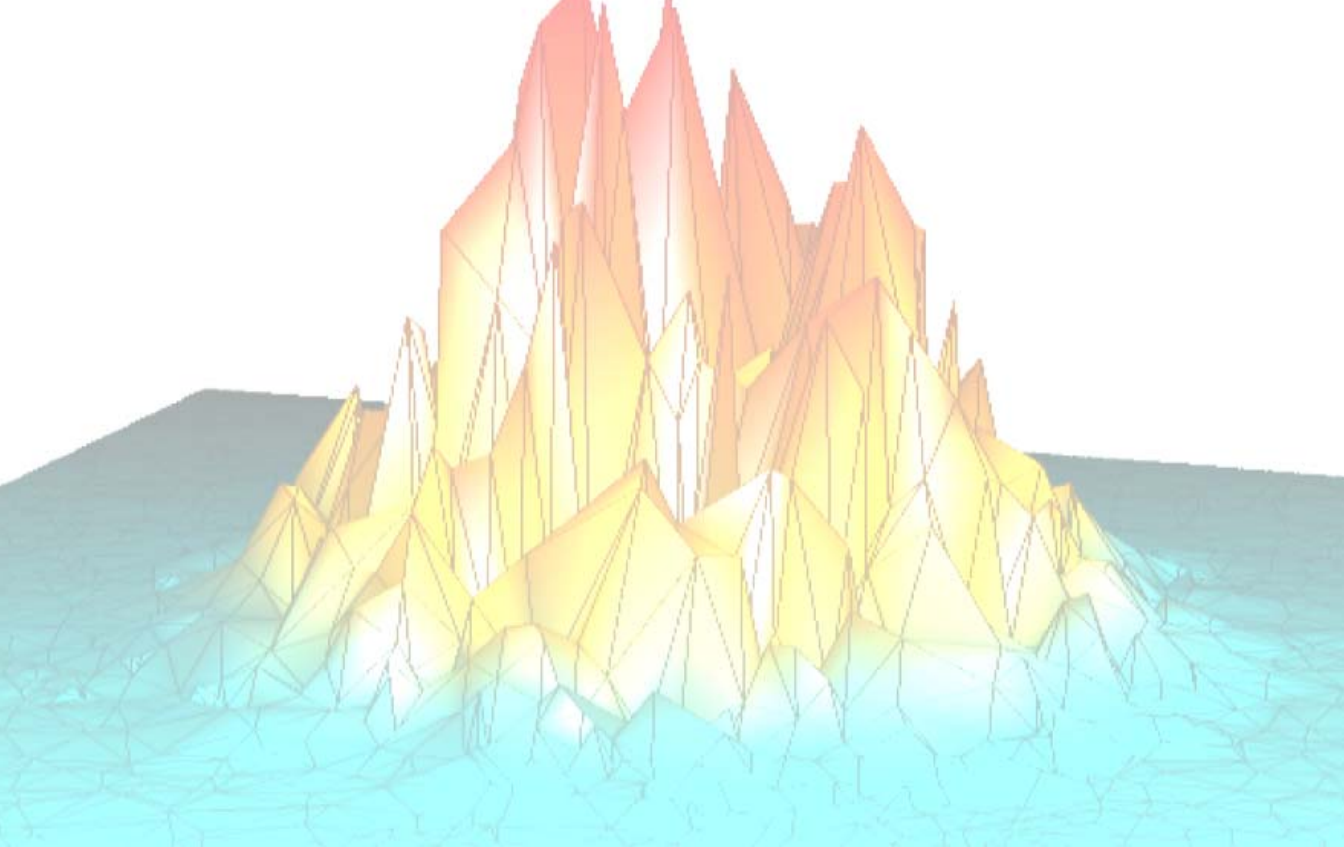
[“IMSL_TIE_STATS” on page 855](#)—Tie statistics.

[“IMSL_WILCOXON” on page 837](#)—Performs a Wilcoxon rank sum test.

[“IMSL_ZEROFCN” on page 413](#)—Finds the real zeros of a real function using Müller’s method.

[“IMSL_ZEROPOLY” on page 410](#)—Finds the zeros of a polynomial with real or complex coefficients using the companion matrix method or, optionally, the Jenkins-Traub, three-stage algorithm.

[“IMSL_ZEROSYS” on page 418](#)—Solves a system of n nonlinear equations using a modified Powell hybrid algorithm.



Part I: Mathematics Routines



Chapter 4

Linear Systems

This section contains the following topics:

Overview: Linear Systems	64	Linear Systems Routines	75

Overview: Linear Systems

This section introduces some of the mathematical concepts used with IDL Advanced Math and Stats.

Matrix Inversion

The [IMSL_INV](#) function inverts an $n \times n$ nonsingular matrix—either real or complex. The inverse also can be obtained by using the INVERSE keyword in functions for solving systems of linear equations. You do not need to compute the inverse of a matrix if the purpose is to solve one or more systems of linear equations. Even with multiple right-hand sides, solving a system of linear equations by computing the inverse and performing matrix multiplication is usually more expensive than the method discussed in the next section.

Solving Systems of Linear Equations

A square system of linear equations has the form $Ax = b$, where A is a user-specified $n \times n$ matrix, b is a given n -vector, and x is the solution n -vector. You must specify each entry of A and b . The entire vector x is returned as output.

When A is invertible, a unique solution to $Ax = b$ exists. The most commonly used direct method for solving $Ax = b$ factors the matrix A into a product of triangular matrices and solves the resulting triangular systems of linear equations. Functions that use direct methods for solving systems of linear equations all compute the solution to $Ax = b$. You can use IDL Advanced Math and Stats functions [IMSL_LUSOL](#), [IMSL_CHSOL](#), and [IMSL_CHNDSOL](#) to compute x .

Matrix Factorizations

In some applications, you may only want to factor the $n \times n$ matrix A into a product of two triangular matrices. Functions and procedures that end with “FAC” are designed to compute these factorizations. Suppose that in addition to the solution x of a linear system of equations $Ax = b$, you want the LU factorization of A . First, use the [IMSL_LUFAC](#) procedure to obtain the LU factorization in a condensed format, then call [IMSL_LUSOL](#) with this factorization and a right-hand side b to compute the solution. If the factorization is desired in separate, full matrices, call the [IMSL_LUFAC](#) procedure with the keywords `L` and `U` to return L and U separately.

Besides the basic matrix factorizations, such as LU and LL^T , additional matrix factorizations also are provided. For a real matrix A , QR factorization can be computed by the [IMSL_QRFAC](#) procedure. Functions for computing the Singular

Value Decomposition (SVD) of a matrix are discussed in “[Singular Value Decomposition and Generalized Inverse](#)” on page 65.

Multiple Right-hand Sides

In a case in which a system of linear equations has more than one right-hand side vector, it is most economical to find the solution vectors by first factoring the coefficient matrix A into products of triangular matrices. Then, the resulting triangular systems of linear equations are solved for each right-hand side. When A is a real general matrix, compute access to the LU factorization of A by using the [IMSL_LUFAC](#) procedure. The solution x_k for the k -th right-hand side vector b_k is then found by two triangular solves, $Ly_k = b_k$ and $Ux_k = y_k$. The [IMSL_LUSOL](#) function is called with the computed factorization and is used to solve each right-hand side. You can follow this process when using other functions for solving systems of linear equations.

Least-squares Solution and QR Factorization

Least-squares solutions are usually computed for an over-determined system of linear equations $A_{m \times n} x = b$, where $m > n$. A least-squares solution x minimizes the Euclidean length of the residual vector $r = Ax - b$. The [IMSL_QRSOL](#) function computes a unique least-squares solution for x when A has full-column rank. If A is rank-deficient, then the *base* solution for some variables is computed. These variables consist of the resulting columns after the interchanges. The QR decomposition, with column interchanges or pivoting, is computed such that $AP = QR$. Here, Q is orthogonal, R is upper-trapezoidal with its diagonal elements nonincreasing in magnitude, and P is the permutation matrix determined by the pivoting. The base solution x_B is obtained by solving $R(P^T)x = Q^T b$ for the base variables. For details, see “[Discussion](#)” on page 98. You can compute the QR factorization of a matrix A , such that $AP = QR$ with a user-specified P , using the [IMSL_QRFAC](#) procedure.

Singular Value Decomposition and Generalized Inverse

The SVD of an m by n matrix A is a matrix decomposition, $A = USV^T$. With $q = \min(m, n)$, the factors $U_{m \times q}$ and $V_{n \times q}$ are orthogonal matrices, and $S_{q \times q}$ is a nonnegative diagonal matrix with nonincreasing diagonal terms. The [IMSL_SVDCOMP](#) function computes the singular values of A by default. By using keywords, you can also obtain part or all of the U and V matrices, an estimate of the rank of A , and the generalized inverse of A .

III-conditioning and Singularity

An $m \times n$ matrix A is mathematically singular if an $x \neq 0$ exists such that $Ax = 0$. In this case, the system of linear equations $Ax = b$ does not have a unique solution. However, a matrix A is *numerically* singular if it is “close” to a mathematically singular matrix. Such problems are called *ill-conditioned*. If the numerical results with an ill-conditioned problem are unacceptable, either use more accuracy if available (for type *float* switch to *double*) or obtain an approximate solution to the system. One form of approximation can be obtained using the SVD of A : If $q = \min(m, n)$ and:

$$A = \sum_{i=1}^q s_{i,i} u_i v_i^T$$

then the approximate solution is given by the following:

$$x_k = \sum_{i=1}^k t_{i,i} (b^T u_i) v_i$$

The scalars $t_{i,i}$ are defined by:

$$t_{i,i} = \begin{cases} s_{i,i}^{-1} & \text{if } s_{i,i} \geq \text{tol} \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Specify the value of *tol*. This value determines how “close” the given matrix is to a singular matrix. Further restrictions may apply to the number of terms in the sum, $k \leq q$. For example, there may be a value of $k \leq q$ such that the scalars $|(b^T u_i)|$, $i > k$, are smaller than the average uncertainty in the right-hand side b . This means that these scalars can be replaced by zero, and b is replaced by a vector that is within the stated uncertainty of the problem.

Notation

Since many functions and procedures described in this chapter operate on both real or complex matrices, the notation A^H is used to represent both the transpose of A if A is real and the conjugate transpose if A is complex.

Sparse Matrices: Direct Methods

Several routines employ direct methods (as opposed to iterative methods) for solving problems involving sparse matrices.

For general sparse linear systems, `IMSL_SP_LUFAC` and `IMSL_SP_LUSOL` form a factor/solve function pair. If a sparse matrix the problem $Ax = b$ is to be solved for a single A , but multiple right-hand sides, b , then `IMSL_SP_LUFAC` should first be used to compute an LU decomposition of A , then follow multiple calls to `IMSL_SP_LUSOL` (one for each right-hand side, b). If only one right-hand side, b , is involved then `IMSL_SP_LUSOL` can be called directly, in which case the factor step is computed internally by `IMSL_SP_LUSOL`.

For general banded systems, `IMSL_SP_BDSOL` and `IMSL_SP_BDFAC` form a factor/solve pair. The relationship between `SP_BSFAC` and `IMSL_SP_BDSOL` is similar to that of `IMSL_SP_LUFAC` and `IMSL_SP_LUSOL`.

The functions `IMSL_SP_PDFAC` and `IMSL_SP_PDSOL` are provided for working with systems involving sparse symmetric positive definite matrices. The relationship between `IMSL_SP_PDFAC` and `IMSL_SP_PDSOL` is similar to that of `IMSL_SP_LUFAC` and `IMSL_SP_LUSOL`.

The functions `SP_BDDFAC` and `IMSL_SP_BDPDSOL` are provided for working with systems involving banded symmetric positive definite matrices. The relationship between `IMSL_SP_BDPDFAC` and `IMSL_SP_BDPDSOL` is similar to that of `IMSL_SP_LUFAC` and `IMSL_SP_LUSOL`.

- `IMSL_SP_LUFAC`— LU factorization of general matrices.
- `IMSL_SP_LUSOL`—Systems involving general matrices.
- `IMSL_SP_BDFAC`— LU factorization of band matrices.
- `IMSL_SP_BDSOL`—Systems involving band matrices.
- `IMSL_SP_PDFAC`—Factorization of symmetric positive definite matrices.
- `IMSL_SP_PDSOL`—Systems involving symmetric positive definite matrices.
- `IMSL_SP_BDPDFAC`—Cholesky factorization of symmetric positive definite matrices in band symmetric storage mode.
- `IMSL_SP_BDPDSOL`— Systems involving symmetric positive definite matrices in band symmetric storage mode

Sparse Matrices: Iterative Methods

Two routines employ iterative methods (as opposed to direct methods) for solving problems involving sparse matrices.

The `IMSL_SP_GMRES` function, based on the FORTRAN subroutine `GMRES` by H. F. Walker, solves the linear system $Ax = b$ using the GMRES method. This method is described in detail by Saad and Schultz (1986) and Walker (1988).

The `IMSL_SP_CG` function solves the symmetric definite linear system $Ax = b$ using the conjugate gradient method with optional preconditioning. This method is described in detail by Golub and Van Loan (1983, chapter 10), and in Hageman and Young (1981, chapter 7).

- [IMSL_SP_GMRES](#)—Restarted generalized minimum residual (GMRES) method.
- [IMSL_SP_CG](#)—Conjugate gradient method.

Sparse Matrices: Utilities

Utilities designed to aid with the manipulation of sparse matrices are also provided. The common operation of matrix-vector multiplication can be efficiently executed using the [IMSL_SP_MVMUL](#) function.

Sparse Matrices: Matrix Storage Modes

The dense linear algebra functions in IDL Advanced Math and Stats require input consisting of matrix dimensions and all values for the matrix entries. This is not practical for sparse linear algebra. Three different storage formats can be used for the functions in the sparse matrix sections. These methods include:

- [Sparse Coordinate Storage Format](#)
- [Band Storage Format](#)
- [Compressed Sparse Column \(CSC\) Format](#)

Sparse Coordinate Storage Format

Only the non-zero elements of a sparse matrix need to be communicated to a function. Sparse coordinate storage format stores the value of each matrix entry along with that entry's row and column index. The following structures are defined by the IDL Advanced Math and Stats module to make it easy to store sparse matrices:


```
{imsl_f_sp_elem, row:0L, col:0L, val:float(0.0)}
{imsl_d_sp_elem, row:0L, col:0L, val:double(0.0)}
{imsl_c_sp_elem, row:0L, col:0L, val:complex(0.0)}
{imsl_z_sp_elem, row:0L, col:0L, val:dcomplex(0.0)}
```

As an example consider the 6 x 6 matrix:

$$A = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 9 & -3 & -1 & 0 & 0 \\ 0 & 0 & 5 & 0 & 0 & 0 \\ -2 & 0 & 0 & -7 & -1 & 0 \\ -1 & 0 & 0 & -5 & 1 & -3 \\ -1 & -2 & 0 & 0 & 0 & 6 \end{bmatrix}$$

The matrix A has 15 non-zero elements, and its sparse coordinate representation would be:

row	0	1	1	1	2	3	3	3	4	4	4	4	5	5	5
col	0	1	2	3	2	0	3	4	0	3	4	5	0	1	5
val	2	9	-3	-1	5	-2	-7	-1	-1	-5	1	-3	-1	-2	6

Since this representation does not rely on order, an equivalent form would be:

row	5	4	3	0	5	1	2	1	4	3	1	4	3	5	4
col	0	0	0	0	1	1	2	2	3	3	3	4	4	5	5
val	-1	-1	-2	2	-2	9	5	-3	-5	-7	-1	1	-1	6	-3

There are different ways this data could be used to initialize. Consider the following program fragment:

```
A = replicate(imsl_f_sp_elem, 15)
a(*).row = [0, 1, 1, 1, 2, $
            3, 3, 3, 4, 4, $
            4, 4, 5, 5, 5]
a(*).col = [0, 1, 2, 3, 2, $
            0, 3, 4, 0, 3, $
            4, 5, 0, 1, 5]
```

```

a(*).val = [2, 9, -3, -1, 5, $
            -2, -7, -1, -1, -5, 1, $
            -3, -1, -2, 6]
B = replicate(ims1_f_sp_elem, 15)
b(*).row = [5, 4, 3, 0, 5, $
            1, 2, 1, 4, 3, $
            1, 4, 3, 5, 4]
b(*).col = [0, 0, 0, 0, 1, $
            1, 2, 2, 3, 3, $
            3, 4, 4, 5, 5]
b(*).val = [-1, -1, -2, 2, -2, $
            9, 5, -3, -5, -7, -1, $
            1, -1, 6, -3]

```

Both a and b represent the sparse matrix A.

A sparse symmetric or Hermitian matrix is a special case, since it is only necessary to store the diagonal and either the upper or lower triangle. As an example, consider the 5 x 5 linear system:

$$H = \begin{bmatrix} 4 & 1-i & 0 & 0 \\ 1+i & 4 & 1-i & 0 \\ 0 & 1+i & 4 & 1-i \\ 0 & 0 & 1+i & 4 \end{bmatrix}$$

The Hermitian and symmetric positive definite system solvers in this module expect the diagonal and lower triangle to be specified. The sparse coordinate form for the lower triangle is given by

row	0	1	2	3	1	2	3
col	0	1	2	3	0	1	2
val	(4,0)	(4,0)	(4,0)	(4,0)	(1,1)	(1,1)	(1,1)

The following program fragment will initialize H.

```

A = replicate(ims1_c_sp_elem, 7)
a(*).row = [0, 1, 2, 3, 1, 2, 3]
a(*).col = [0, 1, 2, 3, 0, 1, 2]
a(*).val = [COMPLEX(4, 0), COMPLEX(4, 0), $
            COMPLEX(4, 0), COMPLEX(4, 0), $

```

```
COMPLEX(1, 1), COMPLEX(1, 1), $
COMPLEX(1, 1)]
```

There are some important points to note here. Note that H is not symmetric, but rather Hermitian. The functions that accept Hermitian data understand this and operate assuming that:

$$h_{ij} = \bar{h}_{ji}$$

The Sparse Matrix Module cannot take advantage of the symmetry in matrices that are not positive definite. A symmetric matrix that happens to be indefinite cannot be stored in this compact symmetric form. Rather, both upper and lower triangles must be specified and the sparse general solver called.

Band Storage Format

A band matrix is an $M \times N$ matrix A with all of its non-zero elements “close” to the main diagonal. Specifically, values $A_{ij} = 0$ if $i - j > nlca$ or $j - i > nuca$. The integer $m = nlca + nuca + 1$ is the total band width. The diagonals, other than the main diagonal, are called codiagonals. While any $M \times N$ matrix is a band matrix, band storage format is only useful when the number of non-zero codiagonals is much less than N .

In band storage format, the $nlca$ lower codiagonals and the $nuca$ upper codiagonals are stored in the rows of an array of size $m \times N$. The elements are stored in the same column of the array as they are in the matrix. The values A_{ij} inside the band width are stored in the linear array in positions $[(i - j + nuca + 1) * i + j]$. This results in a row-major, one-dimensional mapping from the two-dimensional notion of the matrix.

For example, consider the 5×5 matrix A with 1 lower and 2 upper codiagonals:

$$A = \begin{bmatrix} A_{0,0} & A_{0,1} & A_{0,2} & 0 & 0 \\ A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} & 0 \\ 0 & A_{2,1} & A_{2,2} & A_{2,3} & A_{2,4} \\ 0 & 0 & A_{3,2} & A_{3,3} & A_{3,4} \\ 0 & 0 & 0 & A_{4,3} & A_{4,4} \end{bmatrix}$$

In band storage format, the data would be arranged as:

$$\begin{bmatrix} 0 & 0 & A_{0,2} & A_{1,3} & A_{2,4} \\ 0 & A_{0,1} & A_{1,2} & A_{2,3} & A_{3,4} \\ A_{0,0} & A_{1,1} & A_{2,2} & A_{3,3} & A_{4,4} \\ A_{1,0} & A_{2,1} & A_{3,2} & A_{4,3} & 0 \end{bmatrix}$$

This data would then be stored contiguously, row-major order, in an array of length 20.

As an example, consider the following tridiagonal matrix:

$$A = \begin{bmatrix} 10 & 1 & 0 & 0 & 0 \\ 5 & 20 & 2 & 0 & 0 \\ 0 & 6 & 30 & 3 & 0 \\ 0 & 0 & 7 & 40 & 4 \\ 0 & 0 & 0 & 8 & 50 \end{bmatrix}$$

The following code segment will store this matrix in band storage format:

```
a = [0, 1, 2, 3, 4, $
      10, 20, 30, 40, 50, $
      5, 6, 7, 8, 0]
```

As in the sparse coordinate representation, there is a space saving symmetric version of band storage. As an example, we look at the following 5 x 5 symmetric problem:

$$A = \begin{bmatrix} A_{0,0} & A_{0,1} & A_{0,2} & 0 & 0 \\ A_{0,1} & A_{1,1} & A_{1,2} & A_{1,3} & 0 \\ A_{0,2} & A_{1,2} & A_{2,2} & A_{2,3} & A_{2,4} \\ 0 & A_{1,3} & A_{2,3} & A_{3,3} & A_{3,4} \\ 0 & 0 & A_{2,4} & A_{3,4} & A_{4,4} \end{bmatrix}$$

In band symmetric storage format, the data would be arranged as:

$$\begin{bmatrix} 0 & 0 & A_{0,2} & A_{1,3} & A_{2,4} \\ 0 & A_{0,1} & A_{1,2} & A_{2,3} & A_{3,4} \\ A_{0,0} & A_{1,1} & A_{2,2} & A_{3,3} & A_{4,4} \end{bmatrix}$$

The following Hermitian example illustrates the procedure:

$$H = \begin{bmatrix} 8 & 1+i & 1+i & 0 & 0 \\ 1-i & 8 & 1+i & 1+i & 0 \\ 1-i & 1-i & 8 & 1+i & 1+i \\ 0 & 1-i & 1-i & 8 & 1+i \\ 0 & 0 & 1-i & 1-i & 8 \end{bmatrix}$$

The following program fragments stores H in h , using band symmetric storage format:

```
h = complexarr(15)
h(0:1) = 0
h(2:4) = complex(1,1)
h(5) = 0
h(6:9) = complex(1,1)
h(10:14) = 8
```

Choosing Between Banded and Coordinate Forms

Any matrix can be stored in either sparse coordinate or band format; which format to use depends on the sparsity pattern of the matrix. A matrix with all non-zero data stored in bands close to the main diagonal is probably a good candidate for band format. If non-zero information is scattered more or less uniformly through the matrix, sparse coordinate format is the best choice. The following two cases are extreme examples. First, consider an $n \times n$ matrix with all elements on the main diagonal and the $(0, n-1)$ and $(n-1, 0)$ entries non-zero. The sparse coordinate vector would be $n+2$ units long. An array of length $2n^2 - n$ would be required to store the band representation, nearly twice as much storage as a dense solver might require. Second, consider a tridiagonal matrix with all diagonal, superdiagonal and subdiagonal entries non-zero. In band format, an array of length $3n$ is needed. In sparse coordinate format, a vector of length $3n - 2$ is required. But the problem is that, for instance with floating-point precision on a 32 bit machine, each of those $3n -$

2 units in coordinate format requires three times as much storage as any of the $3n$ units needed for band representation. This is due to carrying the row and column indices in coordinate form. Band storage evades this requirement by being essentially an ordered list, and defining location in the original matrix by position in the list.

Compressed Sparse Column (CSC) Format

Functions that accept data in coordinate format can also accept data stored in the format described in the *Users' Guide for the Harwell-Boeing Sparse Matrix Collection*. The scheme is column oriented, with each column held as a sparse vector, represented by a list of the row indices of the entries in an integer array and a list of the corresponding values in a separate *float (double, complex, dcomplex)* array. Data for each column are stored consecutively and in order. A separate integer array holds the location of the first entry of each column and the first free location. Only entries in the lower triangle and diagonal are stored for symmetric and Hermitian matrices. All arrays are based at zero, which is in contrast to the Harwell-Boeing test suite's one-based arrays.

As in the *Users' Guide for the Harwell-Boeing Sparse Matrix Collection*, we illustrate the storage scheme with the following example. The 5x5 matrix:

$$\begin{bmatrix} 1 & -3 & 0 & -1 & 0 \\ 0 & 0 & -2 & 0 & 3 \\ 2 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & -4 & 0 \\ 5 & 0 & -5 & 0 & 6 \end{bmatrix}$$

would be stored in the arrays `colptr` (location of first entry), `rowind` (row indices), and `values` (non-zero entries) as follows:

Subscripts	0	1	2	3	4	5	6	7	8	9	10
colptr	0	3	5	7	9	11					
rowind	0	4	2	3	0	1	4	0	3	4	1
values	1	5	2	4	-3	-2	-5	-1	-4	6	3

Linear Systems Routines

Matrix Inversion

[IMSL_INV](#)—General matrix inversion.

Linear Equations with Full Matrices

[IMSL_LUSOL](#)—Systems involving general matrices.

[IMSL_LUFAC](#)— LU factorization of general matrices.

[IMSL_CHSOL](#)—Systems involving symmetric positive definite matrices.

[IMSL_CHFAC](#)—Factorization of symmetric positive definite matrices.

Linear Least Squares with Full Matrices

[IMSL_QRSOL](#)—Least-squares solution.

[IMSL_QRFAC](#)—Least-squares factorization.

[IMSL_SVDCOMP](#)—Singular Value Decomposition (SVD) and generalized inverse.

[IMSL_CHNDSOL](#)—Solve and generalized inverse for positive semidefinite matrices.

[IMSL_CHNDFAC](#)—Factor and generalized inverse for positive semidefinite matrices.

[IMSL_LINLSQ](#)—Linear constraints.

Sparse Matrices

[IMSL_SP_LUSOL](#)—Solve a sparse system of linear equations $Ax = b$.

[IMSL_SP_LUFAC](#)—Compute an LU factorization of a sparse matrix stored in either coordinate format or CSC format.

[IMSL_SP_BDSOL](#)—Solve a general band system of linear equations $Ax = b$.

[IMSL_SP_BDFAC](#)—Compute the LU factorization of a matrix stored in band storage mode.

[IMSL_SP_PDSOL](#)—Solve a sparse symmetric positive definite system of linear equations $Ax = b$.

IMSL_SP_PDFAC—Compute a factorization of a sparse symmetric positive definite system of linear equations $Ax = b$.

IMSL_SP_BDPDSOL—Solve a symmetric positive definite system of linear equations $Ax = b$ in band symmetric storage mode.

IMSL_SP_BDPDFAC—Compute the $R^T R$ Cholesky factorization of symmetric positive definite matrix, A , in band symmetric storage mode.

IMSL_SP_GMRES—Solve a linear system $Ax = b$ using the restarted generalized minimum residual (GMRES) method.

IMSL_SP_CG—Solve a real symmetric definite linear system using a conjugate gradient method.

IMSL_SP_MVMUL—Compute a matrix-vector product involving a sparse matrix and a dense vector.

IMSL_INV

The IMSL_INV function computes the inverse of a real or complex, square matrix.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your Excelis VIS sales or technical support representative.

Syntax

Result = IMSL_INV(*a* [, /DOUBLE])

Return Value

A two-dimensional matrix containing the inverse of the matrix *A*.

Arguments

a

Two-dimensional matrix containing the matrix to be inverted.

Keywords

DOUBLE

If present and nonzero, double precision is used.

Example

```
RM, a, 3, 3
; Define the matrix to be inverted.
row 0: 1 3 3
row 1: 1 3 4
row 2: 1 4 4
ainv = IMSL_INV(a)
; Call IMSL_INV to perform the inversion.
PM, a
; Output the original matrix.
1.00000      3.00000      3.00000
1.00000      3.00000      4.00000
```

```
1.00000      4.00000      4.00000

PM, ainv
; Output the computed inverse.
4.00000      -0.00000      -3.00000
0.00000      -1.00000      1.00000
-1.00000      1.00000      0.00000
PM, a # ainv
; Check the results.
1.00000      0.00000      0.00000
0.00000      1.00000      0.00000
0.00000      0.00000      1.00000
```

Errors

Fatal Errors

MATH_SINGULAR_MATRIX—Input matrix is singular.

Version History

6.4	Introduced

IMSL_LUSOL

The IMSL_LUSOL function solves a general system of real or complex linear equations $Ax = b$.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_LUSOL(b [, a] [, CONDITION=variable] [, /DOUBLE]
    [, FACTOR=variable] [, INVERSE=variable] [, PIVOT=variable]
    [, TRANSPOSE=value])
```

Return Value

A one-dimensional array containing the solution of the linear system $Ax = b$.

Arguments

b

One-dimensional matrix containing the right-hand side.

a

Two-dimensional matrix containing the coefficient matrix. Element $A(i, j)$ contains the j -th coefficient of the i -th equation.

Keywords

CONDITION

Named variable into which an estimate of the L_1 condition number is stored. This keyword cannot be used with keywords PIVOT and FACTOR.

DOUBLE

If present and nonzero, double precision is used.

FACTOR

Named variable in which the LU factorization of A , computed by the [IMSL_LUFAC](#) procedure, is stored. The strictly lower-triangular part of this array contains information necessary to construct L , and the upper-triangular part contains U . The PIVOT and FACTOR keywords must be used together. The FACTOR and CONDITION keywords cannot be used together.

INVERSE

Named variable into which the inverse of the matrix A is stored.

PIVOT

Named variable into which the pivot sequence for the factorization, computed by the [IMSL_LUFAC](#) procedure, is stored. The PIVOT and FACTOR keywords must be used together. The PIVOT and CONDITION keywords cannot be used together.

TRANSPOSE

If present and nonzero, $A^H x = b$ is solved.

Discussion

The [IMSL_LUSOL](#) function solves a system of linear algebraic equations with a real or complex coefficient matrix A . Any of several related computations can be performed by using keywords. These extra tasks include solving $A^H x = b$ or computing the solution of $Ax = b$ given the LU factorization of A . The function first computes the LU factorization of A with partial pivoting such that $L^{-1}PA = U$.

The matrix U is upper-triangular, while $L^{-1}A \equiv P_{n-1} L_{n-2} P_{n-2} \dots L_0 P_0 A \equiv U$. The factors P_i and L_i are defined by the partial pivoting. Each P_i is an interchange of row i with row $j \geq i$. Thus, P_i is defined by that value of j . Every $L_i = m_i e_i^T$ is an elementary elimination matrix. The vector m_i is zero in entries $0, \dots, i-1$. This vector is stored as column i in the strictly lower-triangular part of the working matrix containing the decomposition information.

The factorization efficiency is based on a technique of “loop unrolling and jamming” by Dr. Leonard J. Harding of the University of Michigan, Ann Arbor, Michigan. The solution of the linear system is then found by solving two simpler systems, $y = L^{-1}b$ and $x = U^{-1}y$. When the solution to the linear system or the inverse of the matrix is sought, an estimate of the L_1 condition number of A is computed using the same algorithm as in Dongarra et al. (1979). If the estimated condition number is greater than $1/\epsilon$ (where ϵ is the machine precision), a warning message is issued. This

indicates that very small changes in A may produce large changes in the solution x . The IMSL_LUSOL function fails if U , the upper-triangular part of the factorization, has a zero diagonal element.

Examples

Example 1: Solving a System

This example solves a system of three linear equations. This is the simplest use of the function. The equations are as follows:

$$x_0 + 3x_1 + 3x_2 = 1$$

$$x_0 + 3x_1 + 4x_2 = 4$$

$$x_0 + 4x_1 + 3x_2 = -1$$

```
RM, a, 3, 3
; Input a matrix containing the coefficients.
row 0: 1 3 3
row 1: 1 3 4
row 2: 1 4 3
RM, b, 3, 1
; Input a vector containing the right-hand side.
row 0: 1
row 1: 4
row 2: -1
x = IMSL_LUSOL(b, a)
; Call IMSL_LUSOL to compute the solution.
PM, x, Title = 'Solution'
; Print solution and residual.
Solution
-2.00000
-2.00000
3.00000
PM, a # x - b, Title = 'Residual'
Residual
0.00000
0.00000
0.00000
```

Example 2: Transpose Problem

This example solves the transpose problem $A^H x = b$.

```
RM, a, 3, 3
; Input the matrix containing the coefficients.
row 0: 1 3 3
row 1: 1 3 4
row 2: 1 4 3
RM, b, 3, 1
; Input the vector containing the right-hand side.
row 0: 1
row 1: 4
row 2: -1
x = IMSL_LUSOL(b, a, /Transpose)
; Call IMSL_LUSOL with keyword Transpose set.
PM, x, Title = 'Solution'
; Print the solution and the residual.
Solution
    4.00000
   -4.00000
    1.00000
PM, TRANSPOSE(a) # x - b, Title = 'Residual'
Residual
    0.00000
    0.00000
    0.00000
```

Example 3: Solving with Multiple Right-hand Sides

This example computes the solution of two systems. Only the right-hand sides differ. The matrix and first right-hand side are given in the initial example. The second right-hand side is the vector $c = [0.5, 0.3, 0.4]^T$. The factorization information is computed by the [IMSL_LUFAC](#) procedure and is used to compute the solutions in calls to [IMSL_LUSOL](#).

```
RM, a, 3, 3
; Input the coefficient matrix.
row 0: 1 3 3
row 1: 1 3 4
row 2: 1 4 3
RM, b, 3, 1
; Input the first right-hand side.
row 0: 1
row 1: 4
row 2: -1
RM, c, 3, 1
; Input the second right-hand side.
row 0: .5
```

```

row 1: .3
row 2: .4
IMSL_LUFAC, a, pvt, fac
; Call IMSL_LUFAC to factor the coefficient matrix.
x = IMSL_LUSOL(b, Factor = fac, Pivot = pvt)
; Call IMSL_LUSOL with factored form of the coefficient
; matrix and the first right-hand side.
PM, x, Title = 'Solution'
; Print the solution of  $Ax = b$ .
Solution
-2.00000
-2.00000
3.00000
PM, a # x - b, Title = 'Residual'
Residual
0.00000
0.00000
0.00000
y = IMSL_LUSOL(c, Factor = fac, Pivot = pvt)
; Call IMSL_LUSOL with factored form of the coefficient
; matrix and the second right-hand side.
PM, y, Title = 'Solution'
; Print the solution of  $Ax = b$ .
Solution
1.40000
-0.100000
-0.200000
PM, a # y - c, $
Title = 'Residual', Format = '(f8.5)'

Residual
0.00000
0.00000
0.00000

```

Errors

Warning Errors

MATH_ILL_CONDITIONED—Input matrix is too ill-conditioned. An estimate of the reciprocal of its L_1 condition number is #. The solution might not be accurate.

Fatal Errors

MATH_SINGULAR_MATRIX—Input matrix is singular.

Version History

6.4	Introduced

See Also

[IMSL_SP_LUFAC](#)

IMSL_LUFAC

The IMSL_LUFAC procedure computes the LU factorization of a real or complex matrix.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
IMSL_LUFAC, a[, pivot[, fac] [, CONDITION=variable] [, /DOUBLE]
    [, INVERSE=variable] [, L=variable] [, PA=variable] [, TRANSPOSE=value]
    [, U=variable]]
```

Arguments

a

Two-dimensional matrix containing the coefficient matrix. Element $A(i, j)$ contains the j -th coefficient of the i -th equation.

fac

A named variable that will contain a two-dimensional matrix containing the LU factorization of A . The strictly lower-triangular part of this matrix contains information necessary to construct L , and the upper-triangular part contains U .

pivot

A named variable that will contain a one-dimensional matrix containing the pivot sequence of the factorization.

Keywords

CONDITION

Named variable into which an estimate of the L_1 condition number is stored.

DOUBLE

If present and nonzero, double precision is used.

INVERSE

Named variable into which the inverse of the matrix A is stored.

L

Named variable into which the strictly lower-triangular matrix L of the LU factorization is stored.

PA

Named variable into which the matrix resulting from applying the pivot permutation to A is stored.

TRANSPOSE

If present and nonzero, $A^T X = b$ is solved.

U

Named variable into which the upper-triangular matrix U of the LU factorization is stored.

Discussion

Any of several related computations can be performed by using keywords. These extra tasks include computing the LU factorization of A^T , computing an estimate of the L_1 condition number, and returning L or U separately.

The IMSL_LUFAC procedure computes the LU factorization of A with partial pivoting such that $L^{-1}PA = U$. The matrix U is upper-triangular, while $L^{-1}A \equiv P_{n-1}L_{n-2}P_{n-2}...L_0P_0A \equiv U$. The factors P_i and L_i are defined by the partial pivoting. Each P_i is an interchange of row i with row $i \geq j$. Thus, P_i is defined by that value of j . Every $L_i = m_i e_i^T$ is an elementary elimination matrix. The vector m_i is zero in entries $0, \dots, i-1$. This vector is stored as column i in the strictly lower-triangular part of the working array containing the decomposition information.

The factorization efficiency is based on a technique of “loop unrolling and jamming” due to Dr. Leonard J. Harding of the University of Michigan, Ann Arbor, Michigan. When the inverse of the matrix is sought, an estimate of the L_1 condition number of A is computed using the same algorithm as in Dongarra et al. (1979). If the estimated

condition number is greater than $1/\epsilon$ (where ϵ is the machine precision), a warning message is issued. This indicates that very small changes in A may produce large changes in the solution x . The IMSL_LUFAC procedure fails if U , the upper-triangular part of the factorization, has a zero diagonal element.

Examples

Example 1

This example computes the LU factorization of a matrix and prints it out in the default form with the information needed to construct L and U combined in one array. The matrix is as follows:

$$\begin{bmatrix} 1 & 3 & 3 \\ 1 & 3 & 4 \\ 1 & 4 & 3 \end{bmatrix}$$

```
RM, a, 3, 3
; Input the matrix to be factored.
row 0: 1 3 3
row 1: 1 3 4
row 2: 1 4 3
IMSL_LUFAC, a, pvt, fac
; Factor the matrix by calling IMSL_LUFAC.
PM, fac, Title = 'LU factors of A'
; Print the results.
LU factors of A
  1.00000      3.00000      3.00000
 -1.00000      1.00000      0.00000
 -1.00000     -0.00000      1.00000
PM, pvt, Title = 'Pivot sequence'
Pivot sequence
  1
  3
  3
```

Example 2

This example computes the factorization, uses keywords to return the factorization in separate named variables, and returns the original matrix after the pivot permutation is applied.

```
RM, a, 3, 3
; Input the matrix to be factored.
row 0: 1 3 3
      row 1: 1 3 4
```

```
row 2: 1 4 3
IMSL_LUFAC, a, L = l, U = u, PA = pa
; Call IMSL_LUFAC with the keywords L and U.
PM, l, Title = 'L'
; Print the results.
L
1.00000      0.00000      0.00000
1.00000      1.00000      0.00000
1.00000      0.00000      1.00000
PM, u, Title = 'U'

U
1.00000      3.00000      3.00000
0.00000      1.00000      0.00000
0.00000      0.00000      1.00000
PM, l # u - pa, $
Title = 'Residual: L # U - PA'
Residual: L # U - PA
0.00000      0.00000      0.00000
0.00000      0.00000      0.00000
0.00000      0.00000      0.00000
```

Errors

Warning Errors

MATH_ILL_CONDITIONED—Input matrix is too ill-conditioned. An estimate of the reciprocal of its L_1 condition number is #. The solution might not be accurate.

Fatal Errors

MATH_SINGULAR_MATRIX—Input matrix is singular.

Version History

6.4	Introduced

IMSL_CHSOL

The IMSL_CHSOL function solves a symmetric positive definite system of real or complex linear equations $Ax = b$.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_CHSOL(b[, a] [, CONDITION=variable] [, /DOUBLE]  
[, FACTOR=variable] [, INVERSE=variable])
```

Return Value

The solution of the linear system $Ax = b$.

Arguments

b

One-dimensional matrix containing the right-hand side.

a

Two-dimensional matrix containing the coefficient matrix. Matrix $A(i, j)$ contains the j -th coefficient of the i -th equation.

Keywords

CONDITION

Named variable into which an estimate of the L_1 condition number is stored. The CONDITION and FACTOR keywords cannot be used together.

DOUBLE

If present and nonzero, double precision is used.

FACTOR

Named variable in which the LL^H factorization of A is stored. The lower-triangular part of this matrix contains L , and the upper-triangular part contains L^H . The `CONDITION` and `FACTOR` keywords cannot be used together.

INVERSE

Specifies a named variable into which the inverse of the matrix A is stored. This keyword is not allowed if A is complex.

Discussion

The `IMSL_CHSOL` function solves a system of linear algebraic equations having a symmetric positive definite coefficient matrix A . The function first computes the Cholesky factorization LL^H of A . The solution of the linear system is then found by solving the two simpler systems, $y = L^{-1}b$ and $x = L^{-H}y$. An estimate of the L_1 condition number of A is computed using the same algorithm as in Dongarra et al. (1979). If the estimated condition number is greater than $1/\epsilon$ (where ϵ is the machine precision), a warning message is issued. This indicates that very small changes in A may produce large changes in the solution x .

The `IMSL_CHSOL` function fails if L , the lower-triangular matrix in the factorization, has a zero diagonal element.

Examples

Example 1

```
RM, a, 3, 3
; Define the coefficient matrix.
row 0:  1  -3  2
row 1: -3  10 -5
row 2:  2  -5  6
RM, b, 3, 1
; Define the right-hand side.
row 0:  27
row 1: -78
row 2:  64
x = IMSL_CHSOL(b, a)
; Call IMSL_CHSOL to compute the solution.
PM, x, Title = 'Solution'
Solution
    1.00000
   -4.00000
```

```

7.00000
PM, a # x - b, Title = 'Residual'
Residual
0.00000
0.00000
0.00000

```

Example 2

This example solves a system of five linear equations with Hermitian positive definite coefficient matrix. The equations are as follows:

$$2x_0 + (-1 + i)x_1 = 1 + 5i$$

$$(-1 - i)x_0 + 4x_1 + (1 + 2i)x_2 = 12 - 6i$$

$$(-1 - 2i)x_1 + 10x_2 + 4ix_3 = 1 + (-16i)$$

$$(-4ix_2) + 6x_3 + (i + 1)x_4 = -3 - 3i$$

$$(1 - i)x_3 + 9x_4 = 25 + 16i$$

```

RM, a, 5, 5, /Complex
; Input the complex matrix A.
row 0: 2      (-1,1) 0      0      0
row 1: (-1,-1) 4      (1,2) 0      0
row 2: 0      (1,-2) 10     (0,4) 0
row 3: 0      0      (0,-4) 6      (1,1)
row 4: 0      0      0      (1,-1) 9
RM, b, 5, 1, /Complex
; Input the right-hand side.
row 0: (1, 5)
row 1: (12, -6)
row 2: (1, -16)
row 3: (-3, -3)
row 4: (25, 16)
x = IMSL_CHSOL(b, a)
; Compute the solution.
PM, x, Title = 'Solution', Format = '("(" ,f8.5,"",",f8.5,"")"'
; Output the results.
Solution
( 2.00000, 1.00000)
( 3.00000,-0.00000)
(-1.00000,-1.00000)
( 0.00000,-2.00000)
( 3.00000, 2.00000)
PM, a # x-b, Title = 'Residual', Format='("(" ,f8.5,"",",f8.5,"")"'
Residual
( 0.00000, 0.00000)
( 0.00000,-0.00000)

```

```
( 0.00000, 0.00000)
( 0.00000, 0.00000)
( 0.00000, 0.00000)
```

Errors

Warning Errors

MATH_ILL_CONDITIONED—Input matrix is too ill-conditioned. An estimate of the reciprocal of its L_1 condition number is #. The solution might not be accurate.

Fatal Errors

MATH_NONPOSITIVE_MATRIX—Leading # by # submatrix of the input matrix is not positive definite.

MATH_SINGULAR_MATRIX—Input matrix is singular.

MATH_SINGULAR_TRI_MATRIX—Input triangular matrix is singular. The index of the first zero diagonal element is #.

Version History

6.4	Introduced

IMSL_CHFAC

The IMSL_CHFAC procedure computes the Cholesky factor, L , of a real or complex symmetric positive definite matrix A , such that $A = LL^H$.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
IMSL_CHFAC, a, fac [, CONDITION=variable] [, /DOUBLE]
[, INVERSE=variable]
```

Arguments

a

Two-dimensional matrix containing the coefficient matrix. Element $A(i, j)$ contains the j -th coefficient of the i -th equation.

fac

A named variable that will contain a two-dimensional matrix containing the Cholesky factorization of A . Note that fac contains L in the lower triangle and L^H in the upper triangle.

Keywords

CONDITION

Named variable into which an estimate of the L_1 condition number is stored.

DOUBLE

If present and nonzero, double precision is used.

INVERSE

Named variable into which the inverse of the matrix A is stored. This keyword is not allowed if A is complex.

Discussion

The IMSL_CHFAC procedure computes the Cholesky factorization LL^H of a symmetric positive definite matrix A . When the inverse of the matrix is sought, an estimate of the L_1 condition number of A is computed using the same algorithm as in Dongarra et al. (1979). If the estimated condition number is greater than $1/\epsilon$ (where ϵ is the machine precision), a warning message is issued. This indicates that very small changes in A may produce large changes in the solution x .

The IMSL_CHFAC function fails if L , the lower-triangular matrix in the factorization, has a zero diagonal element.

Example

This example computes the Cholesky factorization of a 3 x 3 matrix.

```
RM, a, 3, 3
; Define the matrix A.
row 0:  1  -3  2
row 1: -3  10 -5
row 2:  2  -5  6
IMSL_CHFAC, a, fac
; Call IMSL_CHFAC to compute the factorization.
PM, fac, Title = 'Cholesky factor'
Cholesky factor
  1.00000      -3.00000      2.00000
 -3.00000      1.00000      1.00000
  2.00000      1.00000      1.00000
```

Errors

Warning Errors

MATH_ILL_CONDITIONED—Input matrix is too ill-conditioned. An estimate of the reciprocal of its L_1 condition number is #. The solution might not be accurate.

Fatal Errors

MATH_NONPOSITIVE_MATRIX—Leading # by # submatrix of the input matrix is not positive definite.

MATH_SINGULAR_MATRIX—Input matrix is singular.

MATH_SINGULAR_TRI_MATRIX—Input triangular matrix is singular. The index of the first zero diagonal element is #.

Version History

6.4	Introduced

IMSL_QRSOL

The IMSL_QRSOL function solves a real linear least-squares problem $Ax = b$.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_QRSOL(b[, a] [, AUXQR=variable] [, BASIS=variable]
[, /DOUBLE] [, QR=variable] [, PIVOT=variable] [, RESIDUAL=variable]
[, TOLERANCE=value])
```

Return Value

The solution, x , of the linear least-squares problem $Ax = b$.

Arguments

b

Matrix containing the right-hand side.

a

(Optional) Two-dimensional matrix containing the coefficient matrix. Element $A(i, j)$ contains the j -th coefficient of the i -th equation.

Keywords

AUXQR

Named variable in which the matrix containing the scalars τ_k of the Householder transformations that define the decomposition, as computed in the IMSL_QRFAC procedure, is stored. The AUXQR, PIVOT, and QR keywords must be used together.

BASIS

Named variable containing an integer specifying the number of columns used in the solution. The value *BASIS* = *k*, if $|r_{k,k}| < \text{TOLERANCE} * |r_{0,0}|$ and $|r_{i,i}| \geq \text{TOLERANCE} * |r_{0,0}|$ for $i = 0, 1, \dots, k - 1$. For more information on the use of this option, see “[Discussion](#)” on page 98.

DOUBLE

If present and nonzero, double precision is used.

QR

Named variable which stores the matrix containing Householder transformations that define the decomposition, as computed in the [IMSL_QRFAC](#) procedure. The AUXQR, PIVOT, and QR keywords must be used together.

PIVOT

Named variable in which the array containing the desired variable order and usage information is stored. The AUXQR, PIVOT, and QR keywords must be used together.

- On input, if $\text{PIVOT}(k) > 0$, then column *k* of *A* is an initial column. If $\text{PIVOT}(k) = 0$, then the column of *A* is a free column and can be interchanged in the column pivoting. If $\text{PIVOT}(k) < 0$, then column *k* of *A* is a final column. If all columns are specified as initial (or final) columns, then no pivoting is performed. (The permutation matrix *P* is the identity matrix in this case.)
- On output, $\text{PIVOT}(k)$ contains the index of the column of the original matrix that has been interchanged into column *k*.
- Default: $\text{PIVOT}(*) = 0$

Note

If IMSL_QRSOL is used to solve a problem previously factored in [IMSL_QRFAC](#), the matrix specified by PIVOT should contain the same information that the IMSL_QRFAC parameter PIVOT contained upon output.

RESIDUAL

Named variable in which the matrix containing the residual vector $b - Ax$ is stored.

TOLERANCE

Nonnegative tolerance used to determine the subset of columns of A to be included in the solution. Default: $\text{TOLERANCE} = \text{SQRT}(\epsilon)$, where ϵ is machine precision

Discussion

IMSL_QRSOL solves a system of linear least-squares problems $Ax = b$ with column pivoting. It computes a QR factorization of the matrix AP , where P is the permutation matrix defined by the pivoting, and computes the smallest integer k satisfying $|r_{k,k}| < \text{TOLERANCE} * |r_{0,0}|$ to the output keyword BASIS.

Householder transformations:

$$Q_k = I - \tau_k u_k u_k^T, k = 0, \dots, \min(m-1, n) - 1$$

compute the factorization. The decomposition is computed in the form $Q_{\min(m-1, n)-1} \dots Q_0 AP = R$, so $AP = QR$ where $Q = Q_0 \dots Q_{\min(m-1, n)-1}$. Since each Householder vector u_k has zeros in the first $k+1$ entries, it is stored as part of column k of QR . The upper-trapezoidal matrix R is stored in the upper-trapezoidal part of the first $\min(m, n)$ rows of QR . The solution x to the least-squares problem is computed by solving the upper-triangular system of linear equations $R(0:k, 0:k) y(0:k) = (Q^T b)(0:k)$ with $k = \text{Basis} - 1$. The solution is completed by setting $y(k:n-1)$ to zero and rearranging the variables, $x = Py$.

If the QR and $AUXQR$ keywords are specified, then the function computes the least-squares solution to $Ax = b$ given the QR factorization previously defined. There are Basis columns used in the solution. Hence, in the case that all columns are free, x is computed as described in the default case.

Example

This example illustrates the least-squares solution of four linear equations in three unknowns by using column pivoting. This is equivalent to least-squares quadratic polynomial fitting to four data values. The polynomial is written as $p(t) = x_0 + tx_1 + t^2x_2$ and the data pairs (t_i, b_i) , $t_i = 2(i+1)$, $i = 0, 1, 2, 3$. The solution to $Ax = b$ is returned by the `IMSL_QRSOL` function.

```
RM, a, 4, 3
; Define the coefficient matrix.
  row 0:  1 2 4
  row 1:  1 4 16
  row 2:  1 6 36
  row 3:  1 8 64
RM, b, 4, 1
```

```

; Define the right-hand side.
  row 0:  4.999
  row 1:  9.001
  row 2: 12.999
  row 3: 17.001
x = IMSL_QRSOL(b, a)
; Call IMSL_QRSOL.
PM, x, Title = 'Solution', Format = '(f8.5)'
; Output the results.
Solution
  0.99900
  2.00020
  0.00000
PM, a # x - b, Title = 'Residual', Format = '(f10.7)'
Residual
  0.0004015
 -0.0011997
  0.0012007
 -0.0004005

```

Errors

Fatal Errors

MATH_SINGULAR_TRI_MATRIX—Input triangular matrix is singular. The index of the first zero diagonal term is #.

Version History

6.4	Introduced

See Also

[IMSL_SP_LUFAC](#)

IMSL_QRFAC

The IMSL_QRFAC procedure computes the QR factorization of a real matrix A .

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
IMSL_QRFAC, a [, pivot [, auxqr, qr] [, AP=variable] [, BASIS=variable]
[, /DOUBLE] [, Q=variable] [, R=variable] [, TOLERANCE=value]]
```

Arguments

a

A two-dimensional matrix containing the coefficient matrix. Element $A(i,j)$ contains the j -th coefficient of the i -th equation.

pivot

A one-dimensional matrix containing the desired variable order and usage information.

- On input, if $\text{pivot}(k) > 0$, then column k of A is an initial column. If $\text{pivot}(k) = 0$, then the column of A is a free column and can be interchanged in the column pivoting. If $\text{pivot}(k) < 0$, then column k of A is a final column. If all columns are specified as initial (or final) columns, then no pivoting is performed. (The permutation matrix P is the identity matrix in this case.) Default: $\text{pivot} (*) = 0$
- On output, $\text{pivot}(k)$ contains the index of the column of the original matrix that has been interchanged into column k .

auxqr

Matrix containing the scalars τ_k of the Householder transformations that define the decomposition.

qr

Matrix containing the Householder transformations that define the decomposition.

Keywords**AP**

Named variable into which the product AP of the identity $AP = QR$ is stored. This keyword is useful when attempting to compute the residual $AP - QR$.

BASIS

Named variable into which an integer containing the number of columns used in the solution is stored. The value $BASIS = k$, if $|r_{k,k}| < TOLERANCE * |r_{0,0}|$ and $|r_{i,i}| \geq TOLERANCE * |r_{0,0}|$ for $i = 0, 1, \dots, k - 1$. For more information, see [“Discussion”](#) on page 101.

DOUBLE

If present and nonzero, double precision is used.

Q

Named variable in which the two-dimensional matrix containing the orthogonal matrix of the $AP = QR$ factorization is stored.

R

Named variable in which the two-dimensional matrix containing the upper-triangular matrix of the $AP = QR$ decomposition is stored.

TOLERANCE

Nonnegative tolerance used to determine the subset of columns of A to be included in the solution. Default: $TOLERANCE = \text{SQRT}(\epsilon)$, where ϵ is machine precision

Discussion

The `IMSL_QRFAC` procedure computes a QR factorization of the matrix AP , where P is the permutation matrix defined by the pivoting and computes the smallest integer k satisfying $|r_{k,k}| < TOLERANCE * |r_{0,0}|$ to the keyword `BASIS`.

Householder transformations:

$$Q_k = I - \tau_k u_k u_k^T, k = 0, \dots, \min(m-1, n) - 1$$

compute the factorization. The decomposition is computed in the form $Q_{\min(m-1, n)-1} \dots Q_0 A P = R$, so $AP = QR$ where $Q = Q_0 \dots Q_{\min(m-1, n)-1}$. Since each Householder vector u_k has zeros in the first $k+1$ entries, it is stored as part of column k of QR . The upper-trapezoidal matrix R is stored in the upper-trapezoidal part of the first $\min(m, n)$ rows of QR .

When computing the factorization, the procedure computes the QR factorization of AP with P defined by the input *pivot* and by column pivoting among “free” columns. Before the factorization, initial columns are moved to the beginning of the array A and the final columns to the end. Neither initial nor final columns are permuted further during the computation. Only the free columns are moved.

Example

Using the same data as the first example given for the [IMSL_QRSOL](#) function, this sample computes the QR factorization of the coefficient. Using keywords, the factorization is returned in the full matrices, rather than the default condensed format.

```
RM, a, 4, 3
; Define the coefficient matrix.
row 0:  1 2 4
row 1:  1 4 16
row 2:  1 6 36
row 3:  1 8 64
IMSL_QRFAC, a, pvt, Q = q, R = r, AP = ap
; Call IMSL_QRFAC using keywords Q, R, and AP.
PM, q, Title = 'Q', Format = '(4f12.6)'
; Output the results.
Q
    -0.053149    -0.542171     0.808224    -0.223607
    -0.212598    -0.657436    -0.269408     0.670820
    -0.478345    -0.345794    -0.449013    -0.670820
    -0.850390     0.392754     0.269408     0.223607

PM, r, Title = 'R', Format = '(3f12.6)'
R
    -75.259552   -10.629880    -1.594482
     0.000000    -2.646819    -1.152647
     0.000000     0.000000     0.359211
     0.000000     0.000000     0.000000

PM, pvt, Title = 'Pvt'
Pvt
    3
    2
```

1

```
PM, q # r - ap, Title = 'Residual', Format = '(3f12.6)'  
Residual  
-0.000004 -0.000001 -0.000000  
0.000000 -0.000000 0.000000  
0.000000 -0.000000 -0.000000  
0.000000 -0.000000 -0.000000
```

Errors

Fatal Errors

`MATH_SINGULAR_TRI_MATRIX`—Input triangular matrix is singular. The index of the first zero diagonal term is #.

Version History

6.4	Introduced

IMSL_SVDCOMP

The IMSL_SVDCOMP function computes the singular value decomposition (SVD), $A = USV^T$, of a real or complex rectangular matrix A . An estimate of the rank of A also can be computed.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_SVDCOMP(a [, /DOUBLE] [, INVERSE=variable]
    [, RANK=variable] [, TOL_RANK=variable] [, U=variable] [, V=variable])
```

Return Value

One-dimensional array containing ordered singular values of A .

Arguments

a

Two-dimensional matrix containing the coefficient matrix. Element $A(i, j)$ contains the j -th coefficient of the i -th equation.

Keywords

DOUBLE

If present and nonzero, double precision is used.

INVERSE

Named variable into which the generalized inverse of the matrix A is stored.

RANK

Named variable into which an estimate of the rank of A is stored.

TOL_RANK

Named variable containing the tolerance used to determine when a singular value is negligible and replaced by the value zero. If `TOL_RANK > 0`, then a singular value $s_{i,i}$ is considered negligible if $s_{i,i} \leq \text{TOL\_RANK}$. If `TOL_RANK < 0`, then a singular value $s_{i,i}$ is considered negligible if $s_{i,i} \leq \text{TOL\_RANK} * \|A\|_{\text{infinity}}$.

In this case, $|\text{TOL\_RANK}|$ should be an estimate of relative error or uncertainty in the data.

U

Named variable into which the left-singular vectors of A are stored.

V

Named variable into which the right-singular vectors of A are stored.

Discussion

The `IMSL_SVDCOMP` function computes the singular value decomposition of a real or complex matrix A . It reduces the matrix A to a bidiagonal matrix B by pre- and post-multiplying Householder transformations, then, it computes singular value decomposition of B using the implicit-shifted QR algorithm. An estimate of the rank of the matrix A is obtained by finding the smallest integer k such that $s_{k,k} \leq \text{TOL\_RANK}$ or $s_{k,k} \leq \text{TOL\_RANK} * \|A\|_{\text{infinity}}$.

Since $s_{i+1,i+1} \leq s_{i,i}$, it follows that all the $s_{i,i}$ satisfy the same inequality for $i = k, \dots, \min(m, n) - 2$. The rank is set to the value k . If $A = USV^T$, its generalized inverse is $A^+ = VS^+U^T$. Here, $S^+ = \text{diag}(s_{0,0}^{-1}, \dots, s_{i,i}^{-1}, 0, \dots, 0)$. Only singular values that are not negligible are reciprocated. If the keyword `INVERSE` is specified, the function first computes the singular value decomposition of the matrix A , then computes the generalized inverse. The `IMSL_SVDCOMP` function fails if the QR algorithm does not converge after 30 iterations.

Examples

Example 1

This example computes the singular values of a 6-by-4 real matrix.

```
RM, a, 6, 4
; Define the matrix.
row 0: 1 2 1 4
row 1: 3 2 1 3
```

```

row 2: 4 3 1 4
row 3: 2 1 3 1
row 4: 1 5 2 2
row 5: 1 2 2 3
; Call IMSL_SVDCOMP and output the results.
singvals = IMSL_SVDCOMP(a)
PM, singvals
    11.4850
    3.26975
    2.65336
    2.08873

```

Example 2

This example computes the singular value decomposition of the 6-by-4 real matrix *A*. Matrices *U* and *V* are returned using keywords *U* and *V*.

```

RM, a, 6, 4
; Define the matrix.
row 0: 1 2 1 4
row 1: 3 2 1 3
row 2: 4 3 1 4
row 3: 2 1 3 1
row 4: 1 5 2 2
row 5: 1 2 2 3
; Call IMSL_SVDCOMP with keywords U and V and output the results.
singvals = IMSL_SVDCOMP(a, U = u, V = v)
PM, singvals, Title = 'Singular values', Format = '(f12.6)'
Singular values
    11.485018
    3.269752
    2.653356
    2.088730
PM, u, Title = 'Left singular vectors, U', Format = '(4f12.6)'
Left singular vectors, U
    -0.380476    0.119671    0.439083   -0.565399
    -0.403754    0.345111   -0.056576    0.214776
    -0.545120    0.429265    0.051392    0.432144
    -0.264784   -0.068320   -0.883861   -0.215254
    -0.446310   -0.816828    0.141900    0.321270
    -0.354629   -0.102147   -0.004318   -0.545800
PM, v, Title = 'Right singular vectors, V', Format = '(4f12.6)'
Right singular vectors, V
    -0.444294    0.555531   -0.435379    0.551754
    -0.558067   -0.654299    0.277457    0.428336
    -0.324386   -0.351361   -0.732099   -0.485129
    -0.621239    0.373931    0.444402   -0.526066

```

Errors

Warning Errors

`MATH_SLOWCONVERGENT_MATRIX`—Convergence cannot be reached after 30 iterations.

Version History

6.4	Introduced

IMSL_CHNND SOL

The IMSL_CHNND SOL function solves a real symmetric nonnegative definite system of linear equations $Ax = b$. Computes the solution to $Ax = b$ given the Cholesky factor.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_CHNND SOL(b[, a] [, /DOUBLE] [, FACTOR=value]
                        [, INVERSE=variable] [, TOLERANCE=value])
```

Return Value

A solution x of the linear system $Ax = b$.

Arguments

b

Matrix containing the right-hand side.

a

(Optional) Two-dimensional matrix containing the coefficient matrix. Element $A(i, j)$ contains the j -th coefficient of the i -th equation.

Keywords

DOUBLE

If present and nonzero, double precision is used.

FACTOR

The LL^T factorization of A . The lower-triangular part of this matrix contains L , and the upper-triangular part contains L^T .

INVERSE

Named variable into which the inverse of the matrix A is stored.

TOLERANCE

Tolerance used in determining linear dependence. Default: $\text{TOLERANCE} = 100\epsilon$, where ϵ is machine precision

Discussion

The `IMSL_CHNND SOL` function solves a system of linear algebraic equations having a symmetric nonnegative definite (positive semidefinite) coefficient matrix. It first computes a Cholesky (LL^H or $R^H R$) factorization of the coefficient matrix A .

The factorization algorithm is based on the work of Healy (1968) and proceeds sequentially by columns. The i -th column is declared to be linearly dependent on the first $i - 1$ columns if:

$$\left| a_{ii} - \sum_{j=0}^{i-1} r_{ji}^2 \right| \leq \epsilon |a_{ii}|$$

where ϵ (specified by `TOLERANCE`) may be set. When a linear dependence is declared, all elements in the i -th row of R (column of L) are set to zero.

Modifications due to Farebrother and Berry (1974) and Barrett and Healy (1978) for checking for matrices that are not nonnegative definite also are incorporated. The `IMSL_CHNND SOL` function declares A to be not nonnegative definite and issues an error message if either of the following conditions is satisfied:

1.

$$a_{ii} = \sum_{j=0}^{i-1} r_{ji}^2 < r_{ii}^2$$

2.

$$r = 0 \text{ and } \left| a_{ik} - \sum_{j=0}^{i-1} r_{ji} r_{jk} \right| > \epsilon \sqrt{a_{ii} a_{kk}}, k > i$$

Healy's (1968) algorithm and the IMSL_CHNND SOL function permit the matrices A and R to occupy the same storage. Barrett and Healy (1978), in their remark, neglect this fact. The IMSL_CHNND SOL function uses:

$$\sum_{j=0}^{i-1} r_{ij}^2 \text{ for } a_{ii}$$

in condition 2 above to remedy this problem.

If an inverse of the matrix A is required and the matrix is not (numerically) positive definite, then the resulting inverse is a symmetric g_2 inverse of A . For a matrix G to be a g_2 inverse of a matrix A , G must satisfy conditions 1 and 2 for the Moore-Penrose inverse but generally fail conditions 3 and 4. The four conditions for G to be a Moore-Penrose inverse of A are as follows:

1. $AGA = A$
2. $GAG = G$
3. AG is symmetric
4. GA is symmetric

The solution of the linear system $Ax = b$ is computed by solving the factored version of the linear system $R^T Rx = b$ as two successive triangular linear systems. In solving the triangular linear systems, if the elements of a row of R are all zero, the corresponding element of the solution vector is set to zero. For a detailed description of the algorithm, see Section 2 in Sallas and Lioni (1988). This routine is useful to solve normal equations in a linear least-squares problem.

Example

A solution to a system of four linear equations is obtained. Maindonald (1984, pp. 83–86, 104–105) discusses the computations for the factorization and solution to this problem.

```
RM, a, 4, 4
; Define the coefficient matrix.
row 0: 36 12 30 6
row 1: 12 20 2 10
row 2: 30 2 29 1
row 3: 6 10 1 14
RM, b, 4, 1
; Define the right-hand side.
row 0: 18
row 1: 22
```

```

row 2:  7
row 3: 20
x = IMSL_CHNDSOL(b, a)
; Define the right-hand side.
PM, x
; Output the results.
0.166667
0.500000
0.000000
1.000000

```

Errors

Warning Errors

MATH_INCONSISTENT_EQUATIONS_2—Linear system of equations is inconsistent.

MATH_NOT_NONNEG_DEFINITE—Matrix A is not nonnegative definite.

Version History

6.4	Introduced

IMSL_CHNNDFAC

The IMSL_CHNNDFAC procedure computes the Cholesky factorization of the real matrix A such that $A = R^T R = LL^T$.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
IMSL_CHNNDFAC, a, fac [, /DOUBLE] [, INVERSE=variable]
[, TOLERANCE=value]
```

Arguments

a

Two-dimensional matrix containing the coefficient matrix. Element $A(i, j)$ contains the j -th coefficient of the i -th equation.

fac

Matrix containing the LL^T factorization of A .

Keywords

DOUBLE

If present and nonzero, double precision is used.

INVERSE

Named variable into which the inverse of the matrix A is stored.

TOLERANCE

Used in determining linear dependence. Default: $\text{TOLERANCE} = 100 \epsilon$, where ϵ is machine precision

Discussion

The factorization algorithm is based on the work of Healy (1968) and proceeds sequentially by columns. The i -th column is declared to be linearly dependent on the first $i - 1$ columns if:

$$\left| a_{ii} - \sum_{j=0}^{i-1} r_{ji}^2 \right| \leq \epsilon |a_{ii}|$$

where ϵ (specified in TOLERANCE) may be set. When a linear dependence is declared, all elements in the i -th row of R (column of L) are set to zero.

Modifications due to Farebrother and Berry (1974) and Barrett and Healy (1978) for checking for matrices that are not nonnegative definite also are incorporated. The IMSL_CHNNDFAC procedure declares A to not be nonnegative definite and issues an error message if either of the following conditions is satisfied:

1.

$$a_{ii} - \sum_{j=0}^{i-1} r_{ji}^2 < -\epsilon |a_{ii}|$$

2.

$$r = 0 \text{ and } \left| a_{ik} - \sum_{j=0}^{i-1} r_{ji} r_{jk} \right| > \epsilon \sqrt{a_{ii} a_{kk}}, k > i$$

Healy's (1968) algorithm and the IMSL_CHNNDFAC procedure permit the matrices A and R to occupy the same storage. Barrett and Healy (1978) in their remark neglect this fact. The IMSL_CHNNDFAC procedure uses:

$$\sum_{j=0}^{i-1} r_{ij}^2$$

for

$$a_{ii}$$

in condition 2 above to remedy this problem.

If an inverse of the matrix A is required and the matrix is not (numerically) positive definite, then the resulting inverse is a symmetric g_2 inverse of A . For a matrix G to be a g_2 inverse of a matrix A , G must satisfy conditions 1 and 2 for the Moore-Penrose inverse, but generally fail conditions 3 and 4. The four conditions for G to be a Moore-Penrose inverse of A are as follows:

1. $AGA = A$
2. $GAG = G$
3. AG is symmetric
4. GA is symmetric

Example

The symmetric nonnegative definite matrix in the initial example of [IMSL_CHNND SOL](#) is used to compute the factorization only in the first call to `IMSL_CHNND FAC`. Then, `IMSL_CHNND SOL` is called with both the LL^T factorization and the right-hand side vector as the input to compute a solution x .

```
RM, a, 4, 4
; Define the coefficient matrix.
row 0: 36 12 30 6
row 1: 12 20 2 10
row 2: 30 2 29 1
row 3: 6 10 1 14
IMSL_CHNND FAC, a, fac
PM, fac, Title = 'Factor', Format = '(4f12.3)'
Factor
    6.000      2.000      5.000      1.000
    2.000      4.000     -2.000      2.000
    5.000     -2.000      0.000      0.000
    1.000      2.000      0.000      3.000
RM, b, 4, 1
; Define the right-hand side.
row 0: 18
row 1: 22
row 2: 7
row 3: 20
; Compute the solution and output.
x = IMSL_CHNND SOL(b, Factor = fac)
```

```
PM, x, Title = 'Solution'
Solution
0.166667
0.500000
0.00000
1.00000
```

Errors

Warning Errors

MATH_INCONSISTENT_EQUATIONS_2—Linear system of equations is inconsistent.

MATH_NOT_NONNEG_DEFINITE—Matrix A is not nonnegative definite.

Version History

6.4	Introduced

IMSL_LINLSQ

The IMSL_LINLSQ function solves a linear least-squares problem with linear constraints.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_LINLSQ( b, a, c, bl, bu, contype [, ABS_TOLERANCE=value]
[, /DOUBLE] [, ITMAX=value] [, REL_TOLERANCE=value]
[, RESIDUAL=variable] [, XLB=array] [, XUB=array])
```

Return Value

One-dimensional array of length *nca* containing the approximate solution.

Arguments

a

Two-dimensional array of size *nra* by *nca* containing the coefficients of the least-squares equations, where *nra* is the number of least-squares equations and *nca* is the number of variables.

B

One-dimensional array of length *nra* containing the right-hand sides of the least-squares equations.

C

Two-dimensional array of size *ncon* by *nca* containing the coefficients of the constraints, where *ncon* is the number of constraints.

BL

One-dimensional array of length *ncon* containing the lower limit of the general constraints. If there is no lower limit on the *i*-th constraint, then *bl(i)* will not be referenced.

BU

One-dimensional array of length *ncon* containing the upper limit of the general constraints. If there is no upper limit on the *i*-th constraint, then *bu(i)* will not be referenced.

CONTYPE

One-dimensional array of length *ncon* indicating the type of constraints exclusive of simple bounds, where *CONTYPE(i)* = 0, 1, 2, 3 indicates =, ≤, ≥, and range constraints, respectively.

contype(i)	constraint
0	$\sum_{j=0}^{nca-1} c(i, j) = bl(i)$

2	$\sum_{j=0}^{nca-1} c(i, j) \leq bu(i)$
3	$bl(i) \leq \sum_{j=0}^{nca-1} c(i, j)$
4	$bl(i) \leq \sum_{j=0}^{nca-1} c(i, j) \leq bu(i)$

Keywords

ABS_TOLERANCE

Absolute rank determination tolerance to be used. Default:
ABS_TOLERANCE = SQRT(machine epsilon).

DOUBLE

If present and nonzero, double precision is used.

ITMAX

Set the maximum number of iterations. Default: ITMAX = 5*max(nra, nca)

REL_TOLERANCE

Relative rank determination tolerance to be used. Default:
REL_TOLERANCE = SQRT(machine epsilon).

RESIDUAL

Named variable into which an one-dimensional array containing the residuals $b - Ax$ of the least-squares equations at the approximate solution is stored.

XLB

One-dimensional array of length *nca* containing the lower bound on the variables. If there is no lower bound on the *i*-th variable, then *Xlb(i)* should be set to 1.0e30.

XUB

One-dimensional array of length *nca* containing the upper bound on the variables. If there is no upper bound on the *i*-th variable, then *XUB(i)* should be set to -1.0e30.

Discussion

The `IMSL_LINLSQ` function solves linear least-squares problems with linear constraints. These are systems of least-squares equations of the form

$$Ax \cong b$$

subject to

$$b_l \leq Cx \leq b_u$$

$$x_l \leq x \leq x_u$$

Here *A* is the coefficient matrix of the least-squares equations, *b* is the right-hand side, and *C* is the coefficient matrix of the constraints. The vectors *b_l*, *b_u*, *x_l* and *x_u* are the lower and upper bounds on the constraints and the variables, respectively. The system is solved by defining dependent variables $y \equiv Cx$ and then solving the least-squares system with the lower and upper bounds on *x* and *y*. The equation $Cx - y = 0$ is a set of equality constraints. These constraints are realized by heavy weighting, i.e., a penalty method, Hanson (1986, pp. 826-834).

Examples**Example 1**

This example solves the following problem in the least-squares sense:

$$3x_1 + 2x_2 + x_3 = 3.3$$

$$4x_1 + 2x_2 + x_3 = 2.2$$

$$2x_1 + 2x_2 + x_3 = 1.3$$

$$x_1 + x_2 + x_3 = 1.0$$

Subject to:

$$x_1 + x_2 + x_3 \leq 1$$

$$0 \leq x_1 \leq 0.5$$

$$0 \leq x_2 \leq 0.5$$

$$0 \leq x_3 \leq 0.5$$

```

a = TRANSPOSE([[3.0, 2.0, 1.0], [4.0, 2.0, 1.0], $
               [2.0, 2.0, 1.0], [1.0, 1.0, 1.0]])
b = [3.3, 2.3, 1.3, 1.0]
c = [[1.0], [1.0], [1.0]]
xub = [0.5, 0.5, 0.5]
xlb = [0.0, 0.0, 0.0]
contype = [1]
bc = [1.0]
; Note that only upper bound is set for contype=1.
sol = IMSL_LINLSQ(b, a, c, bc, bc, contype, Xlb = xlb, Xub = xub)
PM, sol, Title = 'Solution'
0.500000
0.300000
0.200000

```

Example 2

The same problem solved in the first example is solved again. This time residuals of the least-squares equations at the approximate solution are returned, and the norm of the residual vector is printed.

```

a = TRANSPOSE([[3.0, 2.0, 1.0], [4.0, 2.0, 1.0], $
               [2.0, 2.0, 1.0], [1.0, 1.0, 1.0]])
b = [3.3, 2.3, 1.3, 1.0]
c = [[1.0], [1.0], [1.0]]
xub = [0.5, 0.5, 0.5]
xlb = [0.0, 0.0, 0.0]
contype = [1]
bc = [1.0]
sol = IMSL_LINLSQ(b, a, c, bc, bc, contype, Xlb = xlb, $
                  Xub = xub, Residual = residual)
PM, sol, Title = 'Solution'
Solution
0.500000
0.300000
0.200000
PM, residual, Title = 'Residual'
Residual
-1.00000
0.500000
0.500000
0.00000
PRINT, 'Norm of Residual =', IMSL_NORM(residual)

```

Norm of Residual = 1.22474

Version History

6.4	Introduced

IMSL_SP_LUSOL

The IMSL_SP_LUSOL function solves a sparse system of linear equations $Ax = b$. By using keywords, any of several related computations can be performed.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_SP_LUSOL(b[, a] [, CONDITION=variable] [, /CSC_COL]
[, /CSC_ROW_IND] [, /CSC_VAL] [, FACTOR_COORD=value]
[, GWTH_FACTOR=variable] [, GWTH_LIM=value] [, /HYBRID_DENSITY]
[, /HYBRID_ORDER] [, ITER_REFINE=value] [, PIVOTING=value]
[, MEMORY_BLOCK=value] [, N_NONZERO=variable]
[, N_SEARCH_ROWS=value] [, SMALLEST_PVT=variable]
[, STABILITY=value] [, TOL_DROP=value] [, TRANSPOSE=value])
```

Return Value

A one-dimensional array containing the solution of the linear system $Ax = b$.

Arguments

b

One-dimensional matrix containing the right-hand side.

a

(Optional) Sparse matrix stored as an array of structures containing the coefficient matrix $A(i,j)$. See “[Sparse Matrices: Direct Methods](#)” on page 67 and its related sections for a description of structures used for sparse matrices.

Keywords

CONDITION

Named variable into which an estimate of the L_1 condition number is stored. The `FACTOR_COORD` and `CONDITION` keywords cannot be used together.

CSC_COL

Accept the coefficient matrix in compressed sparse column (CSC) format. See [“Sparse Coordinate Storage Format”](#) on page 68 for a discussion of this storage scheme. The keywords `CSC_COL`, `CSC_ROW_IND`, and `CSC_VAL` must be used together.

CSC_ROW_IND

Accept the coefficient matrix in compressed sparse column (CSC) format. See [“Sparse Coordinate Storage Format”](#) on page 68 for a discussion of this storage scheme. The keywords `CSC_COL`, `CSC_ROW_IND`, and `CSC_VAL` must be used together.

CSC_VAL

Accept the coefficient matrix in compressed sparse column (CSC) format. See [“Sparse Coordinate Storage Format”](#) on page 68 for a discussion of this storage scheme. The keywords `CSC_COL`, `CSC_ROW_IND`, and `CSC_VAL` must be used together.

FACTOR_COORD

The LU factorization of A as computed by `IMSL_SP_LUFAC`. If this keyword is used, then the argument A should not be used. This keyword is useful if solutions to systems involving the same coefficient matrix and multiple right-hand sides will be solved. The keywords `FACTOR_COORD` and `CONDITION` cannot be used together.

GWTH_FACTOR

Named variable into which the largest element in absolute value at any stage of the Gaussian elimination divided by the largest element in absolute value in A is stored.

GWTH_LIM

The computation stops if the growth factor exceeds `GWTH_LIMIT`. Default: `GWTH_LIMIT = 1.0e16`

HYBRID_DENSITY

Enable the function to switch to a dense factorization method when the density of the active submatrix reaches $0.0 \leq \text{Hybrid\_density} \leq 1.0$ and the order of the active submatrix is less than or equal to `Hybrid_order`. The keywords `HYBRID_DENSITY` and `HYBRID_ORDER` must be used together.

HYBRID_ORDER

Enable the function to switch to a dense factorization method when the density of the active submatrix reaches $0.0 \leq \text{Hybrid\_density} \leq 1.0$ and the order of the active submatrix is less than or equal to `Hybrid_order`. The keywords `HYBRID_DENSITY` and `HYBRID_ORDER` must be used together.

ITER_REFINE

If present and nonzero, iterative refinement will be applied.

PIVOTING

Scalar value specifying the pivoting method to use. For Row Markowitz, set `PIVOTING` to 1; for Column Markowitz, set `PIVOTING` to 2; and for Symmetric Markowitz, set `PIVOTING` to 3. Default: `PIVOTING = 3`

MEMORY_BLOCK

Supply the number of non-zeros which will be added to the factor if current allocations are inadequate. Default: `MEMORY_BLOCK = N_ELEMENTS(a)`

N_NONZERO

Named variable into which the total number of non-zeros in the factor is stored.

N_SEARCH_ROWS

The number of rows which have the least number of non-zero elements that will be searched for a pivot element. Default: `N_SEARCH_ROWS = 3`

SMALLEST_PVT

Named variable into which the value of the pivot element of smallest magnitude that occurred during the factorization is stored.

STABILITY

The absolute value of the pivot element must be bigger than the largest element in absolute value in its row divided by STABILITY. Default: STABILITY = 10.0

TOL_DROP

Possible fill-in is checked against this tolerance. If the absolute value of the new element is less than TOL_DROP, it will be discarded. Default: TOL_DROP = 0.0

TRANSPOSE

If present and nonzero, $A^T x = b$ is solved.

Discussion

The IMSL_SP_LUSOL function solves a system of linear equations $Ax = b$, where A is sparse. In its default usage, it solves the so-called one off problem, by first performing an LU factorization of A using the improved generalized symmetric Markowitz pivoting scheme. The factor L is not stored explicitly because the saxpy operations performed during the elimination are extended to the right-hand side, along with any row interchanges. Thus, the system $Ly = b$ is solved implicitly. The factor U is then passed to a triangular solver which computes the solution x from $Ux = y$.

If a sequence of systems $Ax = b$ are to be solved where A is unchanged, it is usually more efficient to compute the factorization once, and perform multiple forward and back solves with the various right-hand sides. In this case the factor L is explicitly stored and a record of all row as well as column interchanges is made. The solve step then solves the two triangular systems $Ly = b$ and $Ux = y$. In this case, you should first call [IMSL_SP_LUFAC](#) to compute the factorization, then use the keyword FACTOR_COORD with the IMSL_SP_LUSOL function.

If the solution to $A^T x = b$ is required, specify the keyword Transpose. This keyword only alters the forward elimination and back substitution so that the operations $U^T y = b$ and $L^T x = y$ are performed to obtain the solution. So, with one call to IMSL_SP_LUFAC to produce the factorization, solutions to both $Ax = b$ and $A^T x = b$ can be obtained.

The keyword CONDITION is used to calculate and return an estimation of the L1 condition number of A . The algorithm used is due to Higham. Specifying CONDITION causes a complete L to be computed and stored, even if a one-off problem is being solved. This is due to the fact that Higham's method requires a solution to problems of the form $Az = r$ and $A^T z = b$.

The default pivoting strategy is symmetric Markowitz (PIVOTING = 3). If a row or column oriented problem is encountered, there may be some reduction in fill-in by selecting either PIVOTING = 1 for Row Markowitz, or PIVOTING = 2 for column Markowitz. The Markowitz strategy will search a pre-elected number of rows or columns for pivot candidates. The default number is three, but this can be changed by using the keyword N_SEARCH_ROWS.

The keyword TOL_DROP can be used to set a tolerance which can reduce fill-in. This works by preventing any new fill element which has magnitude less than the specified drop tolerance from being added to the factorization. Since this can introduce substantial error into the factorization, it is recommended that the keyword ITER_REFINE be used to recover more accuracy in the final solution. The trade-off is between space savings from the drop tolerance and the extra time needed in repeated solve steps needed for refinement.

The IMSL_SP_LUSOL function provides the option of switching to a dense factorization method at some point during the decomposition. This option is enabled by specifying the keywords HYBRID_DENSITY and HYBRID_ORDER. HYBRID_DENSITY specifies a minimum density for the active submatrix before a format switch will occur. A density of 1.0 indicates complete fill-in. HYBRID_ORDER places an upper bound of the order of the active submatrix which will be converted to dense format. This is used to prevent a switch from occurring too early, possibly when the $O(n^3)$ nature of the dense factorization will cause performance degradation. Note that this option can significantly increase heap storage requirements.

Example

As an example, consider the following matrix:

$$A = \begin{bmatrix} 10 & 0 & 0 & 0 & 0 & 0 \\ 0 & 10 & -3 & -1 & 0 & 0 \\ 0 & 0 & 15 & 0 & 0 & 0 \\ -2 & 0 & 0 & 10 & -1 & 0 \\ -1 & 0 & 0 & -5 & 1 & -3 \\ -1 & -2 & 0 & 0 & 0 & 6 \end{bmatrix}$$

Let $x^T = (1, 2, 3, 4, 5, 6)$ so that $Ax = (10, 7, 45, 33, -34, 31)^T$. The number of nonzeros in A is 15.

```
A = replicate(ims1_f_sp_elem, 15)
```

```

; Define the sparse matrix A using coordinate storage format.
a(*).row = [0, 1, 1, 1, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5]
a(*).col = [0, 1, 2, 3, 2, 0, 3, 4, 0, 3, 4, 5, 0, 1, 5]
a(*).val = [10, 10, -3, -1, 15, -2, 10, -1, -1, -5, $
            1, -3, -1, -2, 6]
b = [10, 7, 45, 33, -34, 31]
; Define the right-hand side.
x = IMSL_SP_LUSOL(b, a)
; Call IMSL_SP_LUSOL, then print out result and the residual.
PM, x
    1.0000000
    2.0000000
    3.0000000
    4.0000000
    5.0000000
    6.0000000
PM, IMSL_SP_MVMUL(6, 6, a, x) - b
    0.0000000
   -8.8817842e-16
    0.0000000
    0.0000000
    0.0000000
    0.0000000

```

Version History

6.4	Introduced

See Also

[IMSL_SP_LUFAC](#)

IMSL_SP_LUFAC

The IMSL_SP_LUFAC function computes an LU factorization of a sparse matrix stored in either coordinate format or CSC format. Using keywords, any of several related computations can be performed.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_SP_LUFAC(a, n_rows [, CONDITION=variable] [, /CSC_COL]
    [, /CSC_ROW_IND] [, /CSC_VAL] [, GWTH_FACTOR=variable]
    [, GWTH_LIM=value] [, /HYBRID_DENSITY] [, /HYBRID_ORDER]
    [, /ITER_REFINE=value] [, MEMORY_BLOCK=value]
    [, N_NONZEROS=variable] [, N_SEARCH_ROWS=value]
    [, PIVOTING=value] [, SMALLEST_PVT=variable] [, STABILITY=value]
    [, TOL_DROP=value] [, TRANSPOSE=value])
```

Return Value

Structure containing the LU factorization of A.

Arguments

a

Sparse matrix stored as an array of structures containing the coefficient matrix $A(i,j)$. See “[Sparse Matrices: Direct Methods](#)” on page 67 and its related sections for a description of structures used for sparse matrices.

n_rows

The number of rows in *a*.

Keywords

CONDITION

Named variable into which an estimate of the L_1 condition number is stored.

CSC_COL

Accept the coefficient matrix in compressed sparse column (CSC) format. See [“Sparse Coordinate Storage Format”](#) on page 68 for a discussion of this storage scheme. The keywords CSC_COL, CSC_ROW_IND, and CSC_VAL must be used together.

CSC_ROW_IND

Accept the coefficient matrix in compressed sparse column (CSC) format. See [“Sparse Coordinate Storage Format”](#) on page 68 for a discussion of this storage scheme. The keywords CSC_COL, CSC_ROW_IND, and CSC_VAL must be used together.

CSC_VAL

Accept the coefficient matrix in compressed sparse column (CSC) format. See [“Sparse Coordinate Storage Format”](#) on page 68 for a discussion of this storage scheme. The keywords CSC_COL, CSC_ROW_IND, and CSC_VAL must be used together.

GWTH_FACTOR

Named variable into which the largest element in absolute value at any stage of the Gaussian elimination divided by the largest element in absolute value in A is stored.

GWTH_LIM

The computation stops if the growth factor exceeds GWTH_LIMIT. Default: GWTH_LIMIT = 1.0e16

HYBRID_DENSITY

Enable the function to switch to a dense factorization method when the density of the active submatrix reaches $0.0 \leq \text{HYBRID\_DENSITY} \leq 1.0$ and the order of the active submatrix is less than or equal to HYBRID_ORDER. The keywords HYBRID_DENSITY and HYBRID_ORDER must be used together.

HYBRID_ORDER

Enable the function to switch to a dense factorization method when the density of the active submatrix reaches $0.0 \leq \text{HYBRID\_DENSITY} \leq 1.0$ and the order of the active submatrix is less than or equal to **HYBRID_ORDER**. The keywords **HYBRID_DENSITY** and **HYBRID_ORDER** must be used together.

ITER_REFINE

If present and nonzero, iterative refinement will be applied.

MEMORY_BLOCK

Supply the number of non-zeros which will be added to the factor if current allocations are inadequate. Default: **MEMORY_BLOCK** = **N_ELEMENTS(a)**

N_NONZEROS

Named variable into which the total number of non-zeros in the factor is stored.

N_SEARCH_ROWS

The number of rows which have the least number of non-zero elements that will be searched for a pivot element. Default: **N_SEARCH_ROWS** = 3

PIVOTING

Scalar value specifying the pivoting method to use. For Row Markowitz, set **PIVOTING** to 1; for Column Markowitz, set **PIVOTING** to 2; and for Symmetric Markowitz, set **PIVOTING** to 3. Default: **PIVOTING** = 3

SMALLEST_PVT

Named variable into which the value of the pivot element of smallest magnitude that occurred during the factorization is stored.

STABILITY

The absolute value of the pivot element must be bigger than the largest element in absolute value in its row divided by **STABILITY**. Default: **STABILITY** = 10.0

TOL_DROP

Possible fill-in is checked against this tolerance. If the absolute value of the new element is less than **TOL_DROP**, it will be discarded. Default: **TOL_DROP** = 0.0

TRANSDPOSE

If present and nonzero, $A^T x = b$ is solved.

Discussion

The IMSL_SP_LUFAC function computes an LU factorization of A using the improved generalized symmetric Markowitz pivoting scheme.

If a sequence of systems $Ax = b$ are to be solved where A is unchanged, it is usually more efficient to compute the factorization once, and perform multiple forward and back solves with the various right-hand sides. In this case, the factor L is explicitly stored and a record of all rows as well as column interchanges is made. The solve step then solves the two triangular systems $Ly = b$ and $Ux = y$. In this case, first call IMSL_SP_LUFAC to compute the factorization, then use the keyword FACTOR_COORD with the [IMSL_SP_LUSOL](#) function.

If the solution to $A^T x = b$ is required, specify the keyword TRANSDPOSE. This keyword only alters the forward elimination and back substitution so that the operations $U^T y = b$ and $L^T x = y$ are performed to obtain the solution. So, with one call to IMSL_SP_LUFAC to produce the factorization, solutions to both $Ax = b$ and $A^T x = b$ can be obtained.

The keyword CONDITION is used to calculate and return an estimation of the L_1 condition number of A . The algorithm used is due to Higham. Specifying CONDITION causes a complete L to be computed and stored, even if a one-off problem is being solved. This is due to the fact that Higham's method requires solution to problems of the form $Az = r$ and $A^T z = r$.

The default pivoting strategy is symmetric Markowitz (PIVOTING = 3). If a row or column oriented problem is encountered, there may be some reduction in fill-in by selecting either PIVOTING = 1 for row Markowitz, or PIVOTING = 2 for column Markowitz. The Markowitz strategy will search a pre-elected number of rows or columns for pivot candidates. The default number is three, but this can be changed by using the keyword N_SEARCH_ROWS.

The keyword TOL_DROP can be used to set a tolerance which can reduce fill-in. This works by preventing any new fill element which has magnitude less than the specified drop tolerance from being added to the factorization. Since this can introduce substantial error into the factorization, it is recommended that the keyword ITER_REFINE be used to recover more accuracy in the final solution. The trade-off is between space savings from the drop tolerance and the extra time needed in repeated solve steps needed for refinement.

The `IMSL_SP_LUFAC` function provides the option of switching to a dense factorization method at some point during the decomposition. This option is enabled by specifying the keywords `HYBRID_DENSITY` and `HYBRID_ORDER`. `HYBRID_DENSITY` specifies a minimum density for the active submatrix before a format switch will occur. A density of 1.0 indicates complete fill-in. `HYBRID_ORDER` places an upper bound of the order of the active submatrix which will be converted to dense format. This is used to prevent a switch from occurring too early, possibly when the $O(n^3)$ nature of the dense factorization will cause performance degradation. Note that this option can significantly increase heap storage requirements.

Example

As an example, consider the following matrix:

$$A = \begin{bmatrix} 10 & 0 & 0 & 0 & 0 & 0 \\ 0 & 10 & -3 & -1 & 0 & 0 \\ 0 & 0 & 15 & 0 & 0 & 0 \\ -2 & 0 & 0 & 10 & -1 & 0 \\ -1 & 0 & 0 & -5 & 1 & -3 \\ -1 & -2 & 0 & 0 & 0 & 6 \end{bmatrix}$$

Let:

$$x_1^T = (1, 2, 3, 4, 5, 6)$$

so that:

$$x_1 = (10, 7, 45, 33, -34, 31)^T,$$

and let:

$$x_2^T = (5, 10, 15, 15, 10, 5)$$

so that:

$$Ax_2 = (50, 40, 225, 130, -85, 5)^T$$

This example factors A using `IMSL_SP_LUFAC`, and computes solutions to the systems $Ax_1 = b_1$ and $Ax_2 = b_2$ using the computed factor as input to [IMSL_SP_LUSOL](#).

```
A = replicate(imsf_sp_elem, 15)
; Define the sparse matrix A using coordinate storage format.
a(*).row = [0, 1, 1, 1, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5]
a(*).col = [0, 1, 2, 3, 2, 0, 3, 4, 0, 3, 4, 5, 0, 1, 5]
a(*).val = [10, 10, -3, -1, 15, -2, 10, -1, -1, -5, $
           1, -3, -1, -2, 6]
b1 = [10, 7, 45, 33, -34, 31]
b2 = [50, 40, 225, 130, -85, 5]
; Define the right-hand sides.
factor = IMSL_SP_LUFAC(a, 6)
; Compute the LU factorization.
x1 = IMSL_SP_LUSOL(b1, factor_coord = factor)
; Call IMSL_SP_LUSOL with factor and b1, then print result
; and the sum of the residuals.
PM, x1
```

```
1.0000000
2.0000000
3.0000000
4.0000000
5.0000000
6.0000000
PM, TOTAL(ABS(IMSL_SP_MVMUL(6, 6, a, x1) - b1))
8.8817842e-16
x2 = IMSL_SP_LUSOL(b2, factor_coord = factor)
; Call IMSL_SP_LUSOL with factor and b2, then print out
; result and the sum of the residuals.
PM, x2
5.0000000
10.000000
15.000000
15.000000
10.000000
5.0000000
PM, TOTAL(ABS(IMSL_SP_MVMUL(6, 6, a, x2) - b2))
1.4210855e-14
```

Version History

6.4	Introduced

See Also

[IMSL_SP_LUSOL](#)

IMSL_SP_BDSOL

The IMSL_SP_BDSOL function solves a general band system of linear equations $Ax = b$. By using keywords, any of several related computations can be performed.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_SP_BDSOL(b, nlca, nuca[, a] [, BLK_FACTOR=value]
    [, CONDITION=variable] [, /DOUBLE] [, FACTOR=array] [, PIVOT=array]
    [, TRANSPOSE=value] )
```

Return Value

A one-dimensional array containing the solution of the linear system $Ax = b$.

Arguments

b

One-dimensional matrix containing the right-hand side.

nlca

Number of lower codiagonals in a .

nuca

Number of upper codiagonals in a .

a

(Optional) Array of size $(nlca + nuca + 1) \times n$ containing the $n \times n$ banded coefficient matrix in band storage mode $A(i, j)$. See “[Band Storage Format](#)” on page 71 for a description of band storage mode.

Keywords

BLK_FACTOR

The blocking factor. This keyword must be set no larger than 32. Default: BLK_FACTOR = 1.

CONDITION

Named variable into which an estimate of the L_1 condition number is stored. This keyword cannot be used with PIVOT and FACTOR.

DOUBLE

If present and nonzero, double precision is used.

FACTOR

An array of size $(2*nlca + nuca + 1) \times N\_ELEMENTS(b)$ containing the *LU* factorization of *A* with column pivoting, as returned from [IMSL_SP_BDFAC](#). The keywords PIVOT and FACTOR must be used together. The keywords FACTOR and CONDITION cannot be used together.

PIVOT

One-dimensional array containing the pivot sequence. The keywords PIVOT and FACTOR must be used together. The keywords PIVOT and CONDITION cannot be used together.

TRANSPOSE

If present and nonzero, $A^T x = b$ is solved.

Discussion

The `IMSL_SP_BDSOL` function solves a system of linear algebraic equations with a real or complex band matrix *A*. It first computes the *LU* factorization of *A* with based on the blocked *LU* factorization algorithm given in Du Croz, et al, (1990). Level-3 BLAS invocations were replaced by in-line loops. The blocking factor BLK_FACTOR has the default value of 1, but can be reset to any positive value not exceeding 32.

The solution of the linear system is then found by solving two simpler systems, $y = L^{-1}b$ and $x = U^{-1}y$. When the solution to the linear system or the inverse of the

matrix is sought, an estimate of the L_1 condition number of A is computed using Higham's modifications to Hager's method, as given in Higham (1988). If the estimated condition number is greater than $1/\epsilon$ (where ϵ is the machine precision), a warning message is issued. This indicates that very small changes in A may produce large changes in the solution x . The `IMSL_SP_BDSOL` function fails if U , the upper triangular part of the factorization, has a zero diagonal element.

Example

Consider the 1000 x 1000 banded matrix below:

$$A = \begin{bmatrix} -1 & 4 & & & & \\ 4 & -1 & 4 & & & \\ & 4 & -1 & . & & \\ & & . & . & . & \\ & & & . & -1 & 4 \\ & & & & 4 & -1 & 4 \\ & & & & & 4 & -1 \end{bmatrix}$$

This example computes the solution to $Ax = b$, where b is a random vector.

```
n_rows = 1000L
nlca = 1L
nuca = 1L
a = DBLARR(n_rows*(nlca+nuca+1))
a(1:n_rows-1) = 4
a(n_rows:2*n_rows-1) = -1
a(2*n_rows:*) = 4
; Fill A with the values of the bands.
seed = 123L
b = RANDOMU(seed, n_rows)
; Compute a random right-hand side.
x = IMSL_SP_BDSOL(b, nlca, nuca, a)
; Compute the solution using IMSL_SP_BDSOL above,
; and output residual.
PM, TOTAL(ABS(IMSL_SP_MVMUL(n_rows, n_rows, nlca, nuca, a, x)-b))
1.2367884e-13
```

Version History

6.4	Introduced

See Also

[IMSL_SP_BDFAC](#)

IMSL_SP_BDFAC

The IMSL_SP_BDFAC procedure computes the *LU* factorization of a matrix stored in band storage mode.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

IMSL_SP_BDFAC, *nlca*, *nuca*, *n_rows*, *a*, *pivot*, *factor* [, [BLK_FACTOR=value](#)] [, [CONDITION=variable](#)] [, [/DOUBLE](#)]

Arguments

a

Array of size $(nlca + nuca + 1) \times n$ containing the $n \times n$ banded coefficient matrix in band storage mode $A(i,j)$. See “[Band Storage Format](#)” on page 71 for a description of band storage mode.

factor

A named variable that will contain an array of size $(2*nlca + nuca + 1) \times n\_rows$ containing the *LU* factorization of *A* with column pivoting. The keywords FACTOR and CONDITION cannot be used together.

n_rows

Number of rows in *a*.

nlca

Number of lower codiagonals in *a*.

nuca

Number of upper codiagonals in *a*.

pivot

A named variable that will contain a one-dimensional array containing the pivot sequence. The keywords PIVOT and CONDITION cannot be used together.

Keywords**BLK_FACTOR**

The blocking factor. This keyword must be set no larger than 32. Default: BLK_FACTOR = 1.

CONDITION

Named variable into which an estimate of the L_1 condition number is stored. The keyword CONDITION cannot be used with arguments *pivot* or *factor*.

DOUBLE

If present and nonzero, double precision is used.

Discussion

The IMSL_SP_BDFAC function computes the LU factorization of A with based on the blocked LU factorization algorithm given in Du Croz, et al, (1990). Level-3 BLAS invocations were replaced by in-line loops. The blocking factor BLK_FACTOR has the default value of 1, but can be reset to any positive value not exceeding 32.

An estimate of the L_1 condition number of A is computed using Higham's modifications to Hager's method, as given in Higham (1988). If the estimated condition number is greater than $1/\epsilon$ (where ϵ is the machine precision), a warning message is issued. This indicates that very small changes in A may produce large changes in the solution x .

Example

Consider the 1000 x 1000 banded matrix below:

$$A = \begin{bmatrix} -1 & 4 & & & & \\ 4 & -1 & 4 & & & \\ & 4 & -1 & . & & \\ & & . & . & . & \\ & & & . & -1 & 4 \\ & & & & 4 & -1 & 4 \\ & & & & & 4 & -1 \end{bmatrix}$$

This example computes the solution to $Ax_1 = b_1$ and $Ax_2 = b_2$, where b_1 and b_2 are random vectors. The factorization is computed just once, using `IMSL_SP_BDFAC`, and the solutions are computed using `IMSL_SP_BDSOL`.

```
n_rows = 1000L
nlca = 1L
nuca = 1L
a = DBLARR(n_rows*(nlca+nuca+1))
a(1:n_rows-1) = 4
a(n_rows:2*n_rows-1) = -1
a(2*n_rows:*) = 4
; Fill A with the values of the bands.
seed = 123L
b1 = RANDOMU(seed, n_rows)
b2 = RANDOMU(seed, n_rows)
; Fill random vectors
IMSL_SP_BDFAC, nlca, nuca, n_rows, a, pivot, factor
; Compute the factorization using IMSL_SP_BDFAC.
x1 = IMSL_SP_BDSOL(b1, nlca, nuca, Factor = factor, Pivot = pivot)
; Compute solution of Ax1 = b1 above, and output residual below.
PM, TOTAL(ABS(IMSL_SP_MVMUL(n_rows, n_rows, nlca, nuca, $
    a, x1)-b1))

1.2367884e-13

x2 = IMSL_SP_BDSOL(b2, nlca, nuca, Factor = factor, Pivot = pivot)
; Compute the solution of Ax2 = b2 above, and output residual.
PM, TOTAL(ABS(IMSL_SP_MVMUL(n_rows, n_rows, nlca, nuca, $
    a, x2)-b2))

9.1537888e-14
```

Version History

6.4	Introduced

See Also

[IMSL_SP_BDSOL](#)

IMSL_SP_PDSOL

The IMSL_SP_PDSOL function solves a sparse symmetric positive definite system of linear equations $Ax = b$.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_SP_PDSOL(b[, a] [, /CSC_COL] [, /CSC_ROW_IND]
                        [, /CSC_VAL] [, FACTOR=value] [, LG_DIAG=value]
                        [, MULTIFRONTAL=value] [, N_NONZERO=variable] [, SM_DIAG=value] )
```

Return Value

A one-dimensional array containing the solution of the linear system $Ax = b$.

Arguments

b

One-dimensional matrix containing the right-hand side.

a

(Optional) Sparse matrix stored as an array of structures containing non-zeros in lower triangle of the coefficient matrix $A(i,j)$. See “[Sparse Matrices: Direct Methods](#)” on page 67 and its related sections for a description of structures used for sparse matrices.

Keywords

CSC_COL

Accept the coefficient matrix in compressed sparse column (CSC) format. See “[Sparse Coordinate Storage Format](#)” on page 68 for a discussion of this storage scheme. The keywords CSC_COL, CSC_ROW_IND, and CSC_VAL must be used together.

CSC_ROW_IND

Accept the coefficient matrix in compressed sparse column (CSC) format. See [“Sparse Coordinate Storage Format”](#) on page 68 for a discussion of this storage scheme. The keywords CSC_COL, CSC_ROW_IND, and CSC_VAL must be used together.

CSC_VAL

Accept the coefficient matrix in compressed sparse column (CSC) format. See [“Sparse Coordinate Storage Format”](#) on page 68 for a discussion of this storage scheme. The keywords CSC_COL, CSC_ROW_IND, and CSC_VAL must be used together.

FACTOR

The factorization of A as computed by [IMSL_SP_PDFAC](#). If this keyword is used, then the argument `a` should not be used. This keyword is useful if solutions to systems involving the same coefficient matrix and multiple right-hand sides will be solved.

LG_DIAG

The largest diagonal element that occurred during the numeric factorization. This keyword is not valid if the keyword FACTOR is used.

MULTIFRONTAL

If `present` and `nonzero`, perform the numeric factorization using a multifrontal technique. By default a standard factorization is computed based on a sparse compressed storage scheme. The keywords MULTIFRONTAL and FACTOR cannot be used together.

N_NONZERO

Named variable into which the total number of non-zeros in the factor is stored. This keyword is not valid if the keyword FACTOR is used.

SM_DIAG

The smallest diagonal element that occurred during the numeric factorization. This keyword is not valid if the keyword FACTOR is used.

Discussion

The `IMSL_SP_PDSOL` function solves a system of linear algebraic equations having a sparse symmetric positive definite coefficient matrix A . In `IMSL_SP_PDSOL` default usage, a symbolic factorization of a permutation of the coefficient matrix is computed first, then a numerical factorization is performed. The solution of the linear system is then found using the numeric factor.

The symbolic factorization step of the computation consists of determining a minimum degree ordering and then setting up a sparse data structure for the Cholesky factor, L . This step only requires the “pattern” of the sparse coefficient matrix, that is, the locations of the non-zero elements but not any of the elements themselves.

The numerical factorization can be carried out in one of two ways. By default, the standard factorization is performed based on a sparse compressed storage scheme. This is fully described in George and Liu (1981). Optionally, a multifrontal technique can be used. The multifrontal method requires more storage but will be faster in certain cases. The multifrontal factorization is based on the routines in Liu (1987). For a detailed description of this method, see Liu (1990), also Duff and Reid (1983, 1984), Ashcraft (1987), Ashcraft et al. (1987), and Liu (1986, 1989).

If an application requires that several linear systems be solved where the coefficient matrix is the same but the right-hand sides change, the `IMSL_SP_PDFAC` function can be used to precompute the Cholesky factor. Then the keyword `FACTOR` can be used in `IMSL_SP_PDSOL` to efficiently solve all subsequent systems.

Given the numeric factorization, the solution x is obtained by the following calculations:

$$Ly_1 = Pb$$

$$L^T y_2 = y_1$$

$$x = P^T y_2$$

The permutation information, P , is carried in the numeric factor structure.

Example

As an example consider the 5 x 5 coefficient matrix:

$$a = \begin{bmatrix} 10 & 0 & 1 & 0 & 2 \\ 0 & 20 & 0 & 0 & 3 \\ 1 & 0 & 30 & 4 & 0 \\ 0 & 0 & 4 & 40 & 5 \\ 2 & 3 & 0 & 5 & 50 \end{bmatrix}$$

Let $x^T = (5, 4, 3, 2, 1)$ so that $Ax = (55, 83, 103, 97, 82)^T$. The number of non-zeros in the lower triangle of A is $\text{nz} = 10$. The sparse coordinate form for the lower triangle is given by:

row	0	1	2	2	3	3	4	4	4	4
col	0	1	0	2	2	3	0	1	3	4
val	10	20	1	30	4	40	2	3	5	50

Since this representation is not unique, an equivalent form would be:

row	3	4	4	4	0	1	2	2	3	4
col	3	0	1	3	0	1	0	2	2	4
val	40	2	3	5	10	20	1	30	4	50

```
A = REPLICATE(imsf_sp_elem, 10)
a(*).row = [0, 1, 2, 2, 3, 3, 4, 4, 4, 4]
a(*).col = [0, 1, 0, 2, 2, 3, 0, 1, 3, 4]
a(*).val = [10, 20, 1, 30, 4, 40, 2, 3, 5, 50]
b = [55.0d0, 83, 103, 97, 82]
x = IMSL_SP_PDSOL(b, a)
PM, x
5.0000000
4.0000000
3.0000000
2.0000000
```

1.0000000

Version History

6.4	Introduced

See Also

[IMSL_SP_PDFAC](#)

IMSL_SP_PDFAC

The IMSL_SP_PDFAC function computes a factorization of a sparse symmetric positive definite system of linear equations $Ax = b$.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_SP_PDFAC(a, n_rows [, /CSC_COL] [, /CSC_ROW_IND]  
[, /CSC_VAL] [, LG_DIAG=value] [, MULTIFRONTAL=value]  
[, N_NONZERO=variable] [, SM_DIAG=value] )
```

Return Value

The factorization of $Ax = b$.

Arguments

a

Sparse matrix stored as an array of structures containing non-zeros in lower triangle of the coefficient matrix $A(i,j)$. See “[Sparse Matrices: Direct Methods](#)” on page 67 and its related sections for a description of structures used for sparse matrices.

n_rows

The number of rows in a .

Keywords

CSC_COL

Accept the coefficient matrix in compressed sparse column (CSC) format. See “[Sparse Coordinate Storage Format](#)” on page 68 for a discussion of this storage scheme. The keywords CSC_COL, CSC_ROW_IND, and CSC_VAL must be used together.

CSC_ROW_IND

Accept the coefficient matrix in compressed sparse column (CSC) format. See [“Sparse Coordinate Storage Format”](#) on page 68 for a discussion of this storage scheme. The keywords CSC_COL, CSC_ROW_IND, and CSC_VAL must be used together.

CSC_VAL

Accept the coefficient matrix in compressed sparse column (CSC) format. See [“Sparse Coordinate Storage Format”](#) on page 68 for a discussion of this storage scheme. The keywords CSC_COL, CSC_ROW_IND, and CSC_VAL must be used together.

LG_DIAG

The largest diagonal element that occurred during the numeric factorization.

MULTIFRONTAL

If present and nonzero, perform the numeric factorization using a multifrontal technique. By default a standard factorization is computed based on a sparse compressed storage scheme

N_NONZERO

Specifies a named variable into which the total number of non-zeros in the factor is stored.

SM_DIAG

The smallest diagonal element that occurred during the numeric factorization.

Discussion

The IMSL_SP_PDFAC function computes a factorization of a sparse symmetric positive definite coefficient matrix A . In this function’s default usage, a symbolic factorization of a permutation of the coefficient matrix is computed first. Then a numerical factorization is performed.

The symbolic factorization step of the computation consists of determining a minimum degree ordering and then setting up a sparse data structure for the Cholesky factor, L . This step only requires the “pattern” of the sparse coefficient matrix, that is, the locations of the non-zero elements but not any of the elements themselves.

The numerical factorization can be carried out in one of two ways. By default, the standard factorization is performed based on a sparse compressed storage scheme. This is fully described in George and Liu (1981). Optionally, a multifrontal technique can be used. The multifrontal method requires more storage but will be faster in certain cases. The multifrontal factorization is based on the routines in Liu (1987). For a detailed description of this method, see Liu (1990), also Duff and Reid (1983, 1984), Ashcraft (1987), Ashcraft, et al. (1987), and Liu (1986, 1989).

If an application requires that several linear systems be solved where the coefficient matrix is the same but the right-hand sides change, IMSL_SP_PDFAC can be used to precompute the Cholesky factor. Then the keyword Factor can be used in [IMSL_SP_PDSOL](#) to efficiently solve all subsequent systems.

Given numeric factorization, x is obtained by the following calculations:

$$Ly_1 = Pb$$

$$L^T y_2 = y_1$$

$$x = P^T y_2$$

The permutation information, P , is carried in the numeric factor structure.

Example

As an example consider the 5 x 5 coefficient matrix:

$$a = \begin{bmatrix} 10 & 0 & 1 & 0 & 2 \\ 0 & 20 & 0 & 0 & 3 \\ 1 & 0 & 30 & 4 & 0 \\ 0 & 0 & 4 & 40 & 5 \\ 2 & 3 & 0 & 5 & 50 \end{bmatrix}$$

Let $x_1^T = (5, 4, 3, 2, 1)$ so that $Ax_1 = (55, 83, 103, 97, 82)^T$. Let $x_2^T = (1, 2, 3, 4, 5)$ so that $Ax_2 = (23, 55, 107, 197, 278)^T$. The number of non-zeros in the lower triangle of A is $nz = 10$. The sparse coordinate form for the lower triangle is given by:

row	0	1	2	2	3	3	4	4	4	4
col	0	1	0	2	2	3	0	1	3	4
val	10	20	1	30	4	40	2	3	5	50

Since this representation is not unique, an equivalent form would be:

row	3	4	4	4	0	1	2	2	3	4
col	3	0	1	3	0	1	0	2	2	4
val	40	2	3	5	10	20	1	30	4	50

```
A = REPLICATE(imsf_sp_elem, 10)
a(*).row = [0, 1, 2, 2, 3, 3, 4, 4, 4, 4]
a(*).col = [0, 1, 0, 2, 2, 3, 0, 1, 3, 4]
a(*).val = [10, 20, 1, 30, 4, 40, 2, 3, 5, 50]
b1 = [55, 83, 103, 97, 82]
b2 = [23, 55, 107, 197, 278]
factor = IMSL_SP_PDFAC(a, 5)
x1 = IMSL_SP_PDSOL(b1, FACTOR = factor)
PM, x1
5.0000000
4.0000000
3.0000000
2.0000000
1.0000000
x2 = IMSL_SP_PDSOL(b2, FACTOR = factor)
PM, x2
1.0000000
2.0000000
3.0000000
4.0000000
5.0000000
```

Version History

6.4	Introduced

See Also

[IMSL_SP_PDSOL](#)

IMSL_SP_BDPDSOL

The IMSL_SP_BDPDSOL function solves a symmetric positive definite system of linear equations $Ax = b$ in band symmetric storage mode. Using keywords, any of several related computations can be performed.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_SP_BDPDSOL(b, ncoda[, a] [, CONDITION=variable]
                        [, /DOUBLE] [, FACTOR=array])
```

Return Value

A one-dimensional array containing the solution of the linear system $Ax = b$.

Arguments

b

One-dimensional matrix containing the right-hand side.

ncoda

Number of upper codiagonals in *a*.

a

(Optional) Array of size $(ncoda + 1) \times n$ containing the $n \times n$ banded coefficient matrix in band symmetric storage mode $A(i, j)$. See “[Band Storage Format](#)” on page 71 for a description of band symmetric storage mode.

Keywords

CONDITION

Named variable into which an estimate of the L_1 condition number is stored. This keyword cannot be used if a previously computed factorization is specified with the keyword FACTOR.

DOUBLE

If present and nonzero, double precision is used.

FACTOR

An array of size $(ncoda + 1) \times N\_ELEMENTS(b)$ containing the $R^T R$ factorization of A in band symmetric storage mode, as returned from [IMSL_SP_BDPDFAC](#).

Discussion

The `IMSL_SP_BDPDSOL` function solves a system of linear algebraic equations with a symmetric positive definite band coefficient matrix A . It computes the $R^T R$ Cholesky factorization of A . R is an upper triangular band matrix.

The L_1 condition number of A is computed using Higham's modifications to Hager's method, as given in Higham (1988). If the estimated condition number is greater than $1/\epsilon$ (where ϵ is the machine precision), a warning message is issued. This indicates that very small changes in A may produce large changes in the solution x .

The `IMSL_SP_BDPDSOL` function fails if any submatrix of R is not positive definite or if R has a zero diagonal element. These errors occur only if A is very close to a singular matrix or to a matrix which is not positive definite.

The `IMSL_SP_BDPDSOL` function is partially based on the LINPACK subroutines CPBFA and SPBSL; see Dongarra et al. (1979).

Example

Solve a system of linear equations $Ax = b$, where:

$$A = \begin{bmatrix} 2 & 0 & -1 & 0 \\ 0 & 4 & 2 & 1 \\ -1 & 2 & 7 & -1 \\ 0 & 1 & -1 & 3 \end{bmatrix}$$

$$b = \begin{bmatrix} 6 \\ -11 \\ -11 \\ 19 \end{bmatrix}$$

```
n = 4L
ncoda = 2L
a = DBLARR((ncoda+1)*n)
a(0:n-1) = [0, 0, -1, 1]
a(n:2L*n-1) = [0, 0, 2, -1]
a(2L*n:*) = [2, 4, 7, 3]
; Define A in band symmetric storage mode.
b = [6, -11, -11, 19]
x = IMSL_SP_BDPDSOL(b, ncoda, a)
; Compute the solution
PM, x
      4.0000000
     -6.0000000
      2.0000000
      9.0000000
```

Version History

6.4	Introduced

See Also

[IMSL_SP_BDFAC](#)

[IMSL_SP_BDPDFAC](#)

IMSL_SP_BDPDFAC

The IMSL_SP_BDPDFAC function computes the $R^T R$ Cholesky factorization of symmetric positive definite matrix, A , in band symmetric storage mode.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_SP_BDPDFAC(a, n, ncoda [, CONDITION=variable]  
[, /DOUBLE])
```

Return Value

An array of size $(ncoda + 1) \times n$ containing the $R^T R$ factorization of A in band symmetric storage mode.

Arguments

a

Array of size $(ncoda + 1) \times n$ containing the $n \times n$ banded coefficient matrix in band symmetric storage mode $A(i,j)$. See “[Band Storage Format](#)” on page 71 for a description of band symmetric storage mode.

n

Number rows in a.

ncoda

Number of upper codiagonals in a.

Keywords

CONDITION

Specifies a named variable into which an estimate of the L1 condition number is stored.

DOUBLE

If present and nonzero, double precision is used.

Discussion

The IMSL_SP_BDPDFAC function computes the $R^T R$ Cholesky factorization of A . R is an upper triangular band matrix.

The L1 condition number of A is computed using Higham's modifications to Hager's method, as given in Higham (1988). If the estimated condition number is greater than $1/\epsilon$ (where ϵ is the machine precision), a warning message is issued. This indicates that very small changes in A may produce large changes in the solution x .

The IMSL_SP_BDPDFAC function fails if any submatrix of R is not positive definite or if R has a zero diagonal element. These errors occur only if A is very close to a singular matrix or to a matrix which is not positive definite.

The IMSL_SP_BDPDFAC function is partially based on the LINPACK subroutines CPBFA and SPBSL; see Dongarra et al. (1979).

Example

Solve a system of linear equations $Ax = b$, using both `IMSL_SP_BDPDFAC` and `IMSL_SP_BDPDSOL`, where:

$$A = \begin{bmatrix} 2 & 0 & -1 & 0 \\ 0 & 4 & 2 & 1 \\ -1 & 2 & 7 & -1 \\ 0 & 1 & -1 & 3 \end{bmatrix}$$

$$b = \begin{bmatrix} 6 \\ -11 \\ -11 \\ 19 \end{bmatrix}$$

```
n = 4L
ncoda = 2L
a = DBLARR((ncoda+1)*n)
a(0:n-1) = [0, 0, -1, 1]
a(n:2L*n-1) = [0, 0, 2, -1]
a(2L*n:*) = [2, 4, 7, 3]
; Define A in band symmetric storage mode.
b = [6, -11, -11, 19]
factor = IMSL_SP_BDPDFAC(a, n, ncoda)
; Use IMSL_SP_BDPDFAC to compute the factorization.
x = IMSL_SP_BDPDSOL(b, ncoda, Factor=factor)
; Compute the solution
PM, x
4.0000000
-6.0000000
2.0000000
9.0000000
```

Version History

6.4	Introduced

See Also

[IMSL_SP_BDFAC](#)

[IMSL_SP_BDPDSOL](#)

IMSL_SP_GMRES

The IMSL_SP_GMRES function solves a linear system $Ax = b$ using the restarted generalized minimum residual (GMRES) method.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_SP_GMRES(amultp, b [, /DOUBLE] [, HH_REORTH=value]  
[, ITMAX=value] [, MAX_KRYLOV=value] [, PRECOND=value]  
[, TOLERANCE=value])
```

Return Value

A one-dimensional array containing the solution of the linear system $Ax = b$.

Arguments

amultp

Scalar string specifying a user supplied function that computes $z = Ap$. The function accepts the argument p , and returns the vector Ap .

b

One-dimensional matrix containing the right-hand side.

Keywords

DOUBLE

If present and nonzero, double precision is used.

HH_REORTH

If present and nonzero, perform orthogonalization by Householder transformations, replacing the Gram-Schmidt process.

ITMAX

Initially set to the maximum number of GMRES iterations allowed. On exit, the number of iterations used is returned. Default: ITMAX = 1000

MAX_KRYLOV

The maximum Krylov subspace dimension, that is, the maximum allowable number of GMRES iterations allowed before restarting. Default:
 MAX_KRYLOV = Minimum(N_ELEMENTS(b), 20).

PRECOND

Scalar sting specifying a user supplied function which sets $z = M^{-1}r$, where M is the preconditioning matrix.

TOLERANCE

The algorithm attempts to generate x such that:

$$\|b - Ax\|_2 \leq \tau \|b\|_2$$

where $t = \text{TOLERANCE}$. Default: TOLERANCE = SQRT(machine precision).

Discussion

The IMSL_SP_GMRES, function based on the FORTRAN subroutine GMRES by H. F. Walker, solves the linear system $Ax = b$ using the GMRES method. This method is described in detail by Saad and Schultz (1986) and Walker (1988).

The GMRES method begins with an approximate solution x_0 and an initial residual $r_0 = b - Ax_0$. At iteration m , a correction z_m is determined in the Krylov subspace:

$$\kappa_m(v) = \text{span}(v, Av, \dots, A^{m-1}v)$$

$v = r_0$ which solves the least squares problem:

$$\min_{(z \in \kappa_m(r_0))} \|b - A(x_0 + z)\|_2$$

Then at iteration m , $x_m = x_0 + z_m$.

Orthogonalization by Householder transformations requires less storage but more arithmetic than Gram-Schmidt. However, Walker (1988) reports numerical

experiments which suggest the Householder approach is more stable, especially as the limits of residual reduction are reached.

Example

This example finds the solution to a linear system. The coefficient matrix is stored in coordinate format. Consider the following matrix:

$$A = \begin{bmatrix} 10 & 0 & 0 & 0 & 0 & 0 \\ 0 & 10 & -3 & -1 & 0 & 0 \\ 0 & 0 & 15 & 0 & 0 & 0 \\ -2 & 0 & 0 & 10 & -1 & 0 \\ -1 & 0 & 0 & -5 & 1 & -3 \\ -1 & -2 & 0 & 0 & 0 & 6 \end{bmatrix}$$

Let $x^T = (1, 2, 3, 4, 5, 6)$ so that $Ax = (10, 7, 45, 33, -34, 31)^T$. The number of nonzeros in A is 15.

```

FUNCTION Amultp, p
; This function uses IMSL_SP_MVMUL to multiply a sparse
; matrix stored in coordinate storage mode and a vector.
; The common block holds the sparse matrix.
    COMMON Gmres_ex1, nrows, ncols, a
    RETURN, IMSL_SP_MVMUL(nrows, ncols, a, p)
END

PRO Gmres1
; This procedure defines the sparse matrix A stored in coordinate
; storage mode, and then calls IMSL_SP_GMRES to compute the
; solution to Ax = b.
    COMMON Gmres_ex1, nrows, ncols, a
    ; Initialize sparse matrix structure variables
    @imsl_init

    A = REPLICATE(imsl_f_sp_elem, 15)
    a(*).row = [0, 1, 1, 1, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5]
    a(*).col = [0, 1, 2, 3, 2, 0, 3, 4, 0, 3, 4, 5, 0, 1, 5]
    a(*).val = [10, 10, -3, -1, 15, -2, 10, -1, -1, -5, $
        1, -3, -1, -2, 6]
    nrows = 6
    ncols = 6
    b = [10, 7, 45, 33, -34, 31]
    itmax = 10
    ; Itmax is input/output.

```

```
x = IMSL_SP_GMRES('amultp', b, Itmax = itmax)
pm, x, title = 'Solution to Ax = b'
pm, itmax, title = 'Number of iterations'
END
; Output of this procedure:
Solution to Ax = b
1.0000000
2.0000000
3.0000000
4.0000000
5.0000000
6.0000000
Number of iterations
6
```

Version History

6.4	Introduced

IMSL_SP_CG

The IMSL_SP_CG function solves a real symmetric definite linear system using a conjugate gradient method. A preconditioner can be supplied by using keywords.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_SP_CG(amultp, b [, /DOUBLE] [, ITMAX=value]  
[, JACOBI=vector] [, PRECOND=value] [, REL_ERR=value])
```

Return Value

A one-dimensional array containing the solution of the linear system $Ax = b$.

Arguments

amultp

Scalar string specifying a user supplied function which computes $z = Ap$. The function accepts the argument p , and returns the vector Ap .

b

One-dimensional matrix containing the right-hand side.

Keywords

DOUBLE

If present and nonzero, double precision is used.

ITMAX

Initially set to the maximum number of GMRES iterations allowed. On exit, the number of iterations used is returned. Default: ITMAX = 1000

JACOBI

If present, use the Jacobi preconditioner, that is, $M = \text{diag}(A)$. The supplied vector Jacobi should be set so that $\text{JACOBI}(i) = A_{i,i}$.

PRECOND

Scalar string specifying a user supplied function which sets $z = M^{-1}r$, where M is the preconditioning matrix.

REL_ERR

Initially set to relative error desired. On exit, the computed relative error is returned. Default: $\text{REL\_ERR} = \text{SQRT}(\text{machine precision})$

Discussion

The `IMSL_SP_CG` function solves the symmetric definite linear system $Ax = b$ using the conjugate gradient method with optional preconditioning. This method is described in detail by Golub and Van Loan (1983, chapter 10), and in Hageman and Young (1981, chapter 7).

The preconditioning matrix M , is a matrix that approximates A , and for which the linear system $Mz = r$ is easy to solve. These two properties are in conflict; balancing them is a topic of much current research. In the default usage of `IMSL_SP_CG`, $M = I$. If the keyword `JACOBI` is selected, M is set to the diagonal of A .

The number of iterations needed depends on the matrix and the error tolerance. As a rough guide:

$$Itmax = \sqrt{n}$$

for

$$n \gg 1$$

See the academic references for details.

Let M be the preconditioning matrix, let b , p , r , x , and z be vectors and let t be the desired relative error. Then the algorithm used is as follows:

$$\lambda = -1$$

$$p_0 = x_0$$

$$r_1 = b - Ap$$


```

for k = 1, ..., itmax
    zk = M-1rk
    if k = 1 then
        βk = 1
        pk = zk
    else
        βk = (zkTrk)/(zk-1Trk-1)
        pk = zk + βkpk
    endif
    zk = Ap
    αk = (zk-1Tzk-1)/(zkTpk)
    xk = xk + αkpk
    rk = rk - αkzk

    if(‖zk‖2 ≤ τ(1 - λ)‖xk‖2)then
        recompute λ
    if(‖zk‖2 ≤ τ(1 - λ)‖xk‖2)exit
    endif
endfor

```

Here λ is an estimate of $\lambda_{\max}(G)$, the largest eigenvalue of the iteration matrix $G = I - M^{-1}A$. The stopping criterion is based on the result (Hageman and Young, 1981, pages 148-151):

$$\frac{\|x_k - x\|_M}{\|x\|_M} \leq \left(\frac{1}{1 - \lambda_{\max}(G)} \right) \left(\frac{\|z_k\|_M}{\|x_k\|_M} \right)$$

where . It is also known that:

$$\|x\|_M^2 = x^T M x$$

$$\lambda_{\max}(T_1) \leq \lambda_{\max}(T_2) \leq \dots \leq \lambda_{\max}(G) < 1$$

where the T_n are the symmetric, tridiagonal matrices:

$$T_n = \begin{bmatrix} \mu_1 & \omega_2 & & \\ \omega_2 & \mu_2 & \omega_3 & \\ & \omega_3 & \mu_3 & \omega_4 \\ & & & \ddots \end{bmatrix}$$

with $\mu_k = 1 - \beta_k/\alpha_{k-1}$, $\mu_1 = 1 - 1/\alpha_1$, and $\omega_k = \text{SQRT}(\beta_k)/\alpha_{k-1}$. Usually the eigenvalue computation is needed for only a few of the iterations.

Example

This example finds the solution to a linear system. The coefficient matrix is stored as a full matrix.

```

FUNCTION Amultp, p
; Since A is in dense form, we use the # operator to perform the
; matrix-vector product. The common block is used to hold A.
COMMON Cg_comm1, a
RETURN, a#p
END
Pro CG_EX1
COMMON Cg_comm1, a
a = TRANSPOSE([ [ 1, -3, 2], [-3, 10, -5], [ 2, -5, 6] ])
b = [27, -78, 64]
x = IMSL_SP_CG('amultp', b)
; Use IMSL_SP_CG to compute the solution, then output
; the result.
PM, x, title = 'Solution to Ax = b'
END
; Output of this procedure:
Solution to Ax = b
1.0000000
-4.0000000
7.0000000

```

Version History

6.4	Introduced

IMSL_SP_MVMUL

The IMSL_SP_MVMUL function computes a matrix-vector product involving sparse matrix and a dense vector.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Matrix stored in coordinate format:

Result = IMSL_SP_MVMUL(*n_rows*, *n_cols*, *a*, *x* [, SYMMETRIC=*value*])

Matrix stored in band format:

Result = IMSL_SP_MVMUL(*n_rows*, *n_cols*, *nlca*, *nuca*, *a*, *x* [, SYMMETRIC=*value*])

Return Value

A one-dimensional array containing the product $Ax = b$.

Arguments

nrows

Number of rows in the matrix *a*.

ncols

Number of columns in the matrix *a*.

nlca

Number of lower codiagonals in *a*. *nuca* should be used if *a* is stored in band format.

nuca

Number of upper codiagonals in *a*. *nlca* should be used if *a* is stored in band format.

a

If in coordinate format, a sparse matrix stored as an array of structures. If banded, an array of size $(nlca + nuca + 1) \times nrows$ containing the $nrows \times ncols$ banded coefficient matrix in band storage mode. If banded, and the keyword SYMMETRIC is set, an array of size $(nlca + 1) \times nrows$ containing the $nrows \times ncols$ banded coefficient matrix in band symmetric storage mode $A(i,j)$. See “[Band Storage Format](#)” on page 71 for a description of band storage mode.

x

One-dimensional matrix containing the vector to be multiplied by a .

Keywords

SYMMETRIC

If present and nonzero, then a is stored in symmetric mode. If A is in coordinate format, then $Ax + A^T x - \text{diag}(A)$ is returned. If A is banded, then it must be in band symmetric storage mode. See “[Band Storage Format](#)” on page 71 for a description of band storage modes.

Discussion

The IMSL_SP_MVMUL function computes a matrix-vector product involving a sparse matrix and a dense vector.

If A is stored in coordinate format, then the arguments $nrows$, $ncols$, a , and x should be used. If the keyword SYMMETRIC is set, then $Ax + A^T x - \text{diag}(A)$ is returned.

If A is a banded, then the arguments $nrows$, $ncols$, $nlca$, $nuca$, a , and x should be used. If the keyword SYMMETRIC is set, then A must be in band symmetric storage mode, and the number of codiagonals should be used for both $nlca$ and $nuca$.

Examples

Example 1

This example computes Ax , where A is stored in coordinate format.

$$A = \begin{bmatrix} 10 & 0 & 0 & 0 & 0 & 0 \\ 0 & 10 & -3 & -1 & 0 & 0 \\ 0 & 0 & 15 & 0 & 0 & 0 \\ -2 & 0 & 0 & 10 & -1 & 0 \\ -1 & 0 & 0 & -5 & 1 & -3 \\ -1 & -2 & 0 & 0 & 0 & 6 \end{bmatrix}$$

Let $x^T = (1, 2, 3, 4, 5, 6)$

```
A = replicate(imsl_f_sp_elem, 15)
; Define the sparse matrix A using coordinate storage format.
a(*).row = [0, 1, 1, 1, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5]
a(*).col = [0, 1, 2, 3, 2, 0, 3, 4, 0, 3, 4, 5, 0, 1, 5]
a(*).val = [10, 10, -3, -1, 15, -2, 10, -1, -1, -5, $
           1, -3, -1, -2, 6]
x = [1, 2, 3, 4, 5, 6]
ax = IMSL_SP_MVMUL(6, 6, a, x)
PM, ax
10.000000
7.0000000
45.000000
33.000000
-34.000000
31.000000
```

Example 2

This example computes Ax , where A is stored in band mode. Consider the 1000 x 1000 banded matrix below:

$$A = \begin{bmatrix} -1 & 4 & & & & \\ 4 & -1 & 4 & & & \\ & 4 & -1 & . & & \\ & & . & . & . & \\ & & & . & -1 & 4 \\ & & & & 4 & -1 & 4 \\ & & & & & 4 & -1 \end{bmatrix}$$

Let $x(*) = 2$.

```
n_rows = 1000L
nlca = 1L
nuca = 1L
a = DBLARR(n_rows*(nlca+nuca+1))
a(1:n_rows-1) = 4
a(n_rows:2*n_rows-1) = -1
a(2*n_rows:*) = 4
; Fill A with the values of the bands.
x = DBLARR(n_rows)
x(*) = 2
; Fill up x.
expected = DBLARR(n_rows)
expected(*) = 14
expected(0) = 6
expected(n_rows-1) = 6
; Define the expected result.
ax = IMSL_SP_MVMUL(n_rows, n_rows, nlca, nuca, a, x)
; Compute the product, then output the difference between the
; computed result and the expected result.
PRINT, TOTAL(ABS(ax-expected))
      0.0000000
```

Example 3

This example computes Ax , where A is stored in band symmetric mode. Let ,

$$A = \begin{bmatrix} 2 & 0 & -1 & 0 \\ 0 & 4 & 2 & 1 \\ -1 & 2 & 7 & -1 \\ 0 & 1 & -1 & 3 \end{bmatrix}$$

$$x = \begin{bmatrix} 4 \\ -6 \\ 2 \\ 9 \end{bmatrix}$$

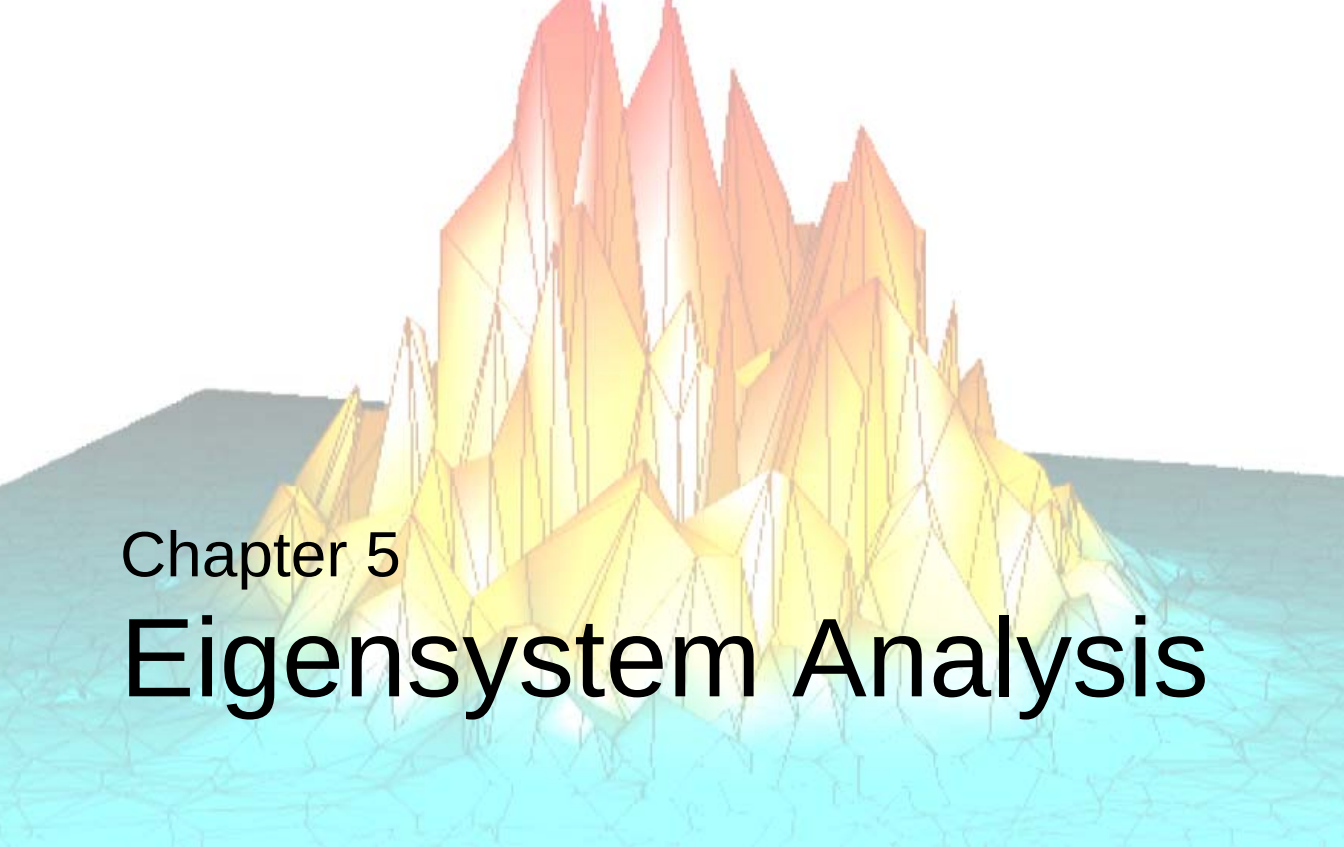
```

n = 4L
ncoda = 2L
a = DBLARR((ncoda+1)*n)
a(0:n-1) = [0, 0, -1, 1]
a(n:2L*n-1) = [0, 0, 2, -1]
a(2L*n:*) = [2, 4, 7, 3]
; Fill up contents of A.
x = [4, -6, 2, 9]
ax = IMSL_SP_MVMUL(n, n, ncoda, ncoda, a, x, /Symmetric)
; Call IMSL_SP_MVMUL with the keyword Symmetric set.
PM, ax
    6.0000000
   -11.000000
   -11.000000
    19.000000

```

Version History

6.4	Introduced



Chapter 5

Eigensystem Analysis

This section contains the following topics:

Overview: Eigensystem Analysis	174	Eigensystem Routines	177

Overview: Eigensystem Analysis

An ordinary linear eigensystem problem is represented by the equation $Ax = \lambda x$, where A denotes an $n \times n$ matrix. The value λ is an *eigenvalue*, and $x \neq 0$ is the corresponding *eigenvector*. The eigenvector is determined up to a scalar factor. In all functions, this factor is chosen so that x has Euclidean length 1, and the component of x of largest magnitude is positive. If x is a complex vector, this component of largest magnitude is scaled to be real and positive. The entry where this component occurs can be arbitrary for eigenvectors having non-unique maximum magnitude values.

A generalized linear eigensystem problem is represented by $Ax = \lambda Bx$, where A and B are $n \times n$ matrices. The value λ is a generalized eigenvalue, and x is the corresponding generalized eigenvector. The generalized eigenvectors are normalized in the same manner as for ordinary eigensystem problems.

Error Analysis and Accuracy

This section discusses ordinary eigenvalue problems. Except in special cases, functions do not return the exact eigenvalue-eigenvector pair for the ordinary eigenvalue problem $Ax = \lambda x$. Typically, the computed pair:

$$\tilde{x}, \tilde{\lambda}$$

is an exact eigenvector-eigenvalue pair for a “nearby” matrix $A + E$. Information about E is known only in terms of bounds of the form:

$$\|E\|_2 \leq f(n) \|A\|_2 \epsilon$$

The value of $f(n)$ depends on the algorithm but is typically a small fractional power of n . The parameter ϵ is the machine precision. The following is by a theorem due to Bauer and Fike (see Golub and Van Loan 1989, p. 342):

$$\min |\hat{\lambda} - \lambda| \leq \kappa(X) \|E\|_2 \quad \text{for all } \lambda \text{ in } \sigma(A)$$

where $\sigma(A)$ is the set of all eigenvalues of A (called the spectrum of A), X is the matrix of eigenvectors:

$$\|\cdot\|_2$$

is Euclidean length, and $\kappa(X)$ is the condition number of X defined as:

$$\kappa(X) = \|X\|_2 \|X^{-1}\|_2$$

If A is a real symmetric or complex Hermitian matrix, then its eigenvector matrix X is respectively orthogonal or unitary. For these matrices, $\kappa(X) = 1$.

The accuracy of the computed eigenvalues:

$$\tilde{\lambda}_j \text{ and eigenvectors } \tilde{x}_j$$

can be checked by computing their performance index τ . The performance index is defined to be:

$$\tau = \max_{1 \leq j \leq n} \frac{\|A\tilde{x}_j - \tilde{\lambda}_j\tilde{x}_j\|_2}{n\epsilon\|A\|_2\|\tilde{x}_j\|_2}$$

where ϵ is again the machine precision.

The performance index τ is related to the error analysis because:

$$\|E\tilde{x}_j\|_2 = \|A\tilde{x}_j - \tilde{\lambda}_j\tilde{x}_j\|_2$$

where E is the “nearby” matrix discussed above.

While the exact value of τ is precision and data dependent, the performance of an eigensystem analysis function is defined as excellent if $\tau < 1$, good if $1 \leq \tau \leq 100$, and poor if $\tau > 100$. This is an arbitrary definition, but large values of τ can serve as a warning that there is a significant error in the calculation.

If the condition number $\kappa(X)$ of the eigenvector matrix X is large, there can be large errors in the eigenvalues even if τ is small. In particular, it is often difficult to recognize near-multiple eigenvalues or unstable mathematical problems from numerical results. This facet of the eigenvalue problem is often difficult for users to understand. Suppose the accuracy of an individual eigenvalue is desired. This can be answered approximately by computing the *condition number of an individual eigenvalue* (see Golub and Van Loan 1989, pp. 344–345). For matrices A such that the computed array of normalized eigenvectors X is invertible, the condition number of λ_j is:

$$\kappa_j = \|e_j^T X^{-1}\|$$

the Euclidean length of the j -th row of X^{-1} . An approximate bound for the accuracy of a computed eigenvalue is then given by:

$$\kappa \in \|A\|$$

To compute an approximate bound for the relative accuracy of an eigenvalue, divide this bound by $|\lambda_j|$.

Reformulating Generalized Eigenvalue Problems

The generalized eigenvalue problem $Ax = \lambda Bx$ is often difficult to analyze because it is frequently ill-conditioned. Occasionally, there are changes of variables that can be

performed on the given problem to ease this ill-conditioning. Using an example where B is singular, but A is nonsingular, define the reciprocal $\mu = \lambda^{-1}$, then the roles of A and B are interchanged so that the reformulated problem $Bx = \mu Ax$ is solved. Those generalized eigenvalues $\mu_j = 0$ correspond to eigenvalues $\lambda_j = \text{infinity}$. The remaining $\lambda_j = \mu_j^{-1}$. The generalized eigenvectors for λ_j correspond to those for μ_j .

If B is nonsingular, you can solve the ordinary eigenvalue problem $Cx = \lambda x$, where $C = B^{-1}A$. Matrix C is subject to perturbations due to ill-conditioning and rounding errors when computing $B^{-1}A$. Computing condition numbers of the eigenvalues for C may, however, be helpful for analyzing the accuracy of results for the generalized problem.

Another method to consider to reduce the generalized problem to an alternate ordinary problem: first compute a matrix decomposition $B = PQ$, where both P and Q are matrices that are “simple” to invert. Then, the given generalized problem is equivalent to the ordinary eigenvalue problem $Fy = \lambda y$. The matrix $F = P^{-1}AQ^{-1}$ and the unnormalized eigenvectors of the generalized problem are given by $x = Q^{-1}y$. An example of this reformulation is used in the case where A and B are real and symmetric, with B positive definite. [IMSL_EIGSYMGGEN](#) uses $P = R^T$ and $Q = R$, where R is an upper-triangular matrix obtained from a Cholesky decomposition, $B = R^T R$. The matrix $F = R^{-T} A R^{-1}$ is symmetric and real. Computation of the eigenvalue-eigenvector expansion for F is based on the [IMSL_EIG](#) function.

Eigensystem Routines

Linear Eigensystem Problems

[IMSL_EIG](#)—General and symmetric matrices.

Generalized Eigensystem Problems

[IMSL_EIGSYMGEN](#)—Real symmetric matrices and B positive definite.

[IMSL_GENEIG](#)—General eigenexpansion of $Ax=\lambda Bx$.

IMSL_EIG

The IMSL_EIG function computes the eigenexpansion of a real or complex matrix *A*. If the matrix is known to be symmetric or Hermitian, a keyword can be used to trigger more efficient algorithms.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_EIG(a [, /DOUBLE] [, /LOWER_LIMIT] [, NUMBER=value]  
[, /SYMMETRIC=value] [, /UPPER_LIMIT] [, VECTORS=variable])
```

Return Value

A one-dimensional matrix containing the complex eigenvalues of the matrix.

Arguments

a

Two-dimensional matrix containing the data.

Keywords

DOUBLE

If present and nonzero, double precision is used.

LOWER_LIMIT

Forces the IMSL_EIG function to return the eigenvalues and, optionally, eigenvectors that lie in the interval within the lower limit LOWER_LIMIT and upper limit UPPER_LIMIT. If LOWER_LIMIT is specified, the keywords UPPER_LIMIT and SYMMETRIC must also be specified. Default: (LOWER_LIMIT, UPPER_LIMIT) = (-infinity, +infinity)

NUMBER

Number of eigenvalues and eigenvectors in the range (LOWER_LIMIT, UPPER_LIMIT). This keyword is only available if also using the keyword SYMMETRIC.

SYMMETRIC

If present and nonzero, a is assumed to be symmetric in the real case and Hermitian in the complex case. Using SYMMETRIC triggers the use of a more appropriate algorithm for symmetric and Hermitian matrices.

UPPER_LIMIT

Forces the IMSL_EIG function to return the eigenvalues and, optionally, eigenvectors that lie in the interval within the lower limit LOWER_LIMIT and upper limit UPPER_LIMIT. If UPPER_LIMIT is specified, SYMMETRIC and LOWER_LIMIT must also be specified. Default: (LOWER_LIMIT, UPPER_LIMIT) = $(-\infty, +\infty)$

VECTORS

The named variable into which the two-dimensional array containing the eigenvectors of the matrix a is stored.

Discussion

If A is a real, general matrix, the IMSL_EIG function computes the eigenvalues of A by a two-phase process. The matrix is reduced to upper Hessenberg form by elementary orthogonal or Gauss similarity transformations, then the eigenvalues are computed using a QR or combined LR - QR algorithm (Golub and Van Loan 1989, pp. 373–382, and Watkins and Elsner 1990). The combined LR - QR algorithm is based on an implementation by Jeff Haag and David Watkins. Eigenvectors are then calculated as required. When eigenvectors are computed, the QR algorithm is used to compute the eigenexpansion. When only eigenvalues are required, the combined LR - QR algorithm is used.

If A is a complex, general matrix, the IMSL_EIG function computes the eigenvalues of A by a two-phase process. The matrix is reduced to upper Hessenberg form by elementary Gauss transformations, then the eigenvalues are computed using an explicitly shifted LR algorithm. Eigenvectors are calculated during the iterations for the eigenvalues (Martin and Wilkinson 1971).

If A is a real, symmetric matrix and the keyword `SYMMETRIC` is used, the `IMSL_EIG` function computes the eigenvalues of A by a two-phase process. The matrix is reduced to tridiagonal form by elementary orthogonal similarity transformations, then the eigenvalues are computed using a rational `QR` or bisection algorithm. Eigenvectors are calculated as required (see Parlett 1980, pp. 169–173).

If A is a complex, Hermitian matrix and the keyword `SYMMETRIC` is used, the `IMSL_EIG` function computes the eigenvalues of A by a two-phase process. The matrix is reduced to tridiagonal form by elementary orthogonal similarity transformations, then the eigenvalues are computed using a rational `QR` or bisection algorithm. Eigenvectors are calculated as required.

If keyword `SYMMETRIC` is used, it is possible to force the `IMSL_EIG` function to return the eigenvalues and, optionally, eigenvectors that lie in a specified interval. The interval is defined using keywords `LOWER_LIMIT` and `UPPER_LIMIT`. The `NUMBER` keyword is provided to return the number of elements of the returned array that contain valid eigenvalues. The first `NUMBER` elements of the returned array contain the computed eigenvalues, and all remaining elements contain NaN (Not a Number).

Examples

Example 1

This example computes the eigenvalues of a real 3-by-3 matrix.

```
RM, a, 3, 3
; Define the matrix.
row 0:  8  -1  -5
row 1: -4   4  -2
row 2: 18  -5  -7
eigval = IMSL_EIG(a)
; Call IMSL_EIG to compute the eigenvalues.
PM, eigval, Title = 'Eigenvalues of A'
; Output the results.
Eigenvalues of A
( 2.00000,  4.00001)
( 2.00000, -4.00001)
( 1.00000,  0.00000)
```


Example 2

This example is a variation of the first example. It computes the eigenvectors as well as the eigenvalues.

```
RM, a, 3, 3
; Define the 3-by-3 matrix.
row 0:  8 -1 -5
row 1: -4  4 -2
row 2: 18 -5 -7
eigval = IMSL_EIG(a, Vectors = eigvec)
; Call IMSL_EIG using keyword Vectors to specify named
; variable into which the eigenvectors are stored.
PM, eigval, Title = 'Eigenvalues of A'
; Output the eigenvalues.
Eigenvalues of A
  ( 2.00000,  4.00000)
  ( 2.00000, -4.00000)
  ( 1.00001,  0.00000)
PM, eigvec, Title = 'Eigenvectors of A'
; Output the eigenvectors.
Eigenvectors of A
  ( 0.316228, 0.316228) ( 0.316228, -0.316228)
  ( 0.408248, 0.00000)
  ( 2.08616e-07, 0.632455) ( 2.08616e-07, -0.632455)
  ( 0.816497, 0.00000)
  ( 0.632456, 0.00000) ( 0.632456, 0.00000)
  ( 0.408247, 0.00000)
```

Example 3

This example computes Eigenvalues of a complex matrix.

```
RM, a, 4, 4, /Complex
; Define a complex matrix.
row 0: (5, 9) (5,  5) (-6, -6) (-7, -7)
row 1: (3, 3) (6, 10) (-5, -5) (-6, -6)
row 2: (2, 2) (3,  3) (-1,  3) (-5, -5)
row 3: (1, 1) (2,  2) (-3, -3) ( 0,  4)
eigval = IMSL_EIG(a)
; Call IMSL_EIG to compute the eigenvalues.
PM, eigval, Title = 'Eigenvalues of A'
; Output the results.
Eigenvalues of A
  ( 4.00000, 8.00000)
  ( 3.00000, 7.00000)
  ( 2.00000, 6.00000)
  ( 1.00000, 5.00000)
```

Errors

Warnings

MATH_SLOW_CONVERGENCE_GEN—Iteration for an eigenvalue did not converge after # iterations.

Version History

6.4	Introduced

IMSL_EIGSYMGEN

The IMSL_EIGSYMGEN function computes the generalized eigenexpansion of a system $Ax = \lambda Bx$. The matrices A and B are real and symmetric, and B is positive definite.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_EIGSYMGEN(*a*, *b* [, /DOUBLE] [, VECTORS=*array*])

Return Value

One-dimensional array containing the eigenvalues of the symmetric matrix.

Arguments

a

Two-dimensional matrix containing symmetric coefficient matrix A .

b

Two-dimensional matrix containing the positive definite symmetric coefficient matrix B .

Keywords

DOUBLE

If present and nonzero, double precision is used.

VECTORS

Compute eigenvectors of the problem. A two-dimensional array containing the eigenvectors is returned in the variable name specified by VECTORS.

Discussion

The IMSL_EIGSYMGEN function computes the eigenvalues of a symmetric, positive definite eigenvalue problem by a three-phase process (Martin and Wilkinson 1971). Matrix B is reduced to factored form using the Cholesky decomposition. These factors are used to form a congruence transformation that yields a symmetric real matrix whose eigenexpansion is obtained. The problem is then transformed back to the original coordinates. Eigenvectors are calculated and transformed as required.

Examples

Example 1

This example computes the generalized eigenexpansion of a system $Ax = \lambda Bx$, where A and B are 3-by-3 matrices.

```
RM, a, 3, 3
; Define the matrix A.
row 0: 1.1 1.2 1.4
row 1: 1.2 1.3 1.5
row 2: 1.4 1.5 1.6
RM, b, 3, 3
; Define the matrix B.
row 0: 2 1 0
row 1: 1 2 1
row 2: 0 1 2
eigval = IMSL_EIGSYMGEN(a, b)
; Call IMSL_EIGSYMGEN to compute the eigenexpansion.
PM, eigval, Title = 'Eigenvalues'
; Output the results.
Eigenvalues
    1.38644
   -0.0583479
   -0.00309042
```

Example 2

This example is a variation of the first example. It computes the eigenvectors as well as the eigenvalues.

```
RM, a, 3, 3
; Define the matrix A.
row 0: 1.1 1.2 1.4
row 1: 1.2 1.3 1.5
row 2: 1.4 1.5 1.6
RM, b, 3, 3
; Define the matrix B.
```

```

row 0: 2 1 0
row 1: 1 2 1
row 2: 0 1 2
eigval = IMSL_EIGSYMGEN(a, b, Vectors = eigvec)
; Call IMSL_EIGSYMGEN with keyword Vectors to specify the named
; variable in which the vectors are stored.
PM, eigval, Title = 'Eigenvalues'
; Output the eigenvalues.

```

```

Eigenvalues
    1.38644
   -0.0583478
   -0.00309040
PM, eigvec, Title = 'Eigenvectors'
; Output the eigenvectors.
Eigenvectors
    0.643094    -0.114730    -0.681688
   -0.0223849   -0.687186     0.726597
    0.765460     0.717365    -0.0857800

```

Errors

Warning Errors

MATH_SLOW_CONVERGENCE_SYM—Iteration for an eigenvalue failed to converge in 100 iterations before deflating.

Fatal Errors

MATH_SUBMATRIX_NOT_POS_DEFINITE—Leading submatrix of the input matrix is not positive definite.

MATH_MATRIX_B_NOT_POS_DEFINITE—Matrix B is not positive definite.

Version History

6.4	Introduced

IMSL_GENEIG

The IMSL_GENEIG procedure computes the generalized eigenexpansion of a system $Ax = \lambda Bx$.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

IMSL_GENEIG, *a*, *b*, *alpha*, *beta* [, /DOUBLE] [, VECTORS=*variable*]

Arguments

a

Two-dimensional array of size n -by- n containing coefficient matrix A .

alpha

One-dimensional array of size n containing scalars α_i . If $\beta_i \neq 0$, $\lambda_i = \alpha_i / \beta_i$ for $i = 0, \dots, n - 1$ are the eigenvalues of the system.

b

Two-dimensional array of size n -by- n containing coefficient matrix B .

beta

One-dimensional array of size n .

Keywords

DOUBLE

If present and nonzero, double precision is used.

VECTORS

Named variable into which a two-dimensional array of size n -by- n containing eigenvectors of the problem is stored. Each vector is normalized to have Euclidean length equal to one.

Discussion

The IMSL_GENEIG function uses the QZ algorithm to compute the eigenvalues and eigenvectors of the generalized eigensystem $Ax = \lambda Bx$, where A and B are matrices of order n . The eigenvalues for this problem can be infinite, so α and β are returned instead of λ . If β is nonzero, $\lambda = \alpha/\beta$.

The QZ algorithm first simultaneously reduces A to upper-Hessenberg form and B to upper-triangular form, then it uses orthogonal transformations to reduce A to quasi-upper-triangular form while keeping B upper triangular. The generalized eigenvalues and eigenvectors for the reduced problem are then computed.

The IMSL_GENEIG function is based on the QZ algorithm due to Moler and Stewart (1973), as implemented by the EISPACK routines QZHES, QZIT and QZVAL; see Garbow et al. (1977).

Examples

Example 1

This example computes the eigenvalue, λ , of system $Ax = \lambda Bx$, where:

$$A = \begin{bmatrix} 1.0 & 0.5 & 0.0 \\ -10.0 & 2.0 & 0.0 \\ 5.0 & 1.0 & 0.5 \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} 0.5 & 0.0 & 0.0 \\ 3.0 & 3.0 & 0.0 \\ 4.0 & 0.5 & 1.0 \end{bmatrix}$$

```
a = TRANSPOSE([[1.0, 0.5, 0.0], [-10.0, 2.0, 0.0], $
               [5.0, 1.0, 0.5]])
b = TRANSPOSE([[0.5, 0.0, 0.0], [3.0, 3.0, 0.0], $
               [4.0, 0.5, 1.0]])
; Compute eigenvalues
IMSL_GENEIG, a, b, alpha, beta
; Print eigenvalues
PM, alpha/beta, Title = 'Eigenvalues'
Eigenvalues
( 0.833334, 1.99304)
( 0.833333, -1.99304)
( 0.500000, 0.00000)
```

Example 2

This example finds the eigenvalues and eigenvectors of the same eigensystem given in the last example.

```
a = TRANSPOSE([[1.0, 0.5, 0.0], [-10.0, 2.0, 0.0], $
               [5.0, 1.0, 0.5]])
b = TRANSPOSE([[0.5, 0.0, 0.0], [3.0, 3.0, 0.0], $
               [4.0, 0.5, 1.0]])
; Compute eigenvalues
IMSL_GENEIG, a, b, alpha, beta, Vectors = vectors
; Print eigenvalues
PM, alpha/beta, Title = 'Eigenvalues'
Eigenvalues
  ( 0.833332, 1.99304)
  ( 0.833332, -1.99304)
  ( 0.500000, -0.00000)
; Print eigenvectors
PM, vectors, Title = 'Eigenvectors'
Eigenvectors
  ( -0.197112, 0.149911) ( -0.197112, -0.149911)
  ( -1.53306e-08, 0.00000)
  ( -0.0688163, -0.567750) ( -0.0688163, 0.567750)
  ( -4.75248e-07, 0.00000)
  ( 0.782047, 0.00000) ( 0.782047, 0.00000)
  ( 1.00000, 0.00000)
```

Example 3

This example solves the eigenvalue, λ , of system $Ax = \lambda Bx$, where:

$$A = \begin{bmatrix} 1 & 0.5+i & 5i \\ -10 & 2+i & 0 \\ 5+i & 1 & 0.5+3i \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} 0.5 & 0 & 0 \\ 3+3i & 3+3i & i \\ 4+2i & 0.5+i & 1+i \end{bmatrix}$$

```
a = TRANSPOSE([$
  [COMPLEX(1.0, 0.0), COMPLEX(0.5, 1.0), COMPLEX(0.0, 5.0)], $
  [COMPLEX(-10.0, 0.0), COMPLEX(2.0, 1.0), COMPLEX(0.0, 0.0)], $
  [COMPLEX(5.0, 1.0), COMPLEX(1.0, 0.0), COMPLEX(0.5, 3.0)]])
b = TRANSPOSE([$
  [COMPLEX(0.5, 0.0), COMPLEX(0.0, 0.0), COMPLEX(0.0, 0.0)], $
  [COMPLEX(3.0, 3.0), COMPLEX(3.0, 3.0), COMPLEX(0.0, 1.0)], $
  [COMPLEX(4.0, 2.0), COMPLEX(0.5, 1.0), COMPLEX(1.0, 1.0)]])
; Compute eigenvalues
IMSL_GENEIG, a, b, alpha, beta
; Print eigenvalues
PM, alpha/beta, Title = 'Eigenvalues'
Eigenvalues
```



```
( -8.18016, -25.3799)
( 2.18006, 0.609113)
( 0.120108, -0.389223)
```

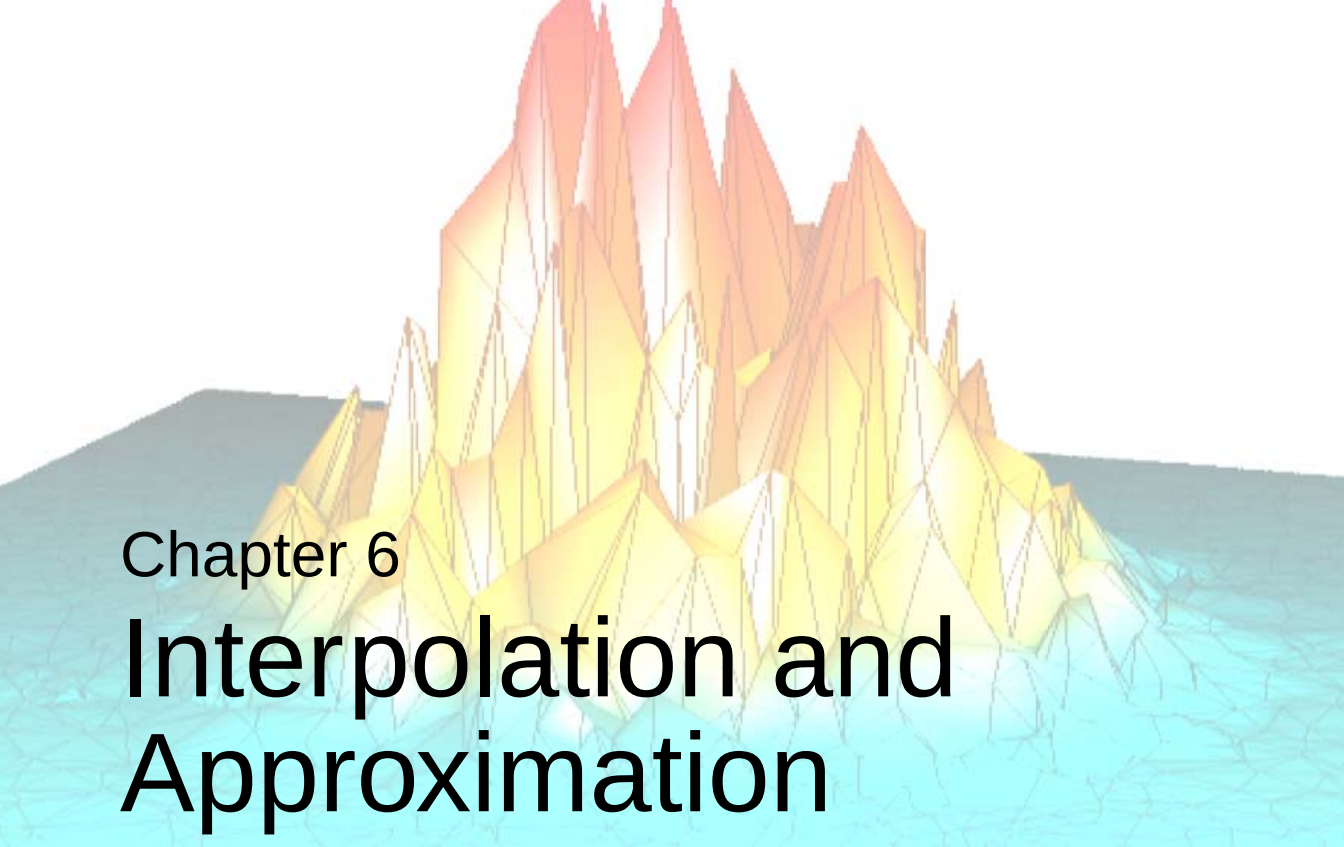
Example 4

This example finds the eigenvalues and eigenvectors of the same eigensystem given in the last example.

```
a = TRANSPOSE([$
[COMPLEX(1.0, 0.0), COMPLEX(0.5, 1.0), COMPLEX(0.0, 5.0)], $
[COMPLEX(-10.0, 0.0), COMPLEX(2.0, 1.0), COMPLEX(0.0, 0.0)], $
[COMPLEX(5.0, 1.0), COMPLEX(1.0, 0.0), COMPLEX(0.5, 3.0)])]
b = TRANSPOSE([$
[COMPLEX(0.5, 0.0), COMPLEX(0.0,0.0), COMPLEX(0.0, 0.0)], $
[COMPLEX(3.0,3.0), COMPLEX(3.0,3.0), COMPLEX(0.0, 1.0)], $
[COMPLEX(4.0, 2.0), COMPLEX(0.5, 1.0), COMPLEX(1.0, 1.0)])]
; Compute eigenvalues
IMSL_GENEIG, a, b, alpha, beta, Vectors = vectors
; Print eigenvalues
PM, alpha/beta, Title = 'Eigenvalues'
Eigenvalues
( -8.18018, -25.3799)
( 2.18006, 0.609112)
( 0.120109, -0.389223)
; Print eigenvectors
PM, vectors, Title = 'Eigenvectors'
Eigenvectors
( -0.326709, -0.124509)( -0.300678, -0.244401)
( 0.0370698, 0.151778)
( 0.176670, 0.00537758)( 0.895923, 0.000000)
( 0.957678, 0.000000)
( 0.920064, 0.000000)( -0.201900, 0.0801192)
( -0.221511, 0.0968290)
```

Version History

6.4	Introduced



Chapter 6

Interpolation and Approximation

This section contains the following topics:

Overview: Interpolation and Approximation . . .	Interpolation and Approximation Routines	199
192		

Overview: Interpolation and Approximation

Many functions in this chapter produce cubic piecewise polynomial or general spline functions that either interpolate or approximate given data or are support functions for the evaluation and integration of these functions. Three major subdivisions of functions are provided. The cubic spline functions begin with the prefix CS and use the piecewise polynomial representation. The spline functions begin with the prefix BS and use the B-spline representation. The third major subdivision includes functions that operate on the output of both the cubic spline and B-spline functions. Most spline functions are based on routines documented by de Boor (1978).

General purpose routines also are provided for general least-squares fit to data and routines to interpolate or approximate scattered data in R^n for $n \geq 1$.

Piecewise Polynomials

A univariate piecewise polynomial function, p , is specified by giving its breakpoint sequence $\xi \in R^n$, the order k (degree $k - 1$) of its polynomial pieces, and the $k \times (n - 1)$ matrix C of its local polynomial coefficients. In terms of this information, the piecewise polynomial (ppoly) function is given by the following equation:

$$p(x) = \sum_{j=1}^k c_{ij} \frac{(x - \xi_i)^{j-1}}{(j-1)!} \quad \text{for} \quad \xi_i \leq x \leq \xi_{i+1}$$

The breakpoint sequence ξ is assumed to be strictly increasing, and the ppoly function is extended to the entire real axis by extrapolation from the first and last intervals. This representation is redundant when the ppoly function is known to be smooth. For example, if p is known to be continuous, then $c_{1, i+1}$ can be computed from the c_{ji} as follows:

$$c_{1, i+1} = p(\xi_{i+1}) = \sum_{j=1}^k c_{ij} \frac{(\xi_{i+1} - \xi_i)^{j-1}}{(j-1)!}$$

For smooth ppoly, the nonredundant representation is used in terms of the “basis” or B-splines, at least when such a function is first to be determined.

Splines and B-splines

B-splines provide a particularly convenient and suitable basis for a given class of smooth ppoly functions. Such a class is specified by giving its breakpoint sequence, its order k , and the required smoothness across each of the interior breakpoints. The corresponding B-spline basis is specified by giving its knot sequence $\mathbf{t} \in \mathbb{R}^m$. The specification rule is the following: If the class is to have all derivatives up to and including the j -th derivative continuous across the interior breakpoint ξ_j , then the number ξ_j should occur $k - j - 1$ times in the knot sequence. Assuming that ξ_1 and ξ_n are the endpoints of the interval of interest, choose the first k knots equal to ξ_1 and the last k knots equal to ξ_n . This can be done since the B-splines are defined to be right continuous near ξ_1 and left continuous near ξ_n .

When the above construction is completed, a knot sequence $\mathbf{t}$ of length M is generated and $m = M - k$ B-splines of order k (for example, $B_0, \dots, B_{m-1}$) span the ppoly functions on the interval with the indicated smoothness. That is, each ppoly function in this class has a unique representation:

$$p = a_0 B_0 + a_1 B_1 + \dots + a_{m-1} B_{m-1}$$

as a linear combination of B-splines. A B-spline is a particularly compact ppoly function. The function B_i is a nonnegative function that is nonzero only on the interval $[\mathbf{t}_i, \mathbf{t}_{i+k}]$. More precisely, the support of the i -th B-spline is $[\mathbf{t}_i, \mathbf{t}_{i+k}]$. No ppoly function in the same class (other than the zero function) has smaller support (i.e., vanishes on more intervals) than a B-spline. This makes B-splines particularly attractive basis functions since the influence of any particular B-spline coefficient extends only over a few intervals. When it is necessary to emphasize the dependence of the B-spline on its parameters, the notation:

$$B_{i, k, \mathbf{t}}$$

is used to denote the i -th B-spline of order k for the knot sequence $\mathbf{t}$.

Cubic Splines

Cubic splines are smooth (i.e., C^1 or C^2), fourth-order ppoly functions. For historical and other reasons, cubic splines are the most frequently used ppoly functions. Therefore, special functions are provided for their construction and evaluation. These routines use the ppoly representation as described above for general ppoly functions (with $k = 4$).

Two cubic spline interpolation functions, [IMSL_CSINTERP](#) and [IMSL_CSSHAPE](#), are provided. The [IMSL_CSINTERP](#) function allows the user to specify various endpoint conditions (such as the value of the first or second derivative at the right and

left points). This means that the natural cubic spline can be obtained using this function by setting the second derivative to zero at both endpoints. The `IMSL_CSSHAPE` function is designed so that the shape of the curve matches the shape of the data. In particular, one option of this function preserves the convexity of the data while the default attempts to minimize oscillations.

It is possible that the cubic spline interpolation functions will produce unsatisfactory results. For example, the interpolant may not have the shape required by the user, or the data may be noisy and require a least-squares fit. The `IMSL_BSINTERP` interpolation function is more flexible, as it allows the user to choose the knots and order of the spline interpolant. The user is encouraged to use this routine and exploit the flexibility provided.

Tensor-product Splines

The simplest method of obtaining multivariate interpolation and approximation functions is to take univariate methods and form a multivariate method via tensor products. In the case of two-dimensional spline interpolation, the derivation proceeds as follows: Let $\mathbf{t}_x$ be a knot sequence for splines of order k_x and $\mathbf{t}_y$ be a knot sequence for splines of order k_y . Let $N_x + k_x$ be the length of $\mathbf{t}_x$ and $N_y + k_y$ be the length of $\mathbf{t}_y$. Then, the tensor-product spline has the following form:

$$\sum_{m=0}^{N_y-1} \sum_{n=0}^{N_x-1} c_{nm} B_{n, k_x, \mathbf{t}_x}(x) B_{m, k_y, \mathbf{t}_y}(y)$$

Given two sets of points:

$$\{x_i\}_{i=1}^{N_x} \quad \text{and} \quad \{y_j\}_{j=1}^{N_y}$$

for which the corresponding univariate interpolation problem can be solved, the tensor-product interpolation problem finds the coefficients c_{nm} , so that the following is true:

$$\sum_{m=0}^{N_y-1} \sum_{n=0}^{N_x-1} c_{nm} B_{n, k_x, \mathbf{t}_x}(x_i) B_{m, k_y, \mathbf{t}_y}(y_j) = f_{ij}$$

This problem can be solved efficiently by repeatedly solving univariate interpolation problems as described in de Boor (1978, p. 347). Three-dimensional interpolation can be handled in an analogous manner. This chapter provides functions that compute the

two-dimensional, tensor-product spline coefficients given two-dimensional interpolation data ([IMSL_BSINTERP](#)) and functions that compute the two-dimensional, tensor-product spline coefficients for a tensor-product, least-squares problem ([IMSL_BSLSQ](#)). In addition, evaluation, differentiation, and integration routines ([IMSL_SPVALUE](#) and [IMSL_SPINTEG](#)) are provided for the two-dimensional, tensor-product spline functions.

Scattered-data Interpolation and Approximation

IDL Advanced Math and Stats provides functions to interpolate and approximate scattered data in R^n for $n \geq 1$. The [IMSL_SCAT2DINTERP](#) function interpolates scattered data in the plane and is based on work by Akima (1978), which uses C^1 piecewise quintics on a triangular mesh. The [IMSL_RADBF](#) function can be used to either interpolate or approximate scattered data in R^n for $n \geq 1$. The [IMSL_RADBF](#) function computes approximations based on radial-basis functions. The fit computed by [IMSL_RADBF](#) can be evaluated using the [IMSL_RADBE](#) function.

Least Squares

IDL Advanced Math and Stats includes functions for smoothing noisy data. The [IMSL_FCNLSQ](#) function computes regressions with user-supplied functions. The [IMSL_BSLSQ](#) function computes a one- or two-dimensional, least-squares fit using splines with fixed knots or variable knots. This function produces cubic-spline, least-squares fit by default. Keywords allow the user to choose the order and the knot sequence.

IDL Advanced Math and Stats contains many functions that provide for polynomial regression and general linear regression.

Smoothing by Cubic Splines

One “smoothing spline” function is provided. The default action of [IMSL_CSSMOOTH](#) estimates a smoothing parameter by cross-validation, then returns the cubic spline that smooths the data. If the user chooses to supply a smoothing parameter, this function returns the appropriate cubic spline.

Structures for Splines and Piecewise Polynomials

This section is optional and is intended for users interested in more details concerning the structures for splines and piecewise polynomials.

A spline can be viewed as a mapping with domain R^d and target R^r , where d and r are positive integers. For this version of the IDL Advanced Math and Stats module, only $r = 1$ is supported. Thus, if s is a spline, then the following is true for some d and r :

$$s: R^d \rightarrow R^r$$

This implies that such a spline s must have d knot sequences and orders (one for each domain dimension). Thus, associated with s , knots and orders are as follows:

$$\mathbf{t}^0, \dots, \mathbf{t}^{d-1}$$

$$k_0, \dots, k_{d-1}$$

The precise form of the spline follows:

$$s(x) = (s_0(x), \dots, s_{r-1}(x)) \quad x = (x_1, \dots, x_d) \in R^d$$

where:

$$s_i(x) := \sum_{j_{d-1}=0}^{n_{d-1}-1} \dots \sum_{j_0=0}^{n_0-1} c_{j_0, \dots, j_{d-1}}^i B_{j_0, k_0, \mathbf{t}^0} \dots B_{j_{d-1}, k_{d-1}, \mathbf{t}^{d-1}}$$

Note that n_i is the number of knots in $\mathbf{t}^i$ minus the order k_i .

All the information for a spline is stored in a structure. Since the components of this structure are generally of varying lengths, an anonymous structure is defined for each spline. An example of the information returned by the HELP command with the keyword STRUCTURES set and an argument containing a spline structure follows:

```
x = FINDGEN(10)
y = IMSL_RANDOM(10)
spline = IMSL_BSINTERP(x, y)
HELP, spline, /Structure
** Structure $1, 7 tags, 116 length:
DOMAIN_DIM      LONG      1
TARGET_DIM      LONG      1
ORDER           LONG      4
NUM_COEF        LONG      10
NUM_KNOTS       LONG      14
KNOTS           FLOAT     Array(14)
COEF            FLOAT     Array(10)
```


For ppoly functions, a ppoly is viewed as a mapping with domain R^d and target R^r , where d and r are positive integers. Thus, if p is a ppoly, then the following is true for some d and r :

$$p: R^d \rightarrow R^r$$

For this version of the IDL Advanced Math and Stats module, only $r = 1$ is supported. This implies that such a ppoly p must have d breakpoint sequences and orders (one for each domain dimension). Thus, associated with p , breakpoints and orders are as follows:

$$\xi^1, \dots, \xi^d$$

$$k_1, \dots, k_d$$

The precise form of the ppoly follows:

$$p(x) = (p_0(x), \dots, p_r(x)) \quad x = (x_1, \dots, x_d) \in R^d$$

where:

$$p_i(x) := \sum_{l_d=0}^{k_d-1} \dots \sum_{l_1=0}^{k_1-1} c_{L^1, \dots, L^d, l_1, \dots, l_d}^i \frac{(x_1 - \xi_{L^1}^1)^{l_1}}{l_1!} \dots \frac{(x_d - \xi_{L^d}^d)^{l_d}}{l_d!}$$

with:

$$L^j := \max \{1, \min \{M^j, n_j - 1\}\}$$

where M^j is chosen so that:

$$\xi_{M^j}^j \leq x_j < \xi_{M^j+1}^j \quad j = 1, \dots, d$$

(with $\xi_0^j = -\infty$ and $\xi_{n_j+1}^j = \infty$).

Note that n_j is the number of breakpoints in ξ^j .

All the information for a spline is stored in a structure. Since the components of this structure are generally of varying lengths, an anonymous structure is defined for each spline. An example of the information returned by the `HELP` command with the keyword `STRUCTURES` set and an argument containing a spline structure is as follows:

```
x = FINDGEN(10)
y = IMSL_RANDOM(10)
ppoly = IMSL_CSINTERP(x, y)
HELP, ppoly, /Structure
Structure <103bc00>, 7 tags, length=204, data length=204, refs=1:
DOMAIN_DIM      LONG      1
TARGET_DIM      LONG      1
ORDER           LONG      Array[1]
NUM_COEF        LONG      Array[1]
NUM_BREAKPOINTS LONG      Array[1]
BREAKPOINTS     FLOAT     Array[10]
COEF            FLOAT     Array[36]
```

Interpolation and Approximation Routines

Cubic Spline Interpolation

[IMSL_CSINTERP](#)—Derivative end conditions.

[IMSL_CSSHAPE](#)—Shape preserving.

B-spline Interpolation

[IMSL_BSINTERP](#)—One-dimensional and two-dimensional interpolation.

[IMSL_BSKNOTS](#)—Knot sequence given interpolation data.

B-spline and Cubic Spline Evaluation and Integration

[IMSL_SPVALUE](#)—Evaluation and differentiation.

[IMSL_SPINTEG](#)—Integration.

Least-squares Approximation and Smoothing

[IMSL_FCNLSQ](#)—General functions.

[IMSL_BSLSQ](#)—Splines with fixed knots.

[IMSL_CONLSQ](#)—Constrained spline fit.

[IMSL_CSSMOOTH](#)—Cubic-smoothing spline.

[IMSL_SMOOTHDATA1D](#)—Smooth one-dimensional data by error detection.

Scattered Data Interpolation

[IMSL_SCAT2DINTERP](#)—Akima's surface-fitting method.

[IMSL_RADBF](#)—Computes a fit using radial-basis functions.

[IMSL_RADBE](#)—Evaluates a radial-basis fit.

IMSL_CSINTERP

The IMSL_CSINTERP function computes a cubic spline interpolant, specifying various endpoint conditions. The default interpolant satisfies the not-a-knot condition.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_CSINTERP(xdata, fdata [, /DOUBLE] [, /ILEFT=value]  
[, /IRIGHT=value] [, /LEFT=value] [, /PERIODIC] [, /RIGHT=value])
```

Return Value

A structure that represents the cubic spline interpolant.

Arguments

xdata

One-dimensional array containing the abscissas of the interpolation problem.

fdata

One-dimensional array containing the ordinates for the interpolation problem.

Keywords

DOUBLE

If present and nonzero, double precision is used.

ILEFT

Sets the value for the first or second derivative of the interpolant at the left endpoint. The keyword ILEFT is used to specify which derivative is set: ILEFT = 1 for the first

derivative and ILEFT = 2 for the second derivative. The only valid values for ILEFT are 1 or 2. If ILEFT is specified, then the keyword LEFT also must be used.

IRIGHT

Sets the value for the first or second derivative of the interpolant at the right endpoint. The keyword IRIGHT is used to specify which derivative is set: IRIGHT = 1 for the first derivative and IRIGHT = 2 for the second derivative. The only valid values for IRIGHT are 1 or 2. If IRIGHT is specified, then the keyword RIGHT also must be used.

LEFT

Sets the value for the first or second derivative of the interpolant at the left endpoint. Use with the keyword ILEFT. If ILEFT = i , then the interpolant s satisfies $s^{(i)}(x_L) = \text{LEFT}$. Here, x_L is the leftmost abscissa.

PERIODIC

If present and nonzero, computes the C^2 periodic interpolant to the data. The following is satisfied:

$$s^{(i)}(x_L) = s^{(i)}(x_R) \quad i = 0, 1, 2$$

where s , x_L , and x_R are defined above.

RIGHT

Sets the value for the first or second derivative of the interpolant at the right endpoint. Use with the keyword IRIGHT. If IRIGHT = i , then the interpolant s satisfies $s^{(i)}(x_R) = \text{RIGHT}$. Here, x_R is the rightmost abscissa.

Discussion

The IMSL_CSINTERP function computes a C^2 cubic spline interpolant to a set of data points (x_i, f_i) for the following:

$$i = 0, \dots, (\text{N\_ELEMENTS}(xdata) - 1) = (n - 1)$$

The breakpoints of the spline are the abscissas. For all univariate interpolation functions, the abscissas need not be sorted. Endpoint conditions are to be selected by the user. The user can specify not-a-knot, or first or second derivatives at each endpoint or C^2 periodicity can be requested (see de Boor 1978, Chapter 4). If no defaults are selected, then the not-a-knot spline interpolant is computed. If the PERIODIC keyword is selected, then all other keywords are ignored and a C^2 is computed. In this case, if the $fdata$ values at the left and right endpoints are not the

same, a warning message is issued and the right value is set equal to the left. If the LEFT and ILEFT or RIGHT and IRIGHT keywords are used, the user has the ability to select the values of the first or second derivative at either endpoint. The default case (when the keyword is not used) is the not-a-knot condition on that endpoint. Thus, when no keywords are chosen, this function produces the not-a-knot interpolant.

If the data (including the endpoint conditions) arise from the values of a smooth (for example, C^4) function f , i.e., $f_i = f(x_i)$, then the error behaves in a predictable fashion. Let ξ be the breakpoint vector for the above spline interpolant. Then, the maximum absolute error satisfies:

$$\|f - s\|_{[\xi_0, \xi_n]} \leq C \|f^{(4)}\|_{[\xi_0, \xi_n]} |\xi|^4$$

where the following is true:

$$|\xi| := \max_{i = 0, \dots, n-1} |\xi_{i+1} - \xi_i|$$

Examples

Example 1

In this example, a cubic spline interpolant, as shown in [Figure 6-1](#), to function values is computed and plotted along with the original data. Since the default settings are used, the interpolant is determined by the not-a-knot condition (see de Boor 1978).

```
x = FINDGEN(11)/10
; Generate the abscissas.
f = SIN(15 * x)
; Generate the function values.
pp = IMSL_CSINTERP(x, f)
; Compute the spline interpolant.
ppval = IMSL_SPVALUE(FINDGEN(100)/99, pp)
PLOT, FINDGEN(100)/99, ppval
; Plot the results.
OPLOT, x, f, Psym = 6
```

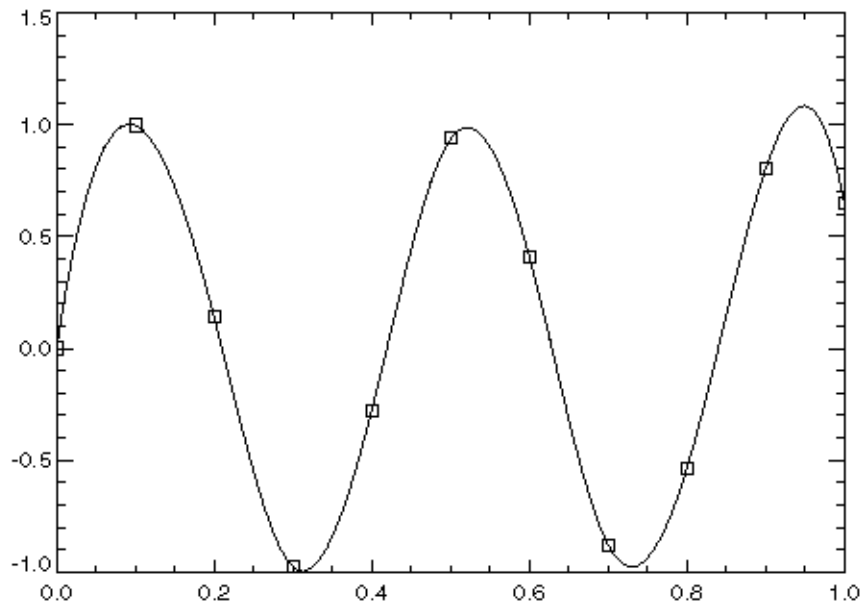


Figure 6-1: Cubic Spline Interpolant

Example 2

In this example, a cubic spline interpolant to function values is computed. The value of the derivative at the left endpoint and the value of the second derivative at the right endpoint are specified. The resulting spline and original data are then plotted as shown in [Figure 6-2](#).

```
x = FINDGEN(11)/10
y = SIN(15 * x)
pp = IMSL_CSINTERP(x, y, ILeft = 1, Left = 0, $
IRight = 2, Right = -225 * SIN(15))
ppval = IMSL_SPVALUE(FINDGEN(100)/99, pp)
PLOT, FINDGEN(100)/99, ppval
OPLLOT, x, y, Psym = 6
```

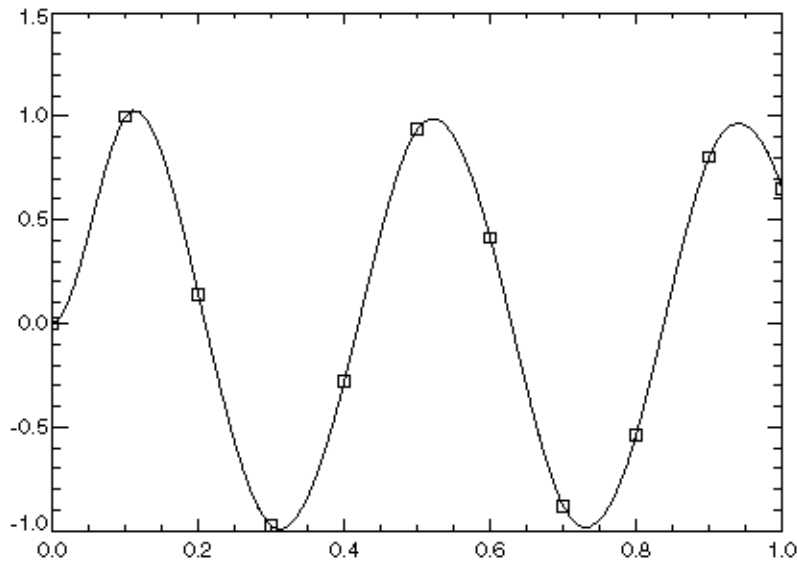


Figure 6-2: Cubic Spline Interpolant with Endpoint Conditions

Errors

Warning Errors

`MATH_NOT_PERIODIC`—Data are not periodic. The rightmost *fdata* value is set to the leftmost *fdata* value.

Fatal Errors

`MATH_DUPLICATE_XDATA_VALUES`—The *xdata* values must be distinct.

Version History

6.4	Introduced

IMSL_CSSHAPE

The IMSL_CSSHAPE function computes a shape-preserving cubic spline.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_CSSHAPE(xdata, fdata [, /CONCAVE] [, /DOUBLE]  
[, ITMAX=value])
```

Return Value

A structure that represents the cubic spline interpolant.

Arguments

xdata

One-dimensional array containing the abscissas of the interpolation problem.

fdata

One-dimensional array containing the ordinates for the interpolation problem.

Keywords

CONCAVE

If present and nonzero, IMSL_CSSHAPE produces a cubic interpolant that preserves the concavity of the data.

DOUBLE

If present and nonzero, double precision is used.

ITMAX

Allows the user to set the maximum number of iterations of Newton's Method. To use ITMAX, the keyword CONCAVE must also be set. Default: ITMAX = 25.

Discussion

The IMSL_CSSHAPE function computes a C^1 cubic spline interpolant to a set of data points (x_i, f_i) for the following:

$$i = 0, \dots, (N\_ELEMENTS(xdata) - 1) = (n - 1)$$

The breakpoints of the spline are the abscissas. This computation is based on a method by Akima (1970) to combat wiggles in the interpolant. Endpoint conditions are automatically determined by the program (see Akima 1970, de Boor 1978).

If the CONCAVE keyword is set, then this function computes a cubic spline interpolant to the data. For ease of explanation, $x_i < x_{i+1}$ is assumed, although it is not necessary for the user to sort these data values. If the data are strictly convex, then the computed spline is convex, C^2 , and minimizes the expression

$$\int_{x_1}^{x_n} (g'')^2$$

over all convex C^1 functions that interpolate the data. In the general case, when the data have both convex and concave regions, the convexity of the spline is consistent with the data, and the above integral is minimized under the appropriate constraints. For more information on this interpolation scheme, refer to Micchelli et al. (1985) and Irvine et al. (1986).

One important feature of the splines produced by this function is that it is not possible, *a priori*, to predict the number of breakpoints of the resulting interpolant. In most cases, there will be breakpoints at places other than data locations. This function should be used when it is important to preserve the convex and concave regions implied by the data.

Both methods are nonlinear, and although the interpolant is a piecewise cubic, cubic polynomials are not reproduced. (However, linear polynomials are reproduced.) This explains the theoretical error estimate below.

If the data points arise from the values of a smooth (for example, C^4) function f , i.e., $f_i = f(x_i)$, then the error behaves in a predictable fashion. Let ξ be the breakpoint vector for either of the above spline interpolants. Then, the maximum absolute error satisfies:

$$\|f - s\|_{[\xi_0, \xi_n]} \leq C \|f^{(2)}\|_{[\xi_0, \xi_n]} |\xi|^2$$

where:

$$|\xi| := \max_{i = 0, \dots, n-1} |\xi_{i+1} - \xi_i|$$

and ξ_m is the last breakpoint.

The returned value for this function is a structure. This structure contains all the information to determine the spline (stored as a piecewise polynomial) that is computed by this function. For example, the following code sequence evaluates this spline at x and returns the value in y :

```
y = IMSL_SPVALUE(x, spline)
```

Examples

Example 1

In this example, a cubic spline interpolant to function values is computed. Evaluations of the computed spline are plotted along with the original data values.

```
x = FINDGEN(10)/9
; Define the abscissas.
f = FLTARR(10)
f(0:4) = 0.25
f(5:9) = 0.75
; Define the function values.
pp = IMSL_CSSHAPE(x, f)
; Compute the interpolant.
ppval = IMSL_SPVALUE(FINDGEN(100)/99, pp)
; Evaluate the interpolant at 100 values in [0,1].
PLOT, FINDGEN(100)/99, ppval
; Plot the results.
OPLOT, x, f, Psym = 6
```

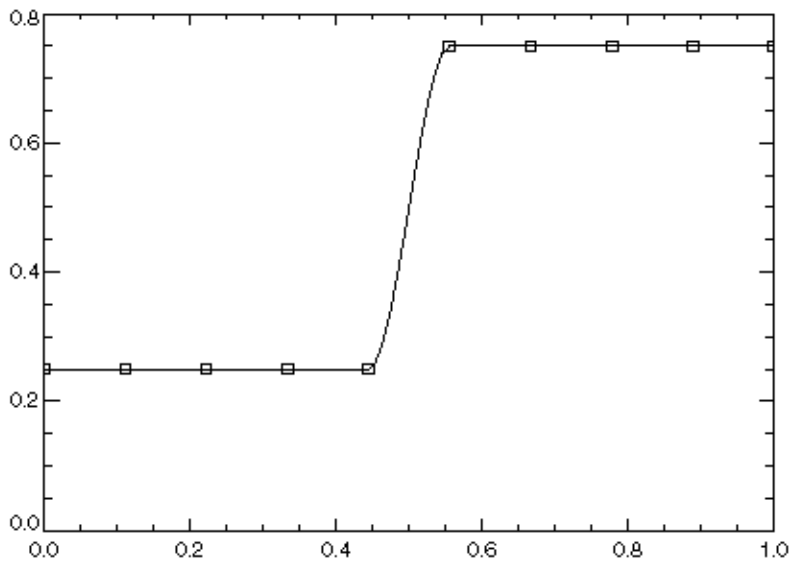


Figure 6-3: Shape-Preserving Cubic Spline

Example 2

This example compares interpolants computed by `IMSL_CSINTERP` and `IMSL_CSSHAPE` with the keyword `CONCAVE`, as shown in Figure 6-4.

```
x = [0, .1, .2, .3, .4, .5, .6, .8, 1]
y = [0, .9, .95, .9, .1, .05, .05, .2, 1]
; Define the data set and compute interpolant from IMSL_CSINTERP.
pp1 = IMSL_CSINTERP(x, y)
pp2 = IMSL_CSSHAPE(x, y, /Concave)
; Compute the interpolant from IMSL_CSSHAPE with keyword Concave.
x2 = FINDGEN(100)/99
PLOT, x2, IMSL_SPVALUE(x2, pp1), Linestyle = 2
OPLOT, x2, IMSL_SPVALUE(x2, pp2)
OPLOT, x, y, Psym = 6
XYOUTS, .4, .9, 'IMSL_CSINTERP', Charsize = 1.2
OPLOT, [.73, .85], [.925, .925], Linestyle = 2
XYOUTS, .4, .8, 'IMSL_CSSHAPE !cwith CONCAVE', Charsize = 1.2
OPLOT, [.73, .85], [.8, .8]
XYOUTS, .4, .6, 'Original data', Charsize = 1.2
OPLOT, [.73], [.622], Psym = 6
```

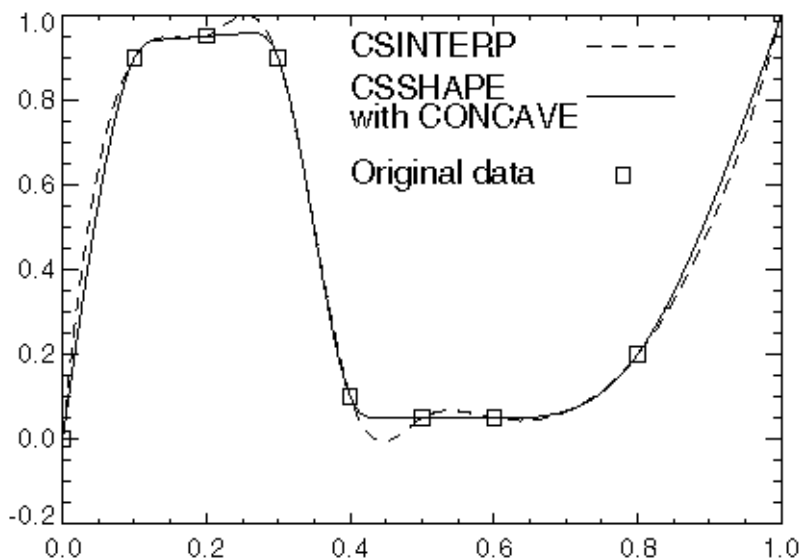


Figure 6-4: Cubic Spline Comparison

Errors

Warning Errors

`MATH_MAX_ITERATIONS_REACHED`—Maximum number of iterations has been reached. The best approximation is returned.

Fatal Errors

`MATH_DUPLICATE_XDATA_VALUES`—The *xdata* values must be distinct.

Version History

6.4	Introduced

IMSL_BSINTERP

The IMSL_BSINTERP function computes a one- or two-dimensional spline interpolant.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_BSINTERP(xdata, fdata [, /DOUBLE] [, XKNOTS=value]
                        [, XORDER=value] [, YKNOTS=value] [, YORDER=value])
```

or

```
Result = IMSL_BSINTERP(xdata, ydata, fdata [, DOUBLE=value]
                        [, XKNOTS=value] [, XORDER=value] [, YKNOTS=value]
                        [, YORDER=value])
```

Return Value

A structure containing information that defines the one- or two-dimensional spline.

Arguments

If a one-dimensional spline is desired, then the arguments *xdata* and *fdata* are required. If a two-dimensional, tensor-product spline is desired, then *xdata*, *ydata*, and *fdata* are required.

xdata

Array containing the abscissas in the *x*-direction of the interpolation problem.

ydata

Array containing the abscissas in the *y*-direction of the interpolation problem.

fdata

Array containing the ordinates of the interpolation problem. If a one-dimensional spline is being computed, then *fdata* (*i*) is the data value at *xdata* (*i*). If a two-

dimensional spline is being computed, then *fdata* is a two-dimensional array, where *fdata* (*i*, *j*) is the data value at (*xdata* (*i*), *ydata* (*i*)).

Keywords

DOUBLE

If present and nonzero, double precision is used.

XKNOTS

Specifies the array of knots in the *x*-direction to be used when computing the definition of the spline. Default: knots are selected by the [IMSL_BSKNOTS](#) function using its defaults.

XORDER

Specifies the order of the spline in the *x*-direction. Default: XORDER = 4, i.e., cubic splines.

YKNOTS

Specifies the array of knots in the *y*-direction to be used when computing the definition of the spline. Default: knots are selected by the [IMSL_BSKNOTS](#) function using its defaults.

YORDER

Specifies the order of the spline in the *y*-direction. If a one-dimensional spline is being computed, then YORDER has no effect on the computations. Default: YORDER = 4, i.e., cubic splines.

Discussion

The [IMSL_BSINTERP](#) function is designed to compute either a one-dimensional spline interpolant or two-dimensional, tensor-product spline interpolant to input data. The decision of whether to compute the one- or two-dimensional spline is based on the number of arguments passed to the function. Keywords are provided to allow the user to specify the order of the spline and the knots used for the spline. When computing a one-dimensional spline, the available keywords are XORDER and XKNOTS. When computing a two-dimensional spline, the order and knots in *x*-direction and/or *y*-direction can be specified using the keywords XORDER, XKNOTS, YORDER, and YKNOTS.

Separate discussions on one- and two-dimensional splines follow.

One-dimensional B-splines

Given the data points $x = xdata$, $f = fdata$, and the number of elements (n) in $xdata$ and $fdata$, the default action of IMSL_BSINTERP computes a cubic ($k = 4$) spline interpolant s to the data using the default knot sequence generated by the [IMSL_BSKNOTS](#) function.

Optional keyword XORDER allows the user to choose the order, k , of the spline interpolant; optional keyword XKNOTS allows user specification of knots.

The IMSL_BSINTERP function is based on the routine SPLINT by de Boor (1978, p. 204).

First, IMSL_BSINTERP sorts the $xdata$ vector and stores the result in x . The elements of the $fdata$ vector are permuted appropriately and stored in f , yielding the equivalent data (x_i, f_i) for $i = 0$ to $n - 1$.

The following preliminary checks are performed on the data:

$$\begin{aligned} x_i &< x_{i+1} & i = 0, \dots, n-2 \\ \mathbf{t}_i &< \mathbf{t}_{i+k} & i = 0, \dots, n-1 \\ \mathbf{t}_i &\leq \mathbf{t}_{i+1} & i = 0, \dots, n+k-2 \end{aligned}$$

The first test checks to see that the abscissas are distinct. The second and third inequalities verify that a valid knot sequence has been specified.

In order for the interpolation matrix to be nonsingular, $\mathbf{t}_{k-1} \leq x_i \leq \mathbf{t}_n$ is also checked for $i = 0$ to $n - 1$. This first inequality in the last check is necessary since the method used to generate the entries of the interpolation matrix requires that the k possibly nonzero B-splines at x_i :

$$B_{j-k+1}, \dots, B_j \quad \text{where } j \text{ satisfies } \mathbf{t}_j \leq x_i < \mathbf{t}_{j+1}$$

be well-defined (that is, $j - k + 1 \geq 0$).

General conditions are not known for the exact behavior of the error in spline interpolation; however, if $\mathbf{t}$ and x are selected properly and the data points arise from the values of a smooth (for example, C^k) function f , i.e., $f_i = f(x_i)$, then the error behaves in a predictable fashion. The maximum absolute error satisfies:

$$\|f - s\|_{[\mathbf{t}_{k-1}, \mathbf{t}_n]} \leq C \|f^{(k)}\|_{[\mathbf{t}_{k-1}, \mathbf{t}_n]} |\mathbf{t}|^k$$

where the following is true:

$$|t| := \max_{i = k-1, \dots, n-1} |t_{i+1} - t_i|$$

For more information on this, see de Boor (1978, Chapter 13) and the references therein. This function can be used in place of the [IMSL_CSINTERP](#) function.

The returned value for this function is a structure. This structure contains all the information to determine the spline (stored as a linear combination of B-splines) that is computed by this function. For example, the following code sequence evaluates this spline at x and returns the value in y :

```
y = IMSL_SPVALUE(x, spline)
```

Two-dimensional, Tensor-product B-splines

If arguments $xdata$, $ydata$, and $fdata$ are all included in the call to the [IMSL_BSINTERP](#) function, the function computes a two-dimensional, tensor-product spline interpolant. The tensor-product spline interpolant to data $\{(x_i, y_j, f_{ij})\}$, where $0 \leq i \leq n_x - 1$ and $0 \leq j \leq n_y - 1$, has the form:

$$\sum_{m=0}^{n_y-1} \sum_{n=0}^{n_x-1} c_{nm} B_{n, k_x, t_x}(x) B_{m, k_y, t_y}(y)$$

where k_x and k_y are the orders of the splines. These numbers are defaulted to 4 but can be set to any positive integer using keywords `XORDER` and `YORDER`. Likewise, t_x and t_y are the corresponding knot sequences (`XKNOTS` and `YKNOTS`). These values are defaulted to the knots returned by the [IMSL_BSKNOTS](#) function. The algorithm requires that the following is true:

$$\begin{aligned} t_x(k_x - 1) &\leq x_i \leq t_x(n_x) & 0 \leq i \leq n_x - 1 \\ t_y(k_y - 1) &\leq y_j \leq t_y(n_y) & 0 \leq j \leq n_y - 1 \end{aligned}$$

Tensor-product spline interpolants in two dimensions can be computed quite efficiently by solving (repeatedly) two univariate interpolation problems. The computation is motivated by the following observations:

$$\sum_{m=0}^{n_y-1} \sum_{n=0}^{n_x-1} c_{nm} B_{n, k_x, \mathbf{t}_x}(x_i) B_{m, k_y, \mathbf{t}_y}(y_j) = f_{ij}$$

Setting:

$$h_{mi} = \sum_{n=0}^{n_x-1} c_{nm} B_{n, k_x, \mathbf{t}_x}(x_i)$$

note that for each fixed i from 0 to $n_x - 1$, there are n_y linear equations in the same number of unknowns as can be seen below:

$$\sum_{m=0}^{n_y-1} h_{mi} B_{m, k_y, \mathbf{t}_y}(y_j) = f_{ij}$$

The same matrix appears in the previous equation.

$$[B_{m, k_y, \mathbf{t}_y}(y_j)] \quad 1 \leq m, j \leq n_y - 1$$

Thus, this matrix is factored only once, then the factorization to solve the n_x right-hand sides is applied. Once this is done and h_{mi} is computed, then the coefficients c_{nm} are solved using the relation:

$$\sum_{n=0}^{n_x-1} c_{nm} B_{n, k_x, \mathbf{t}_x}(x_i) = h_{mi}$$

for m from 0 to $n_y - 1$, which involves one factorization and n_y solutions to the different right-hand sides. This ability of the IMSL_BSINTERP function is based on the SPLI2D routine by de Boor (1978, p. 347).

The returned value is a structure containing all the information to determine the spline (stored in B-spline format) that is computed by this function. For example, the following code sequence evaluates this spline at (x, y) and returns the value in z :

```
z = IMSL_SPVALUE(x, y, spline)
```

Examples

Example 1

In this example, a one-dimensional B-spline interpolant to function values is computed, as shown in [Figure 6-5](#). Evaluations of the computed spline are then plotted along with the original data values. Since the default settings are being used, the interpolant is determined by the not-a-knot condition (see de Boor 1978).

```
x = FINDGEN(11)/10
; Define data values.
f = SIN(15 * x)
bs = IMSL_BSINTERP(x, f)
; Compute interpolant.
bsval = IMSL_SPVALUE(FINDGEN(100)/99, bs)
PLOT, FINDGEN(100)/99, bsval
; Output results.
OPLOT, x, f, Psym = 6
```

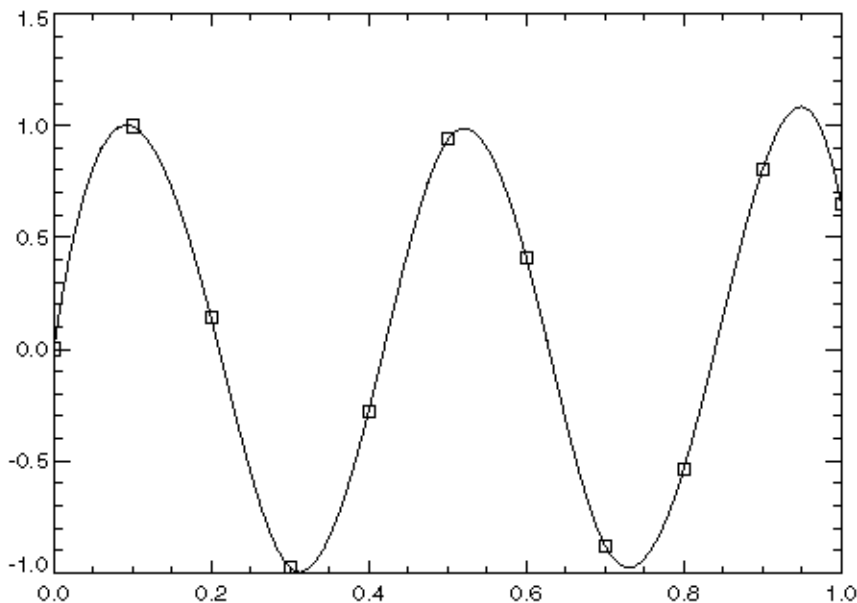


Figure 6-5: B-Spline Interpolant

Example 2

In this example, a two-dimensional, tensor-product B-spline interpolant to gridded data is computed as shown in [Figure 6-6](#).

```
x = FINDGEN(5)/4
; Define the abscissas in the x-direction.
y = FINDGEN(5)/4
; Define the abscissas in the y-direction.
f = FLTARR(5, 5)
; Define the sample function values.
FOR i = 0, 4 DO $
    f(i, *) = SIN(2 * x(i)) - COS(5 * y)
bs = IMSL_BSINTERP(x, y, f)
; Compute the spline interpolant.
bsval = IMSL_SPVALUE(FINDGEN(20)/19, FINDGEN(20)/19, bs)
; Use IMSL_SPVALUE to evaluate the computed spline.
!P.Charsize = 1.5
!P.Multi = [0, 1, 2]
WINDOW, XSize = 400, YSize = 800
; Plot the original and computed surfaces in a tall window.
```

```
SURFACE, f, x, y  
SURFACE, bsval, FINDGEN(20)/19, $  
FINDGEN(20)/19
```

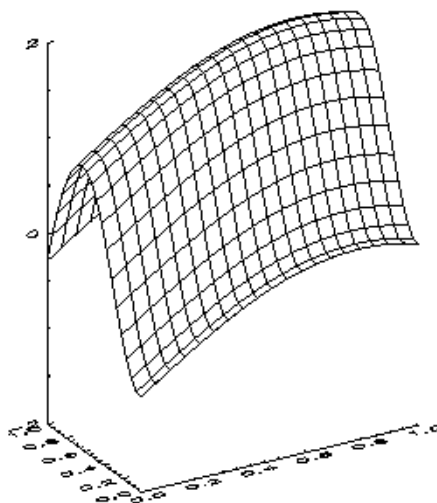
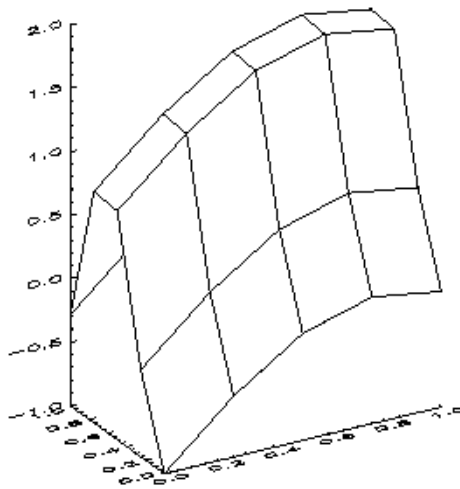


Figure 6-6: Two-Dimensional B-Spline

Errors

Warning Errors

MATH_ILL_COND_INTERP_PROB—Interpolation matrix is ill-conditioned. Solution might not be accurate.

Fatal Errors

MATH_DUPLICATE_XDATA_VALUES—The *xdata* values must be distinct.

MATH_YDATA_NOT_INCREASING—The *ydata* values must be strictly increasing.

MATH_KNOT_MULTIPLICITY—Multiplicity of the knots cannot exceed the order of the spline.

MATH_KNOT_NOT_INCREASING—Knots must be nondecreasing.

MATH_KNOT_XDATA_INTERLACING—The *i*-th smallest element of *xdata* (x_i) must satisfy $\mathbf{t}_i \leq x_i < \mathbf{t}_{i + \text{Order}}$, where $\mathbf{t}$ is the knot sequence.

MATH_XDATA_TOO_LARGE—Array *xdata* must satisfy $xdata_i \leq \mathbf{t}_{\text{ndata}}$, for $i = 1, \dots, \text{ndata}$.

MATH_XDATA_TOO_SMALL—Array *xdata* must satisfy $xdata_i \geq \mathbf{t}_{\text{Order} - 1}$, for $i = 1, \dots, \text{ndata}$.

MATH_KNOT_DATA_INTERLACING—The *i*-th smallest element of the data arrays *xdata* and *ydata* must satisfy $\mathbf{t}_i \leq data_i + \text{Order}$, where $\mathbf{t}$ is the knot sequence.

MATH_DATA_TOO_LARGE—Data arrays *xdata* and *ydata* must satisfy $data_i \leq \mathbf{t}_{\text{num\_data}}$, for $i = 1, \dots, \text{num\_data}$.

MATH_DATA_TOO_SMALL—Data arrays *xdata* and *ydata* must satisfy $data_i \geq \mathbf{t}_{\text{Order} - 1}$, for $i = 1, \dots, \text{num\_data}$.

Version History

6.4	Introduced

IMSL_BSKNOTS

The IMSL_BSKNOTS function computes the knots for a spline interpolant.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_BSKNOTS(xdata [, /DOUBLE] [, ITMAX=value]  
[, ORDER=value] [, /OPTIMUM])
```

Return Value

A one-dimensional array containing the computed knots.

Arguments

xdata

One-dimensional array containing the abscissas of the interpolation problem.

Keywords

DOUBLE

If present and nonzero, double precision is used.

ITMAX

Integer value used to set the maximum number of iterations of Newton's method. To use this keyword, the keyword OPTIMUM must also be set. Default: ITMAX = 10.

ORDER

Order of the spline subspace for which the knots are desired. Default: ORDER = 4, i.e., cubic splines.

OPTIMUM

If present and nonzero, knots that satisfy an optimal criterion are computed. See “Discussion” on page 220 for more information.

Discussion

Given the data points $x = xdata$, the order of the spline $k = \text{ORDER}$, and the number $n = \text{N\_ELEMENTS}(xdata)$ of elements in $xdata$, the default action of `IMSL_BSKNOTS` returns a knot sequence that is appropriate for interpolation of data on x by splines of order k (the default order is $k = 4$). The knot sequence is contained in its $n + k$ elements. If k is even and it is assumed that the entries in the input vector x are increasing, then the resulting knot sequence $\mathbf{t}$ is returned as follows:

$$\begin{aligned} \mathbf{t}_i &= x_0 \quad \text{for } i = 0, \dots, k-1 \\ \mathbf{t}_i &= x_{i-k/2-1} \quad \text{for } i = k, \dots, n-1 \quad (1) \\ \mathbf{t}_i &= x_{n-1} \quad \text{for } i = n, \dots, n+k-1 \end{aligned}$$

There is some discussion concerning this selection of knots in de Boor (1978, p. 211). If k is odd, then $\mathbf{t}$ is returned as follows:

$$\begin{aligned} \mathbf{t}_i &= x_0 & \text{for } i = 0, \dots, k-1 \\ \mathbf{t}_i &= \frac{1}{2} \left(x_{i-\frac{k-1}{2}-1} + x_{i-1-\frac{k-2}{2}} \right) & \text{for } i = k, \dots, n-1 \\ \mathbf{t}_i &= x_{n-1} & \text{for } i = n, \dots, n+k-1 \end{aligned}$$

It is not necessary to sort the values in $xdata$.

If keyword `OPTIMUM` is set, then the knot sequence returned minimizes the constant c in the error estimate:

$$\|f - s\| \leq c \|f^{(k)}\|$$

where f is any function in C^k and s is the spline interpolant to f at the abscissa x with knot sequence $\mathbf{t}$.

The algorithm is based on a routine described by de Boor (1978, p. 204), which in turn is based on a theorem of Micchelli et al. (1976).

Examples

Example 1

In this example, knots for a cubic spline are generated and printed. Notice that the knots are stacked at the endpoints; also, the second and next-to-last data points are not knots.

```
x = FINDGEN(6)
    knots = IMSL_BSKNOTS(x)
PM, knots, FORMAT = '(f5.2)'
    0.00
    0.00
    0.00
    0.00
    2.00
    3.00
    5.00
    5.00
    5.00
    5.00
```

Example 2

This example compares the default knots with the knots returned using keyword OPTIMIZE as shown in [Figure 6-7](#). The order is changed from the default value of 4 to 3.

```
x = FINDGEN(11)/10
; Define the abscissa values.
f = FLTARR(11)
; Define the function values.
f(0:3) = .25
f(4:7) = .5
f(8:10) = .25
sp1 = IMSL_BSINTERP(x, f)
; Compute the default spline.
knots2 = IMSL_BSKNOTS(x, /OPTIMUM, ORDER = 3)
; Compute the optimum knots of order 3.
sp2 = IMSL_BSINTERP(x, f, XKNOTS = knots2, XORDER = 3)
; Compute the spline of order 3, with the optimum knots.
x2 = FINDGEN(100)/99
; Evaluate the two splines for plotting.
sp1eval = IMSL_SPVALUE(x2, sp1)
sp2eval = IMSL_SPVALUE(x2, sp2)
PLOT, x2, sp1eval, Linestyle = 2
; Plot the results.
OPLOT, x2, sp2eval
```

```

OPLOT, x, f, PSYM = 6
XYOUTS, .25, .18, 'With optimum knots:', CHARSIZE = 1.5
OPLOT, [.65, .75], [.188, .188]
XYOUTS, .25, .135, 'With default knots:', CHARSIZE = 1.5
OPLOT, [.65, .75], [.143, .143], LINESSTYLE = 2
XYOUTS, .3, .09, 'Original data', CHARSIZE = 1.5
OPLOT, [.70], [.098], PSYM = 6

```

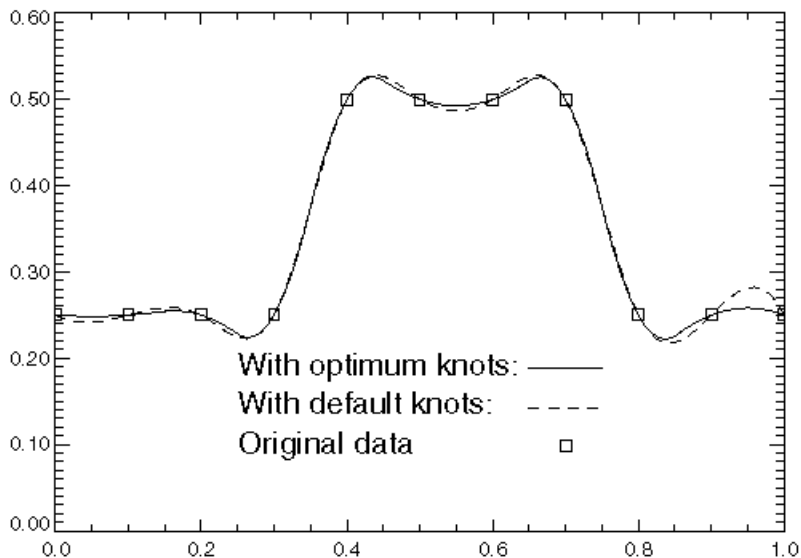


Figure 6-7: Optimum Knot Placement

Errors

Warning Errors

MATH_NO_CONV_NEWTON—Newton's method iteration did not converge.

Fatal Errors

MATH_DUPLICATE_XDATA_VALUES—The *xdata* values must be distinct.

MATH_ILL_COND_LIN_SYS—Interpolation matrix is singular. The *xdata* values may be too close together.

Version History

6.4	Introduced

IMSL_SPVALUE

The IMSL_SPVALUE function computes values of a spline or values of one of its derivatives.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_SPVALUE(*x*, *spline* [, *XDERIV*=value] [, *YDERIV*=value])

or

Result = IMSL_SPVALUE(*x*, *y*, *spline* [, *XDERIV*=value] [, *YDERIV*=value])

Return Value

The values of a spline or one of its derivatives.

Arguments

If evaluation of a one-dimensional spline is desired, then arguments *x* and *spline* are required. If evaluation of a two-dimensional spline is desired, then *x*, *y*, and *spline* are required.

x

Scalar value or an array of values at which the spline is to be evaluated in the *x*-direction. If *x* is an array, then *x* must be strictly increasing, i.e., $x(i) < x(i+1)$ for $i = 0, (N\_ELEMENTS(x) - 2)$.

y

Scalar value or an array of values at which the spline is to be evaluated in the *y*-direction. This argument should only be used if *spline* is a two-dimensional, tensor-product spline. If *y* is an array, then *y* must be strictly increasing, i.e., $y(i) < y(i+1)$ for $i = 0, (N\_ELEMENTS(y) - 2)$.

spline

Structure that represents the spline.

Keywords

XDERIV

Let $XDERIV = p$, and let s be the spline that is represented by *spline*. If s is a one-dimensional spline, this keyword produces the p -th derivative of s at x , $s^{(p)}(x)$. If s is a two-dimensional spline, this keyword specifies the order of the partial derivative in the x -direction. Let $q = YDERIV$, which has a default value of 0. Then, `IMSL_SPVALUE` produces the (p, q) -th derivative of s at (x, y) , $s^{(p, q)}(x, y)$. Default: $XDERIV = 0$

YDERIV

If $s = \textit{spline}$ is a two-dimensional spline, this keyword specifies the order of the partial derivative in the y -direction. Let $p = XDERIV$, which has a default value of zero, and $q = YDERIV$. Then, `IMSL_SPVALUE` produces the (p, q) -th derivative of s at (x, y) , $s^{(p, q)}(x, y)$. If *spline* is a one-dimensional spline, this keyword has no effect on computations. Default: $YDERIV = 0$

Discussion

The `IMSL_SPVALUE` function can be used to evaluate splines of the following type:

- Piecewise polynomials returned by `IMSL_CSINTERP`, `IMSL_CSSHAPE`, and `IMSL_CSSMOOTH`.
- One-dimensional B-splines returned by `IMSL_BSINTERP`, `IMSL_BSLSQ`, and `IMSL_CONLSQ`.
- Two-dimensional, tensor-product B-splines returned from `IMSL_BSINTERP` and `IMSL_BSLSQ`.

If *spline* is a piecewise polynomial, the `IMSL_SPVALUE` function computes the values of a cubic spline or one of its derivatives. In this case, supply the arguments x and *spline*, but do not supply the argument y . If x is a scalar, then a scalar is returned. If x is a one-dimensional array, then a one-dimensional array of values is returned. The first and last pieces of the cubic spline are extrapolated so that the cubic spline structures returned by the cubic spline routines are defined and can be evaluated on the entire real line. This ability is based on the routine `PPVALU` by de Boor (1978, p. 89).

If *spline* is a one-dimensional B-spline, the `IMSL_SPVALUE` function computes the values of a spline or one of its derivatives. In this case, the user is required to supply the arguments *x* and *spline* and must not supply the argument *y*. If *x* is a scalar, then a scalar is returned. If *x* is a one-dimensional array, then a one-dimensional array of values is returned. This ability is based on the routine `BVALUE` by de Boor (1978, p. 144).

If *spline* is a two-dimensional, tensor-product B-spline, the `IMSL_SPVALUE` function computes the values of a tensor-product spline or one of its derivatives. In this case, the user is required to supply the arguments *x*, *y*, and *spline*. If *x* and *y* are both scalars, then a scalar is returned. If *x* and *y* are both one-dimensional arrays, then a two-dimensional array of values is returned, where the (i, j) -th element of the returned matrix is the desired value of *spline* ($x(i), y(j)$). This ability is based on the discussion in de Boor (1978, pp. 351–353).

Examples

Example 1

This example computes a cubic spline interpolant to function values. The spline is then evaluated, and the results are plotted as shown in [Figure 6-8](#). Since the default settings are used, the interpolant is determined by the not-a-knot condition (see de Boor 1978).

```
x = FINDGEN(10)/9
f = SIN(15 * x)
pp = IMSL_CSINTERP(x, f)
x2 = FINDGEN(100)/99
ppeval = IMSL_SPVALUE(x2, pp)
PLOT, x2, ppeval
```

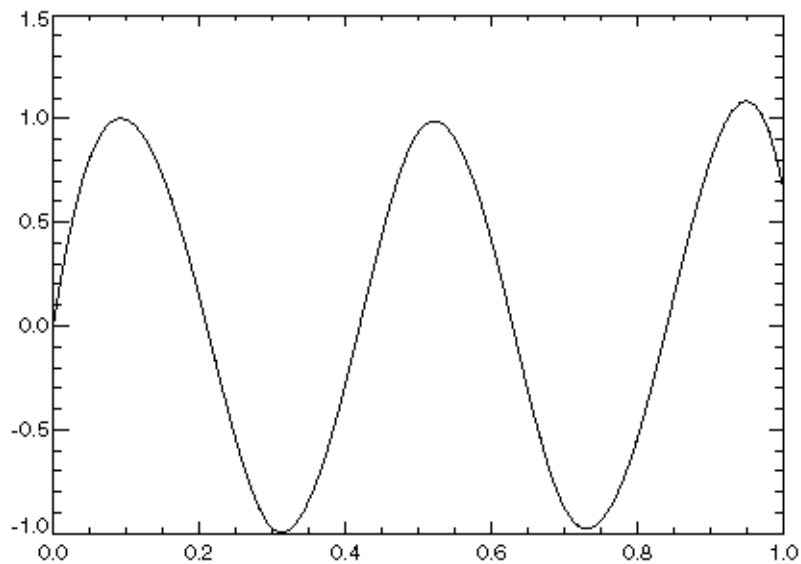


Figure 6-8: Spline Evaluation Plot

Example 2

This example computes a two-dimensional, tensor-product B-spline using `IMSL_BSINTERP`, then uses `IMSL_SPVALUE` to evaluate the spline on a grid, and plots the results as shown in Figure 6-9.

```
x = FINDGEN(5)/4
y = FINDGEN(5)/4
f = FLTARR(5, 5)
FOR i = 0, 4 DO f(i,*) = SIN(2 * !Pi * x(i)) * (-COS(!Pi*y/2))
; Generate the data.
bs = IMSL_BSINTERP(x, y, f)
; Compute the spline by calling IMSL_BSINTERP.
bsval = FLTARR(20, 20)
FOR i = 0, 19 DO BSVAL(i, *) = IMSL_SPVALUE(i/19., FINDGEN(20)/19,
bs)
; Evaluate the spline on a grid.
!P.Multi = [0, 1, 2]
WINDOW, XSize = 400, YSize = 800
; Plot the original data and the evaluations of the spline in the
; same plot window.
ax = 50
; The angle of rotation about x-axis in plots is defined by ax.
```

```

!P.Charsize = 1.5
SURFACE, f, x, y, Ax = ax, XTitle = 'X', YTitle = 'Y'
SURFACE, bsval, FINDGEN(20)/19, FINDGEN(20)/19, Ax = ax, $
    XTitle = 'X', YTitle = 'Y'

```

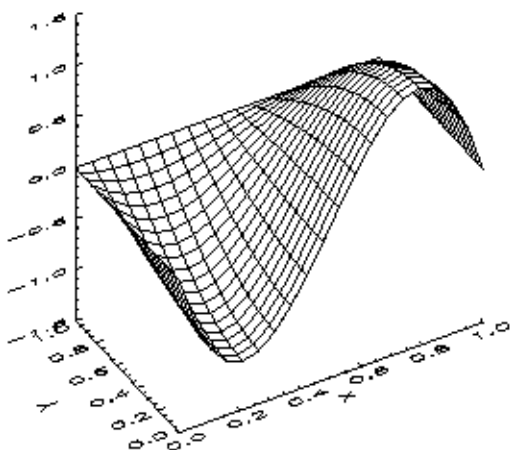
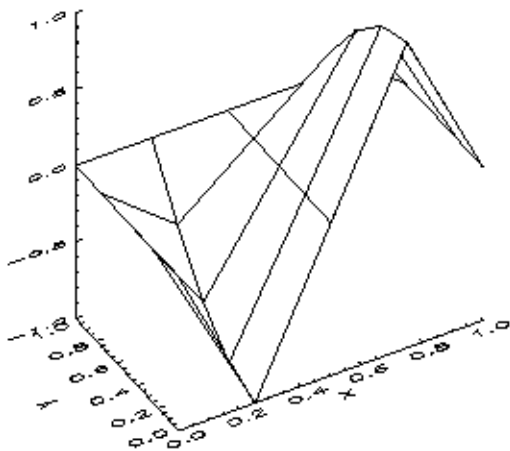


Figure 6-9: Two-Dimensional Spline Plot Evaluation

Errors

Warning Errors

`MATH_X_NOT_WITHIN_KNOTS`—Value of x does not lie within the knot sequence.

`MATH_Y_NOT_WITHIN_KNOTS`—Value of y does not lie within the knot sequence.

Fatal Errors

`MATH_KNOT_MULTIPLICITY`—Multiplicity of the knots cannot exceed the order of the spline.

`MATH_KNOT_NOT_INCREASING`—Knots must be nondecreasing.

Version History

6.4	Introduced

IMSL_SPINTEG

The IMSL_SPINTEG function computes the integral of a one- or two-dimensional spline.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_SPINTEG(*a*, *b*, *spline*)

Result = IMSL_SPINTEG(*a*, *b*, *c*, *d*, *spline*)

Return Value

If *spline* is a one-dimensional spline, then the returned value is the integral from *a* to *b* of *spline*. If *spline* is a two-dimensional, tensor-product spline, then the returned value is the value of the integral of *spline* over the rectangle $[a, b] \times [c, d]$. If no value can be computed, NaN (Not a Number) is returned.

Arguments

If integration of a one-dimensional spline is desired, then arguments *a*, *b*, and *spline* are required. If integration of a two-dimensional spline is desired, then *a*, *b*, *c*, *d*, and *spline* are required.

a

Right endpoint of integration.

b

Left endpoint of integration.

c

Right endpoint of integration for the second variable of the tensor-product spline. This argument should only be used if *spline* is a two-dimensional, tensor-product spline.

d

Left endpoint of integration for the second variable of the tensor-product spline. This argument should only be used if *spline* is a two-dimensional, tensor-product spline.

spline

Structure that represents the spline to be integrated.

Discussion

The IMSL_SPINTEG function can be used to integrate splines of the following type:

- Piecewise polynomials returned by [IMSL_CSINTERP](#), [IMSL_CSSHAPE](#), and [IMSL_CSSMOOTH](#).
- One-dimensional B-splines returned by [IMSL_BSINTERP](#), [IMSL_BSLSQ](#), and [IMSL_CONLSQ](#).
- Two-dimensional, tensor-product B-splines returned from [IMSL_BSINTERP](#) and [IMSL_BSLSQ](#).

If $s = \textit{spline}$ is a one-dimensional piecewise polynomial or B-spline, then IMSL_SPINTEG computes:

$$\int_a^b s(x) dx$$

If *spline* is a one-dimensional B-spline, then this function uses identity (22) of de Boor (1978, p. 115).

If $s = \textit{spline}$ is a two-dimensional, tensor-product spline, then the arguments c and d are required, and IMSL_SPINTEG computes:

$$\int_a^b \int_c^d s(x, y) dy dx$$

This function uses the (univariate integration) identity (22) of de Boor (1978, p. 151):

$$\int_{t_0}^x \sum_{i=0}^{n-1} \alpha_i B_{i,k}(\tau) d\tau = \sum_{i=0}^{r-1} \left[\sum_{j=0}^i \alpha_j \frac{t_{j+k} - t_j}{k} \right] B_{i,k+1}(x)$$

where $t_0 \leq x \leq t_r$. It assumes (for all knot sequences) that the first and last k knots are stacked; that is, $t_0 = \dots = t_{k-1}$ and $t_n = \dots = t_{n+k-1}$, where k is the order of the spline in the x or y direction.

Example

This example computes a cubic spline interpolant to function values. The values of the integral of this spline are then compared with the exact integral values. Since the default settings are being used, the interpolant is determined by the not-a-knot condition (de Boor 1978).

```

n = 21
; Generate the data.
x = FINDGEN(n)/(n - 1)
f = SIN(15 * x)
pp = IMSL_CSINTERP(x, f)
; Compute the interpolant.
results = FLTARR(22, 4)
; Define an array to hold some results to be output later.
FOR i = n/2, 3 * n/2 DO BEGIN & $
    x2 = i/FLOAT(2 * n - 2) & $
    y = IMSL_SPINTEG(0, x2, pp) & $
    results[i - n/2, *] = $
        [x2, (1 - COS(15 * x2))/15, y, $
        ABS((1 - COS(15 * x2))/15 - y)] & $
; Loop over different limits of integration and compare the
; results with the true answer.
ENDFOR
PM, results, FORMAT = '(4f12.4)', $
Title = '    X      True    Approx    Error'
; Output the results.

```

X	True	Approx	Error
0.2500	0.1214	0.1215	0.0001
0.2750	0.1036	0.1037	0.0001
0.3000	0.0807	0.0808	0.0001
0.3250	0.0559	0.0560	0.0001
0.3500	0.0325	0.0327	0.0001
0.3750	0.0139	0.0141	0.0002
0.4000	0.0027	0.0028	0.0002
0.4250	0.0003	0.0004	0.0002
0.4500	0.0071	0.0073	0.0002
0.4750	0.0223	0.0224	0.0001
0.5000	0.0436	0.0437	0.0001
0.5250	0.0681	0.0682	0.0001
0.5500	0.0924	0.0925	0.0001
0.5750	0.1131	0.1132	0.0001
0.6000	0.1274	0.1275	0.0001
0.6250	0.1333	0.1333	0.0001
0.6500	0.1298	0.1299	0.0001
0.6750	0.1176	0.1177	0.0001
0.7000	0.0984	0.0985	0.0001

0.7250	0.0747	0.0748	0.0001
0.7500	0.0499	0.0500	0.0001
0.7750	0.0274	0.0276	0.0001

Errors

Warning Errors

MATH_SPLINE_LEFT_ENDPT—Left endpoint of x integration is not within the knot sequence. Integration occurs only from $t_{Order-1}$ to b .

MATH_SPLINE_RIGHT_ENDPT—Right endpoint of x integration is not within the knot sequence. Integration occurs only from $t_{Order-1}$ to a .

MATH_SPLINE_LEFT_ENDPT_1—Left endpoint of x integration is not within the knot sequence. Integration occurs only from b to $t_{Spline\_Space\_Dim-1}$.

MATH_SPLINE_RIGHT_ENDPT_1—Right endpoint of x integration is not within the knot sequence. Integration occurs only from a to $t_{Spline\_Space\_Dim-1}$.

MATH_SPLINE_LEFT_ENDPT_2—Left endpoint of y integration is not within the knot sequence. Integration occurs only from $t_{Order-1}$ to d .

MATH_SPLINE_RIGHT_ENDPT_2—Right endpoint of y integration is not within the knot sequence. Integration occurs only from $t_{Order-1}$ to c .

MATH_SPLINE_LEFT_ENDPT_3—Left endpoint of y integration is not within the knot sequence. Integration occurs only from d to $t_{Spline\_Space\_Dim-1}$.

MATH_SPLINE_RIGHT_ENDPT_3—Right endpoint of y integration is not within the knot sequence. Integration occurs only from c to $t_{Spline\_Space\_Dim-1}$.

Fatal Errors

MATH_KNOT_MULTIPLICITY—Multiplicity of the knots cannot exceed the order of the spline.

MATH_KNOT_NOT_INCREASING—Knots must be nondecreasing.

Version History

6.4	Introduced

IMSL_FCNLSQ

The IMSL_FCNLSQ function computes a least-squares fit using user-supplied functions.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_FCNLSQ(f, nbasis, xdata, fdata [, /DOUBLE]  
[, INTERCEPT=variable] [, SSE=variable] [, WEIGHTS=value])
```

Return Value

A one-dimensional array containing the coefficients of the basis functions.

Arguments

f

Scalar string specifying the name of a user-supplied function that defines the subspace from which the least-squares fit is to be performed. The k -th basis function evaluated at x is $f(k, x)$, where $k = 1, 2, \dots, nbasis$.

nbasis

Number of basis functions.

xdata

One-dimensional array containing the abscissas of the least-squares problem.

fdata

One-dimensional array containing the ordinates of the least-squares problem.

Keywords

DOUBLE

If present and nonzero, double precision is used.

INTERCEPT

Named variable into which the coefficient of a constant function used to augment the user-supplied basis functions in the least-squares fit is stored. Setting this keyword forces an intercept to be added to the model.

SSE

Named variable into which the error sum of squares is stored.

WEIGHTS

Array of weights used in the least-squares fit.

Discussion

The IMSL_FCMLSQ function computes a best least-squares approximation to given univariate data of the form:

$$\{(x_i, f_i)\}_{i=0}^{n-1}$$

by M basis functions:

$$\{F_j\}_{j=1}^M$$

(where $M = nbasis$). In particular, the default for this function returns the coefficients a which minimize:

$$\sum_{i=0}^{n-1} w_i \left(f_i - \sum_{j=1}^M a_{j-1} F_j(x_i) \right)^2$$

where $w = \text{WEIGHTS}$, $n = \text{N\_ELEMENTS}(xdata)$, $x = xdata$, and $f = fdata$.

If the optional keyword `INTCERCEPT` is used, then an intercept is placed in the model and the coefficients a , returned by `IMSL_FCNLSQ`, minimize the error sum of squares as indicated below:

$$\sum_{i=0}^{n-1} w_i \left(f_i - \text{intercept} - \sum_{j=1}^M a_{j-1} F_j(x_i) \right)^2$$

Example

In this example, the following function is fit:

$$1 + \sin x + 7\sin 3x$$

This function is evaluated at 90 equally spaced points on the interval $[0, 6]$. Four basis functions, 1, $\sin x$, $\sin 2x$, and $\sin 3x$, are used.

```
.RUN
; Define the basis functions.
FUNCTION f, k, x
IF (k EQ 1) THEN RETURN, 1. $
    ELSE RETURN, SIN((k - 1) * x)
END

n = 90
xdata = 6 * FINDGEN(n) / (n - 1)
fdata = 1 + SIN(xdata) + 7 * SIN(3 * xdata)
nbasis = 4
; Generate the data.
coefs = IMSL_FCNLSQ('f', nbasis, xdata, fdata)
; Compute the coefficients summing IMSL_FCNLSQ.
PM, coefs, FORMAT = '(f10.5)'

; Print the results.
1.00000
1.00000
0.00000
7.00000
```


Errors

Warning Errors

`MATH_LINEAR_DEPENDENCE`—Linear dependence of the basis functions exists. One or more components of *coef* are set to zero.

`MATH_LINEAR_DEPENDENCE_CONST`—Linear dependence of the constant function and basis functions exists. One or more components of *coef* are set to zero.

Fatal Errors

`MATH_NEGATIVE_WEIGHTS_2`—All weights must be greater than or equal to zero.

Version History

6.4	Introduced

IMSL_BSLSQ

The IMSL_BSLSQ function computes a one- or two-dimensional, least-squares spline approximation.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_BSLSQ(xdata, fdata, xspacedim [, /DOUBLE] [, OPTIMIZE=value]
[, SSE=variable] [, XKNOTS=value] [, XORDER=value] [, XWEIGHTS=value]
[, YKNOTS=value] [, YORDER=value] [, YWEIGHTS=value])
```

or

```
Result = IMSL_BSLSQ(xdata, ydata, fdata, xspacedim, yspacedim
[, DOUBLE=value] [, OPTIMIZE=value] [, SSE=variable] [, XKNOTS=value]
[, XORDER=value] [, XWEIGHTS=value] [, YKNOTS=value]
[, YORDER=value] [, YWEIGHTS=value])
```

Return Value

A structure containing all the information to determine the spline fit.

Arguments

If a one-dimensional B-spline is desired, then arguments *xdata*, *fdata*, and *xspacedim* are required. If a two-dimensional, tensor-product B-spline is desired, then arguments *xdata*, *ydata*, *fdata*, *xspacedim*, and *yspacedim* are required.

xdata

One-dimensional array containing the data points in the x-direction.

ydata

One-dimensional array containing the data points in the y-direction.

fdata

Array containing the values to be approximated. If a one-dimensional approximation is to be computed, then *fdata* is a one-dimensional array. If a two-dimensional approximation is to be computed, then *fdata* is a two-dimensional array, where *fdata* (*i*, *j*) contains the value at (*xdata* (*i*), *ydata*(*j*)).

xspacedim

Linear dimension of the spline subspace for the *x* variable. It should be smaller than the number of data points in the *x*-direction and greater than or equal to the order of the spline in the *x*-direction (whose default value is 4).

yspacedim

Linear dimension of the spline subspace for the *y* variable. It should be smaller than the number of data points in the *y*-direction and greater than or equal to the order of the spline in the *y*-direction (whose default value is 4).

Keywords**DOUBLE**

If present and nonzero, double precision is used.

OPTIMIZE

If present and nonzero, optimizes the knot locations by attempting to minimize the least-squares error as a function of the knots. This keyword is only active if a one-dimensional spline is being computed.

SSE

Set this keyword equal to a named variable that will contain the weighted error sum of squares is stored.

XKNOTS

Specifies the array of knots in the *x*-direction to be used when computing the definition of the spline. Default: knots are equally spaced in the *x*-direction.

XORDER

Specifies the order of the spline in the *x*-direction. Default: *XORDER* = 4, i.e., cubic splines.

XWEIGHTS

Array containing the weights to use in the x -direction. Default: all weights equal to 1.

YKNOTS

Specifies the array of knots in the y -direction to be used when computing the definition of the spline. Default: knots are equally spaced in the y -direction.

YORDER

Specifies the order of the spline in the y -direction. If a one-dimensional spline is being computed, then YORDER has no effect on the computations. Default: YORDER = 4, i.e., cubic splines.

YWEIGHTS

Array containing the weights to use in the y -direction. If a one-dimensional spline is being computed, then YWEIGHTS has no effect on the computations. Default: all weights equal to 1.

Discussion

The IMSL_BSLSQ function computes a least-squares approximation to weighted data returning either a one-dimensional B-spline or a two-dimensional, tensor-product B-spline. The determination of whether to return a one- or two-dimensional spline is made based on the number of arguments passed to the function.

One-dimensional, B-spline Least-squares Approximation

Make the following identifications:

$$n = N\_ELEMENTS(xdata)$$

$$x = xdata$$

$$f = fdata$$

$$m = xspacedim$$

$$k = XOrder$$

For convenience, assume that the sequence x is increasing (although the function does not require this).

By default, $k = 4$ and the knot sequence selected equally distributes the knots through the distinct x_i 's. In particular, the $m + k$ knots are generated in $[x_0, x_{n-1}]$ with k

knots stacked at each of the extreme values. The interior knots are equally spaced in the interval.

Once knots $\mathbf{t}$ and weights w are determined (and assuming that keyword OPTIMIZE is not set), then the function computes the spline least-squares fit to the data by minimizing over the linear coefficients a_j , such that:

$$\sum_{i=0}^{n-1} w_i \left(f_i - \sum_{j=0}^{m-1} a_j B_j(x_i) \right)^2$$

where $B_j, j = 0, \dots, m-1$, is a (B-spline) basis for the spline subspace.

The XORDER keyword allows the user to choose the order of the spline fit. The XKNOTS keyword allows user specification of knots. The one-dimensional functionality of IMSL_BSLSQ is based on the routine L2APPR by de Boor (1978, p. 255).

If the keyword OPTIMIZE is used, the function attempts to find the best placement of knots that minimizes the least-squares error to the given data by a spline of order k with m coefficients. For this problem, it is necessary that $m > k$. Then, to find the minimum of the functional, use the following:

$$F(\mathbf{a}, \mathbf{t}) = \sum_{i=0}^{n-1} w_i \left(f_i - \sum_{j=0}^{m-1} a_j B_{j,k}(\mathbf{t}; x_i) \right)^2$$

The technique employed here uses the fact that for a fixed-knot sequence $\mathbf{t}$ the minimization in $\mathbf{a}$ is a linear least-squares problem that is easily solved. Thus, objective function F is a function of only $\mathbf{t}$ by setting the following:

$$G(\mathbf{t}) = \min F(\mathbf{a}, \mathbf{t})$$

A Gauss-Seidel (cyclic coordinate) method is then used to reduce the value of the new objective function G . In addition to this local method, there is a global heuristic built into the algorithm that is useful if the data arise from a smooth function. This heuristic is based on the routine NEWNOT of de Boor (1978, pp. 184, 258–261).

The guess, $\mathbf{t}^g$, for the knot sequence is either provided by the user or is the default. This must be a *valid* knot sequence for splines of order k with:

$$- \infty \leq \dots \leq \mathbf{t}_{k-1}^g \leq x_i \leq \mathbf{t}_m^g \leq \dots \leq \mathbf{t}_{m+k-1}^g \quad i = 1, \dots,$$

with $\mathbf{t}^g$ nondecreasing and $\mathbf{t}_i^g < \mathbf{t}_{i+k}^g$ for $i = 0, \dots, m-1$.

In regard to execution speed, this function can be several orders of magnitude slower than a simple least-squares fit.

The return value for this function is a structure containing all the information to determine the spline (stored in B-spline form) that is computed by this function.

In [Figure 6-10](#), two cubic splines are fit to $\text{SQRT}(|x|)$. Both splines are cubics with the same $x\text{spacedim} = 8$. The first spline is computed with the default settings, while the second spline is computed by optimizing the knot locations using the OPTIMIZE keyword.

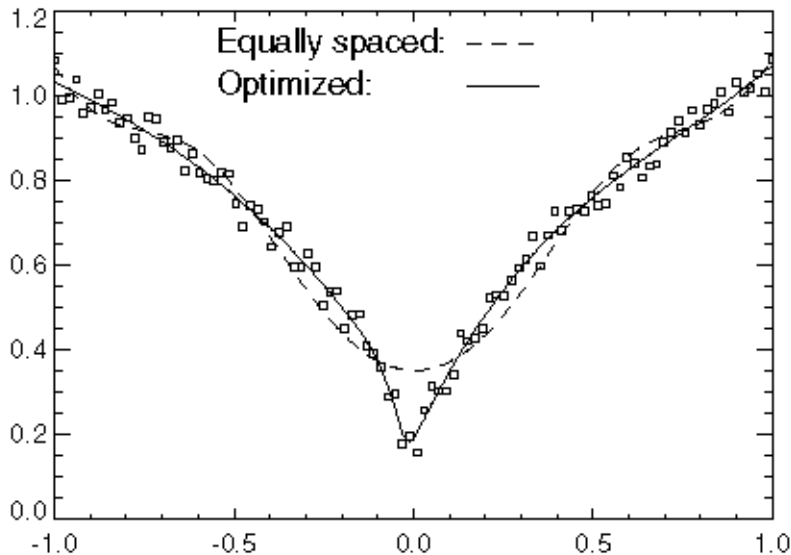


Figure 6-10: Two Fits to Noisy $\text{SQRT}(|x|)$

Two-dimensional, B-spline Least-squares Approximation

If a two-dimensional, tensor-product B-spline is desired, the `IMSL_BSLSQ` function computes a tensor-product spline, least-squares approximation to weighted, tensor-product data. The input for this function consists of data vectors to specify the tensor-product grid for the data, two vectors with the weights (optional, the default is 1), the values of the surface on the grid, and the specification for the tensor-product spline (optional, a default is chosen). The grid is specified by the two vectors $x = xdata$ and $y = ydata$ of length $n = \text{N\_ELEMENTS}(xdata)$ and $m = \text{N\_ELEMENTS}(ydata)$,

respectively. A two-dimensional array $f = fdata$ contains the data values to be fit. The two vectors $w_x = XWEIGHTS$ and $w_y = YWEIGHTS$ contain the weights for the weighted, least-squares problem. The information for the approximating tensor-product spline can be provided using keywords `XORDER`, `YORDER`, `XKNOTS`, and `YKNOTS`. This information is contained in $k_x = XORDER$, $t_x = XKNOTS$, and $n = xspacedim$ for the spline in the first variable, and in $k_y = YORDER$, $t_y = YKNOTS$, and $m = yspacedim$ for the spline in the second variable.

This function computes coefficients for the tensor-product spline by solving the normal equations in tensor-product form as discussed in de Boor (1978, Chapter 17). For more information, see the paper by Grosse (1980).

As the computation proceeds, coefficients c are obtained minimizing:

$$\left(\sum_{i=0}^{n-1} \sum_{j=0}^{m-1} w_x(i) w_y(j) \left[\sum_{k=0}^{N-1} \sum_{l=0}^{M-1} c_{kl} B_{kl}(x_i, y_j) - f_{ij} \right]^2 \right)$$

where the function B_{kl} is the tensor-product of two B-splines of order k_x and k_y :

$$B_{kl}(x, y) = B_{k, k_x, t_x}(x) B_{l, k_y, t_y}(y)$$

The spline:

$$\sum_{k=0}^{N-1} \sum_{l=0}^{M-1} c_{kl} B_{kl}$$

and its partial derivatives can be evaluated using [IMSL_SPVALUE](#).

The return value for this function is a structure containing all the information to determine the spline that is computed by this function. For example, the following code sequence evaluates this spline (stored in the structure *sp*) at (x, y) and returns the value in *v*:

```
v = IMSL_SPVALUE(x, y, sp)
```

Examples

Example 1

This example fits data generated from a trigonometric polynomial:

$$1 + \sin x + 7 \sin 3x + \varepsilon$$

where ϵ is a random uniform deviate over the range $[-1, 1]$. The data is obtained by evaluating this function at 90 equally spaced points on the interval $[0, 6]$. This data is fit with a cubic spline with 12 degrees of freedom (eight equally spaced interior knots). The computed fit and original data are then plotted, as shown in [Figure 6-11](#), as follows:

```
n = 90
x = 6 * FINDGEN(n)/(n - 1)
f = 1 + SIN(x) + 7 * SIN(3 * x) + (1 - 2 * IMSL_RANDOM(n))
; Set up the data.
sp = IMSL_BSLSQ(x, f, 8)
; Compute the spline fit.
speval = IMSL_SPVALUE(x, sp)
; Evaluate the computed spline at the original data abscissa.
PLOT, x, speval
; Plot the results.
OPLOT, x, f, Psym = 6
```

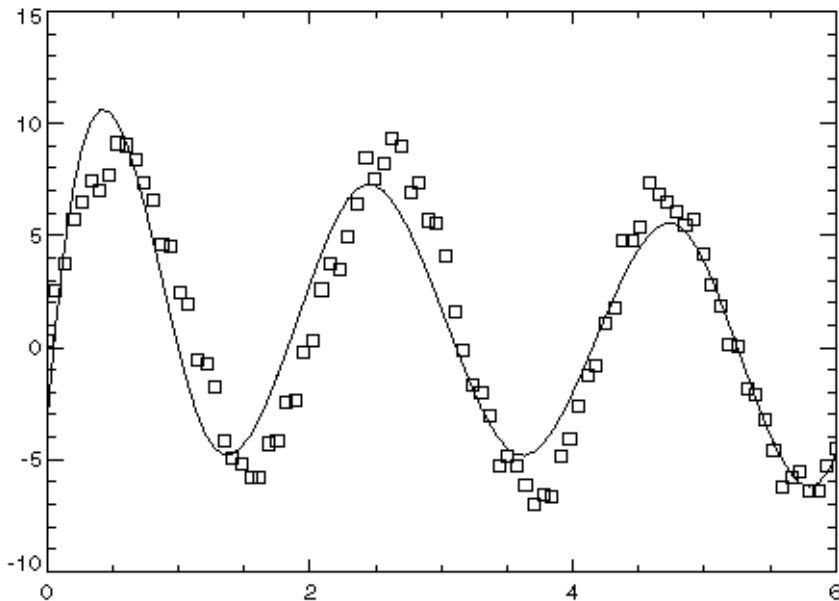


Figure 6-11: One-Dimensional Least-Squares B-Spline Fit

Example 2

This example fits noisy data that arises from the function $e^x \sin(x + y) + \varepsilon$, where ε is a random uniform deviate in the range $(-1, 1)$, on the rectangle $[0, 3] \times [0, 5]$. This function is sampled on a 50×25 grid and the original data and the evaluations of the computed spline are plotted as shown in [Figure 6-12](#).

```

nx = 50
ny = 25
; Generate noisy data on a 50 x 25 grid.
x = 3 * FINDGEN(nx)/(nx - 1)
y = 5 * FINDGEN(ny)/(ny - 1)
f = FLTARR(nx, ny)
FOR i = 0, nx - 1 DO f(i, *) = EXP(x(i)) * $
    SIN(x(i) + y) + 2 * IMSL_RANDOM(ny) - 1
sp = IMSL_BSLSQ(x, y, f, 5, 7)
; Call IMSL_BSLSQ to compute the least-squares fit. Notice that
; xspacedim = 5 and yspacedim = 7.
speval = IMSL_SPVALUE(x, y, sp)
; Evaluate the fit on the original grid.
!P.Multi = [0, 1, 2]
WINDOW, XSize = 500, YSize = 800
; Plot the original data and the fit in the same window.
SURFACE, f, x, y, Ax = 45, XTitle = 'X', YTitle = 'Y'
SURFACE, speval, x, y, Ax = 45, XTitle = 'X', YTitle = 'Y'

```

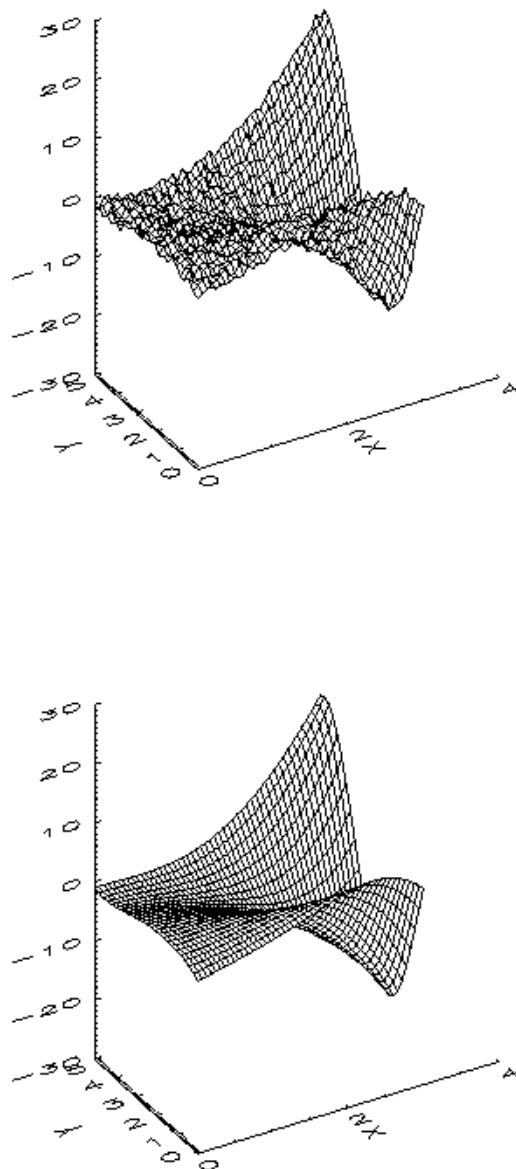


Figure 6-12: Two-Dimensional B-Spline Fit to Noisy Data

Errors

Warning Errors

MATH_ILL_COND_LSQ_PROB—Least-squares matrix is ill-conditioned. Solution might not be accurate.

MATH_SPLINE_LOW_ACCURACY—There may be less than one digit of accuracy in the least-squares fit. Try using higher precision if possible.

MATH_OPT_KNOTS_STACKED_1—Knots found to be optimal are stacked more than *Order*. This indicates that fewer knots will produce the same error sum of squares. Knots have been separated slightly.

Fatal Errors

MATH_KNOT_MULTIPLICITY—Multiplicity of the knots cannot exceed the order of the spline.

MATH_KNOT_NOT_INCREASING—Knots must be nondecreasing.

MATH_SPLINE_LRGST_ELEMNT—Data arrays *xdata* and *ydata* must satisfy $data_i \leq t_{\text{Spline\_Space\_Dim}}$, for $i = 1, \dots, num\_data$.

MATH_SPLINE_SMLST_ELEMNT—Data arrays *xdata* and *ydata* must satisfy $data_i \geq t_{\text{Order} - 1}$, for $i = 1, \dots, num\_data$.

MATH_NEGATIVE_WEIGHTS—All weights must be greater than or equal to zero.

MATH_DATA DECREASING—The *xdata* values must be nondecreasing.

MATH_XDATA_TOO_LARGE—Array *xdata* must satisfy $xdata_i \leq t_{\text{ndata}}$, for $i = 1, \dots, ndata$.

MATH_XDATA_TOO_SMALL—Array *xdata* must satisfy $xdata_i \geq t_{\text{Order} - 1}$, for $i = 1, \dots, ndata$.

MATH_OPT_KNOTS_STACKED_2—Knots found to be optimal are stacked more than *Order*. This indicates fewer knots will produce the same error sum of squares.

Version History

6.4	Introduced

IMSL_CONLSQ

The IMSL_CONLSQ function computes a least-squares constrained spline approximation.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_CONLSQ(xdata, fdata, spacedim, constraints[, nhard] [, /DOUBLE]  
[, /KNOTS=value] [, /ORDER=value] [, /WEIGHTS=value])
```

Return Value

A structure that represents the spline fit.

Arguments

xdata

One-dimensional array containing the abscissas of the least-squares problem.

fdata

One-dimensional array containing the ordinates of the least-squares problem.

spacedim

Linear dimension of the spline subspace. It should be smaller than the number of data points and greater than or equal to the order of the spline (whose default value is 4).

constraints

Array of structures containing the abscissas at which the fit is to be constrained, the derivative of the spline that is to be constrained, the type of constraints, and any lower or upper limits. A description of the structure fields follows.

- XVAL—Point at which fit is constrained (float).
- DER—Derivative value of the spline to be constrained (long int).

- TYPE—Types of the general constraints (long int).
- BL—Lower limit of the general constraints (float).
- BU—Upper limit of the general constraints (float).

Note

To constrain the integral of the spline over the closed interval (c, d) , set *constraints* $(i).$ XVAL = c and *constraints* $(i + 1).$ XVAL = d . For consistency, insist that *constraints* $(i).$ TYPE = *constraints* $(i + 1).$ TYPE = 5, 6, 7, or 8 and $c \leq d$.

constraints(i).TYPE-th constraint

$$1 \quad bl_i = f^{(d_i)}(x_i)$$

$$2 \quad f^{(d_i)}(x_i) \leq bu_i$$

$$3 \quad f^{(d_i)}(x_i) \geq bl_i$$

$$4 \quad bl_i \leq f^{(d_i)}(x_i) \leq bu_i$$

$$5 \quad bl_i = \int_c^d f(t) dt$$

$$6 \quad \int_c^d f(t) dt \leq bu_i$$

$$7 \quad \int_c^d f(t) dt \geq bl_i$$

$$8 \quad bl_i \leq \int_c^d f(t) dt \leq bu_i$$

20 periodic end conditions

99 disregard this constraint

In order to have two-point constraints, *constraints* $(i).$ TYPE = *constraints* $(i + 1).$ TYPE is needed.

constraints(i).TYPE i-th constraint

$$9 \quad bl_i = f^{(d_i)}(x_i) - f^{(d_{i+1})}(x_{i+1})$$

$$10 \quad f^{(d_i)}(x_i) - f^{(d_{i+1})}(x_{i+1}) \leq bu_i$$

$$11 \quad f^{(d_i)}(x_i) - f^{(d_{i+1})}(x_{i+1}) \geq bl_i$$

$$12 \quad bl_i \leq f^{(d_i)}(x_i) - f^{(d_{i+1})}(x_{i+1}) \leq bu_i$$

nhard

(Optional) Number of entries of *constraints* involved in the “hard” constraints. Note that $0 \leq nhard \leq (\text{SIZE}(\text{constraints})) (1)$. The default, *nhard* = 0, always results in a fit, while setting *nhard* = $(\text{SIZE}(\text{constraints})) (1)$ forces all constraints to be met.

The “hard” constraints must be met or the function signals fail. The “soft” constraints

need not be satisfied, but there is an attempt to satisfy the “soft” constraints. The constraints must be listed in terms of priority with the most important constraints first. Thus, all “hard” constraints must precede “soft” constraints. If infeasibility is detected among the “soft” constraints, the function satisfies, in order, as many of the “soft” constraints as possible. Default: $n_{hard} = 0$

Keywords

DOUBLE

If present and nonzero, double precision is used.

KNOTS

Specifies the array of knots to be used when computing the spline. Default: knots are equally spaced.

ORDER

Specifies the order of the spline. Default: ORDER = 4, i.e., cubic splines.

WEIGHTS

Array containing the weights to be used. Default: all weights equal 1.

Discussion

The IMSL_CONLSQ function produces a constrained, weighted, least-squares fit to data from a spline subspace. Constraints involving one-point, two-points, or integrals over an interval are allowed.

The types of constraints supported by the functions are of four types:

$$\begin{aligned} E_p[f] &= f^{(j_p)}(y_p) \\ \text{or} &= f^{(j_p)}(y_p) - f^{(j_{p+1})}(y_{p+1}) \\ \text{or} &= \int_{y_p}^{y_{p+1}} f(t) dt \\ \text{or} &= \text{periodic end conditions} \end{aligned}$$

An interval, I_p , (which may be a point, a finite interval, or a semi-infinite interval) is associated with each of these constraints.

The input for this function consists of the data set (x_i, f_i) for $i = 1, \dots, N$ (where $N = N\_ELEMENTS(xdata)$); that is, the data which is to be fit and the dimension of the spline space from which a fit is to be computed, *spacedim*. The *constraints* argument is an array of structures that contains the abscissas of the points involved in specifying the constraints, as well as information relating the type of constraints and the constraint interval. The optional argument *nhard* allows users of this code to specify which constraints must be met and which constraints can be removed in order to compute a fit. The algorithm tries to satisfy all the constraints, but if the constraints are inconsistent, then it drops constraints in the reverse order specified, until either a consistent set of constraints is found or the “hard” constraints are determined to be inconsistent (the “hard” constraints are those involving *constraints*(0), ..., *constraints*(*nhard* – 1)).

Let n_f denote the number of feasible constraints as described above. The function solved the problem:

$$\sum_{i=1}^n \left| f_i - \sum_{j=1}^m a_j B_j(x_i) \right|^2 w_i$$

subject to:

$$E_p \left[\sum_{j=1}^m a_j B_j \right] \in I_p \quad p = 1, \dots, n_f$$

This linearly constrained least-squares problem is treated as a quadratic program and is solved by invoking [IMSL_QUADPROG](#).

The choice of weights depends on the data uncertainty in the problem. In some cases, there is a natural choice for the weights based on the estimates of errors in the data points.

Determining feasibility of linear constraints is a numerically sensitive task. If difficulties are encountered, a quick fix is to widen the constraint intervals I_p .

Example

This example is a simple application of `IMSL_CONLSQ`. Data from the function $x/2 + \sin(x/2)$ contaminated with random noise is generated and then fit with cubic splines. The function is increasing so it is hoped that the least-squares fit also is

increasing. This is not the case for the unconstrained least-squares fit generated by the `IMSL_BSLSQ` function. The derivative is then forced to be greater than zero at 15 equally spaced points and `IMSL_CONLSQ` is called. The resulting curve is monotone as shown in [Figure 6-13](#).

```
IMSL_RANDOMOPT, Set = 234579
; Set the random seed.
ndata = 15;
spacedim = 8;
; Generate the data to be fit.
x = 10 * FINDGEN(ndata)/(ndata - 1)
y = .5 * (x) + SIN(.5 * (x)) + IMSL_RANDOM(ndata) - .5
sp1 = IMSL_BSLSQ(x, y, spacedim)
; Compute the unconstrained least-squares fit.
nconstraints = 15
; Define the constraints to be used by IMSL_CONLSQ.
constraints = REPLICATE({constraint, $
    XVAL:0.0, DER:0L, TYPE:0L, BL:0.0, BU:0.0}, nconstraints)
; Define an array of constraint structures. Each element of the
; array contains one structure that defines a constraint.
constraints.XVAL = 10*FINDGEN(nconstraints)/(nconstraints-1)
; Put a constant at 15 equally spaced points.
FOR i = 0, nconstraints - 1 DO BEGIN &$
    constraints(i).DER = 1 &$
    constraints(i).TYPE = 3 &$
    constraints(i).BL = 0. &$
ENDFOR
; Define constraints to force the second derivative to be greater
; than zero at the 15 equally spaced points.
sp2 = IMSL_CONLSQ(x, y, spacedim, constraints)
; Call IMSL_CONLSQ.
nplot = 100
xplot = 10 * FINDGEN(nplot)/(nplot - 1)
yplot1 = IMSL_SPVALUE(xplot, sp1)
yplot2 = IMSL_SPVALUE(xplot, sp2)
PLOT, xplot, yplot1, Linestyle = 2
; Plot the results.
OPLOT, xplot, yplot2
OPLOT, x, y, Psym = 6
XYOUTS, 1, 4.5, 'IMSL_CONLSQ', Charsize = 2
XYOUTS, 1, 4, 'IMSL_BSLSQ', Charsize = 2
OPLOT, [5.0, 6.0], [4.6, 4.6]
OPLOT, [5.0, 6.0], [4.1, 4.1], Linestyle = 2
```

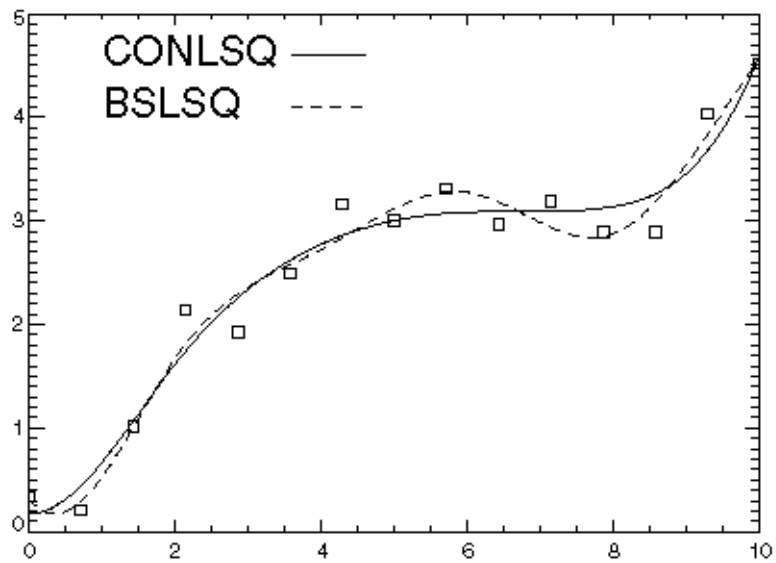



Figure 6-13: Monotonic B-Spline Fit to Noisy Data

Version History

6.4	Introduced

IMSL_CSSMOOTH

The CSSMOTH function computes a smooth cubic spline approximation to noisy data by using cross-validation to estimate the smoothing parameter or by directly choosing the smoothing parameter.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_CSSMOOTH(xdata, fdata [, /DOUBLE] [, SMPAR=value]  
[, WEIGHTS=value])
```

Return Value

The structure that represents the cubic spline.

Arguments

xdata

One-dimensional array containing the abscissas of the problem.

fdata

One-dimensional array containing the ordinates of the problem.

Keywords

DOUBLE

If present and nonzero, double precision is used.

SMPAR

Specifies the real, scalar smoothing parameter explicitly. See “[Discussion](#)” on page 255 for more details.

WEIGHTS

Array containing the weights to be used in the problem. Default: all weights are equal to 1

Discussion

The IMSL_CSSMOOTH function is designed to produce a C^2 cubic spline approximation to a data set in which the function values are noisy. This spline is called a *smoothing spline*.

Consider first the situation when the optional keyword SMPAR is selected. Then, a natural cubic spline with knots at all the data abscissas $x = xdata$ is computed, but it does *not* interpolate the data (x_i, f_i) . The smoothing spline s is the unique C^2 function which minimizes:

$$\int_a^b s''(x)^2 dx$$

subject to the constraint:

$$\sum_{i=0}^{n-1} |(s(x_i) - f_i)w_i|^2 \leq \sigma$$

where $w = \text{WEIGHTS}$, $\sigma = \text{SMPAR}$ is the smoothing parameter, and $n = \text{N\_ELEMENTS}(xdata)$.

Recommended values for σ depend on the weights w . If an estimate for the standard deviation of the error in the value f_i is available, then w_i should be set to the inverse of this value. The smoothing parameter σ should be chosen in the confidence interval corresponding to the left side of the above inequality; that is:

$$n - \sqrt{2n} \leq \sigma \leq n + \sqrt{2n}$$

The IMSL_CSSMOOTH function is based on an algorithm of Reinsch (1967). This algorithm also is discussed in de Boor (1978, pp. 235–243).

The default for this function chooses the smoothing parameter σ by a statistical technique called cross-validation. For more information on this topic, refer to Craven and Wahba (1979).

The return value for this function is a structure containing all the information to determine the spline (stored as a piecewise polynomial) that is computed by this procedure.

Example

In this example, function values are contaminated by adding a small “random” amount to the correct values. The `IMSL_CSSMOOTH` function is used to approximate the original, uncontaminated data as shown in [Figure 6-14](#).

```
n = 25
x = 6 * FINDGEN(n) / (n - 1)
f = SIN(x) + .5 * (IMSL_RANDOM(n) - .5)
; Generate the data.
pp = IMSL_CSSMOOTH(x, f)
; Compute the fit.
x2 = 6 * FINDGEN(100) / 99
; Evaluate the computed fit at 100 values in [0, 6].
ppeval = IMSL_SPVALUE(x2, pp)
PLOT, x2, ppeval
; Plot the results.
OPLOT, x, f, Psym = 6, Symsize = .5
```

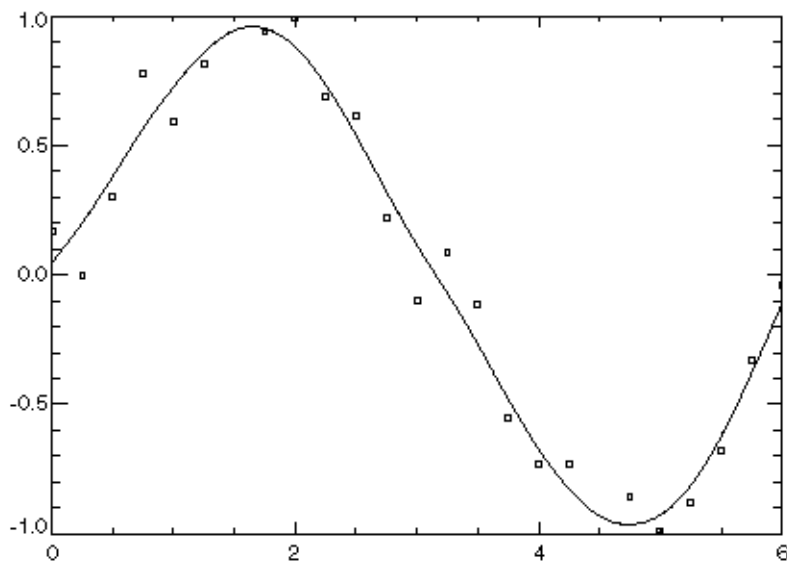


Figure 6-14: Smoothing Spline

Errors

Warning Errors

`MATH_MAX_ITERATIONS_REACHED`—Maximum number of iterations has been reached. The best approximation is returned.

Fatal Errors

`MATH_DUPLICATE_XDATA_VALUES`—The *xdata* values must be distinct.

`MATH_NEGATIVE_WEIGHTS`—All weights must be greater than or equal to zero.

Version History

6.4	Introduced

IMSL_SMOOTHDATA1D

The IMSL_SMOOTHDATA1D function smooths one-dimensional data by error detection.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_SMOOTHDATA1D(x, y [, DISTANCE=value] [, /DOUBLE]  
[, ITMAX=value] [, SC=value])
```

Return Value

One-dimensional array containing the smoothed data.

Arguments

x

One-dimensional array containing the abscissas of the data points.

y

One-dimensional array containing the ordinates of the data points.

Keywords

DISTANCE

Proportion of the distance the ordinate in error is moved to its interpolating curve. It must be in the range 0.0 to 1.0. Default: DISTANCE = 1.0

DOUBLE

If present and nonzero, double precision is used.

ITMAX

The maximum number of iterations allowed. Default: ITMAX = 500

SC

The stopping criterion. SC should be greater than or equal to zero. Default: SC = 0.0

Discussion

The IMSL_SMOOTHDATA1D function is designed to smooth a data set that is mildly contaminated with isolated errors. In general, the routine will not work well if more than 25% of the data points are in error. The routine IMSL_SMOOTHDATA1D is based on an algorithm of Guerra and Tapia (1974).

Setting $N\_ELEMENTS(x) = n$, $Y = f$, $Result = s$ and $X = x$, the algorithm proceeds as follows. Although the user need not an ordered x sequence, we will assume that x is increasing for simplicity. The algorithm first sorts the x values into an increasing sequence and then continues. A cubic spline interpolant is computed for each of the 6-point data sets (initially setting $s = f$):

$$(x_j, s_j) \quad j = i - 3, \dots, i + 3 \quad j \neq i$$

where $i = 4, \dots, n - 3$. For each i the interpolant, which we will call S_i , is compared with the current value of s_i , and a 'point energy' is computed as:

$$pe_i = S_i(x_i) - s_i$$

Setting $sc = SC$, the algorithm terminates either if ITMAX iterations have taken place or if:

$$|pe_i| \leq sc \frac{(x_{i+3} - x_{i-3})}{6} \quad i = 4, \dots, n - 3$$

If the above inequality is violated for any i , then we update the i -th element of s by setting $s_i = s_i + d(pe_i)$, where $d = \text{DISTANCE}$. Note that neither the first three nor the last three data points are changed. Thus, if these points are inaccurate, care must be taken to interpret the results.

The choice of the parameters DISTANCE, SC and ITMAX are crucial to the successful usage of this subroutine. If the user has specific information about the extent of the contamination, then he should choose the parameters as follows: DISTANCE = 1, SC = 0 and ITMAX to be the number of data points in error. On the other hand, if no such specific information is available, then choose DISTANCE = 0.5, ITMAX $\leq 2n$, and:

$$Sc = 0.5 \frac{\text{maxs} - \text{mins}}{(x_n - x_1)}$$

In any case, we would encourage the user to experiment with these values.

Example

We take 91 uniform samples from the function $5 + (5 + t^2 \sin t)/t$ on the interval $[1, 10]$. First, define function F from which samples will be taken

```
FUNCTION F, xdata
    RETURN, (xdata*xdata*SIN(xdata) + 5)/xdata + 5
END
```

Next, we contaminate 10 of the samples and try to recover the original function values.

```
isub = [5, 16, 25, 33, 41, 48, 55, 61, 74, 82]
rnoise = [2.5, -3.0, -2.0, 2.5, 3.0, -2.0, -2.5, 2.0, -2.0, 3.0]

; Example 1: No specific information available.
dis = 0.5
sc = 0.56
itmax = 182
; Set values for xdata and fdata.
xdata = 1 + 0.1*FINDGEN(91)
fdata = f(xdata)

; Contaminate the data.
fdata(isub) = fdata(isub) + rnoise

; Smooth the data.
sdata = IMSL_SMOOTHDATA1D(xdata, fdata, Itmax = itmax, $
    Distance = dis, Sc = sc)

; Output the results.
PM, [[f(xdata(isub))], [fdata(isub)], [sdata(isub)]], $
    Title = '          F(X)      F(X) + noise      sdata'
          F(X)      F(X) + noise      sdata
9.82958      12.3296      9.87030
8.26338      5.26338      8.21537
5.20083      3.20083      5.16823
2.22328      4.72328      2.26399
1.25874      4.25874      1.30825
3.16738      1.16738      3.13830
7.16751      4.66751      7.13076
10.8799      12.8799      10.9092
12.7739      10.7739      12.7075
7.59407      10.5941      7.63885

; Example 2: Specific information available.
```



```

dis = 1.0
sc = 0.0
itmax = 10.0
; A warning message is produce because the maximum number
; of iterations is reached.
sdata = IMSL_SMOOTHDATA1D(xdata, fdata, Itmax = itmax, $
    Distance = dis, Sc = sc)
% IMSL_SMOOTHDATA1D: Warning: MATH_ITMAX_EXCEEDED
; Maximum number of iterations limit 'ITMAX' = 10 exceeded. The
; best answer found is returned. Output the results.
PM, [[f(xdata(isub))], [fdata(isub)], [sdata(isub)]], $
    Title = '          F(X)    F(X) + noise    sdata'
    F(X)    F(X) + noise    sdata
9.82958    12.3296    9.83127
8.26338    5.26338    8.26223
5.20083    3.20083    5.19946
2.22328    4.72328    2.22495
1.25874    4.25874    1.26142
3.16738    1.16738    3.16958
7.16751    4.66751    7.16986
10.8799    12.8799    10.8779
12.7739    10.7739    12.7699
7.59407    10.5941    7.59194

```

Version History

6.4	Introduced

IMSL_SCAT2DINTERP

The IMSL_SCAT2DINTERP function computes a smooth bivariate interpolant to scattered data that is locally a quintic polynomial in two variables.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_SCAT2DINTERP(*xydata*, *fdata*, *xout*, *yout* [, /DOUBLE])

Return Value

A two-dimensional array containing the grid of values of the interpolant.

Arguments

xydata

Two-dimensional array containing the data points for the interpolation problem.

Argument *xydata* is dimensioned (2, N_ELEMENTS(*fdata*)). The *i*-th data point (x_i , y_i) is stored in *xydata* (0, *i*) = x_i and *xydata* (1, *i*) = y_i .

fdata

One-dimensional array containing the values to be interpolated.

xout

One-dimensional array specifying the *x* values for the output grid. It must be strictly increasing.

yout

One-dimensional array specifying the *y* values for the output grid. It must be strictly increasing.

Keywords

DOUBLE

If present and nonzero, double precision is used.

Discussion

The IMSL_SCAT2DINTERP function computes a C^1 interpolant to scattered data in the plane. Given the data points (in R^3):

$$\{(x_i, y_i, f_i)\}_{i=0}^{n-1}$$

where $n = N\_ELEMENTS(xydata) / 2$, IMSL_SCAT2DINTERP returns the values of the interpolant s on the user-specified grid. The computation of s is as follows.

First, the Delaunay triangulation of the points:

$$\{(x_i, y_i)\}_{i=0}^{n-1}$$

is computed. On each triangle T in this triangulation, s has the following form:

$$s(x, y) = \sum_{m+n \leq 5} c_{mn}^T x^m y^n \quad \forall (x, y) \in T$$

Thus, s is a bivariate quintic polynomial on each triangle of the triangulation. In addition:

$$s(x_i, y_i) = f_i \quad \text{for } i = 0, \dots, n-1$$

and s is continuously differentiable across the boundaries of neighboring triangles. These conditions do not exhaust the freedom implied by the above representation. This additional freedom is exploited in an attempt to produce an interpolant that is faithful to the global shape properties implied by the data. For more information on this procedure, refer to the article by Akima (1978). The output grid is specified by the two real vectors, *xout* and *yout*, that represent the first (second) coordinates of the grid.

Example

In this example, IMSL_SCAT2DINTERP is used to fit a surface to randomly scattered data. The resulting surface and the original data points are then plotted as shown in [Figure 6-15](#).

```

IMSL_RANDOMOPT, Set = 12345
ndata = 15
xydata = FLTARR(2, ndata)
xydata(*) = IMSL_RANDOM(2 * ndata)
fdata = IMSL_RANDOM(ndata)
x = xydata(0, *)
y = xydata(1, *)
ngrid = 20
xout = FINDGEN(ngrid)/(ngrid - 1)
yout = FINDGEN(ngrid)/(ngrid - 1)
; Define the grid used to evaluate the computed surface.
surf = IMSL_SCAT2DINTERP(xydata, fdata, xout, yout)
; Call IMSL_SCAT2DINTERP.
SURFACE, surf, xout, yout, /Save, Ax = 45, Charsize = 1.5
; Plot the computed surface.
PLOTS, x, y, fdata, /T3d, Symsize = 2, Psym = 2
; Plot the original data points.

```

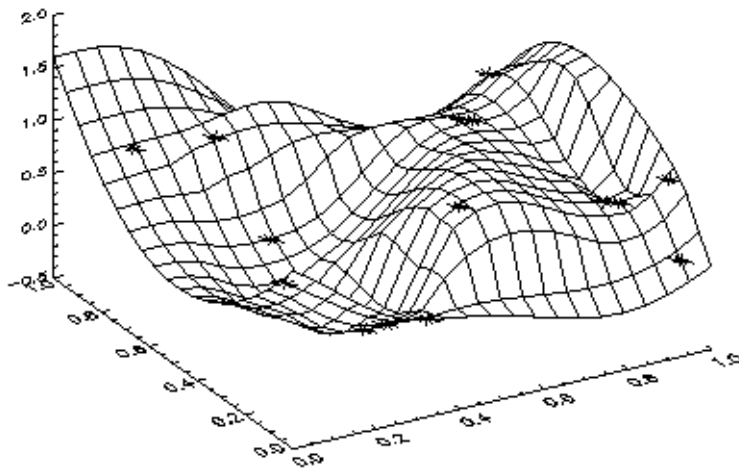


Figure 6-15: Fit to Scattered Data

Errors

Fatal Errors

MATH_DUPLICATE_XYDATA_VALUES—Two-dimensional data values must be distinct.

MATH_XOUT_NOT_STRICTLY_INCRSING—Vector *xout* must be strictly increasing.

MATH_YOUT_NOT_STRICTLY_INCRSING—Vector *yout* must be strictly increasing.

Version History

6.4	Introduced

IMSL_RADBF

The IMSL_RADBF function computes an approximation to scattered data in R^n for $n \geq 2$ using radial-basis functions.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_RADBF(abscissa, fdata, num_centers [, BASIS=string]  
[, CENTERS=value] [, DELTA=value] [, /DOUBLE]  
[, RANDOM_SEED=value] [, RATIO_CENTERS=value] [, WEIGHTS=value])
```

Return Value

A structure that represents the radial-basis fit.

Arguments

abscissa

Two-dimensional array containing the abscissas of the data points. Parameter *abscissa* (i, j) is the abscissa value of the j -th data point in the i -th dimension.

fdata

One-dimensional array containing the ordinates for the problem.

num_centers

Number of centers to be used when computing the radial-basis fit. The *num_centers* argument should be less than or equal to N_ELEMENTS (*fdata*).

Keywords

BASIS

Character string specifying a user-supplied function to compute the values of the radial functions. The form of the input function is $\phi(r)$. Default: the Hardy multiquadratic

CENTERS

User-supplied centers. See “[Discussion](#)” below for details.

DELTA

Delta used in the default basis function, $\phi(r) = \text{SQRT}(r^2 + \delta^2)$. Default: DELTA = 1.

DOUBLE

If present and nonzero, double precision is used.

RANDOM_SEED

Value of the random seed used when determining the random subset of abscissa to use as centers. By changing the value of seed on different calls to IMSL_RADBF, with the same data set, a different set of random centers are chosen. Setting RANDOM_SEED to zero forces the random number seed to be based on the system clock, so possibly, a different set of centers is chosen each time the program is executed. Default: *RANDOM_SEED* = 234579.

RATIO_CENTERS

Desired ratio of centers placed on an evenly spaced grid to the total number of centers. There is a condition: The same number of centers placed on a grid for each dimension must be equal. Thus, the actual number of centers placed on a grid is usually less than *RATIO_CENTERS* * *num_centers*, but is never more than *RATIO_CENTERS* * *num_centers*. The remaining centers are randomly chosen from the set of abscissa given in *abscissa*. Default: *RATIO_CENTERS* = 0.5

WEIGHTS

Requires the user to provide the weights. Default: all weights equal 1.

Discussion

The IMSL_RADBF function computes a least-squares fit to scattered data in R^d . More precisely, let $n = N\_ELEMENTS(fdata)$, $x = abscissa$, $f = fdata$, and $d = N\_ELEMENTS(abscissa(0, *))$. Then:

$$x^0, \dots, x^{n-1} \in \mathbb{R}^d$$

$$f_0, \dots, f_{n-1} \in \mathbb{R}^1$$

This function computes a function F which approximates the above data in the sense that it minimizes the sum-of-squares error:

$$\sum_{i=0}^{n-1} w_i (F(x^i) - f_i)^2$$

where $w = WEIGHTS$.

The functional form of F is, of course, restricted as follows:

$$F(x) := \sum_{j=0}^{k-1} \alpha_j \left(\sqrt{\|x - c_j\|^2 + \delta^2} \right) = \sum_{j=0}^{k-1} d_j \phi(\|x - c_j\|)$$

The function ϕ is called the radial function. It maps R^1 into R^1 . It needs to be defined only for the nonnegative reals. For the purpose of this routine, the user supplied a function:

$$\phi(r) = \sqrt{(r^2 + \delta^2)}$$

Note that the value of delta is defaulted to 1. It can be set by the user by using keyword *Delta*.

The default-basis function is called the Hardy multiquadric and is defined as:

$$\phi(r) = \sqrt{(r^2 + \delta^2)}$$

A key feature of this routine is the user's control over the selection of the basis function.

In order to obtain the default selection of centers, first compute the number of centers that will be on a grid and the number that will be on a random subset of the abscissa. Next, compute those centers on a grid. Finally, a random subset of abscissa is

obtained. This determines where the centers are placed. The selection of centers is discussed in more detail below.

First, the computed grid is restricted to have the same number of grid values in each of the “dimension” directions. Then, the number of centers placed on a grid, *num_gridded*, is computed as follows:

$$\begin{aligned}\alpha &= (\text{Ratio\_Centers})(\text{num\_centers}) \\ \beta &= \lfloor \alpha^{1/\text{dimension}} \rfloor \\ \text{num\_gridded} &= \beta^{\text{dimension}}\end{aligned}$$

Note that there are β grid values in each of the “dimension” directions. Then:

$$\text{num\_random} = (\text{num\_centers}) - (\text{num\_gridded})$$

How many centers are placed on a grid and how many are placed on a random subset of the abscissa is now known. The gridded centers are computed such that they are equally spaced in each of the “dimension” directions. The last problem is to compute a random subset, without replacement, of the abscissa. The selection is based on a random seed. The default seed is 234579. The user can change this using optional keyword `IMSL_RANDOM_SEED`. Once the subset is computed, the abscissa as centers is used.

Since the selection of good centers for a specific problem is an unsolved problem at this time, ultimate flexibility is given to the user; that is, the user can select centers using keyword `CENTERS`. As a rule of thumb, the centers should be interspersed with the abscissa.

The return value for this function is a pointer to the structure containing all the information necessary to evaluate the fit. This pointer is then passed to the [IMSL_RADBE](#) function to produce values of the fitted function.

Examples

Example 1: Fitting Noisy Data with Default Radial Function

In this example, `IMSL_RADBF` is used to fit noisy data. Four plots are generated using different values for *num_centers* as shown in [Figure 6-16](#). The plots generated by running this example are included after the code. Note that the triangles represent the placement of the centers.

```
PRO radbf_ex1
  !P.Multi = [0, 2, 2]
  ndata = 10
  noise_size = .05
  xydata = DBLARR(1, ndata)
```

```

fdata = DBLARR(ndata)
; Set up parameters.
IMSL_RANDOMOPT, Set = 234579
; Set the random number seed.
noise = 1 - 2 * IMSL_RANDOM(ndata, /Double)
; Generate the noisy data.
xydata(0, *) = 15 * IMSL_RANDOM(ndata)
fdata = REFORM(COS(xydata(0, *)) + noise_size * noise, ndata)
FOR i = 0, 3 DO BEGIN
    num_centers = ndata/3 + i
    ; Loop on different values of num_centers.
    radial_struct = IMSL_RADBF(xydata, fdata, num_centers)
    ; Compute the fit.
    a = DBLARR(1, 100)
    a(0, *) = 15 * FINDGEN(100)/99.
    fit = IMSL_RADBE(a, radial_struct)
    ; Evaluate fit.
    title = 'Fit with NUM_CENTERS = ' + $
        STRCOMPRESS(num_centers, /Remove_All)
    PLOT, xydata(0, *), fdata, Title = title, $
        Psym = 6, Yrange = [-1.25, 1.25]
    ; Plot results.
    OPLOT, a(0, *), fit
    ; Plot the original data as squares.
    OPLOT, radial_struct.CENTERS, $
        MAKE_ARRAY(num_centers, Value=-1.25), Psym = 5
    ; Plot the x-values of the centers as triangles.
END
END

```

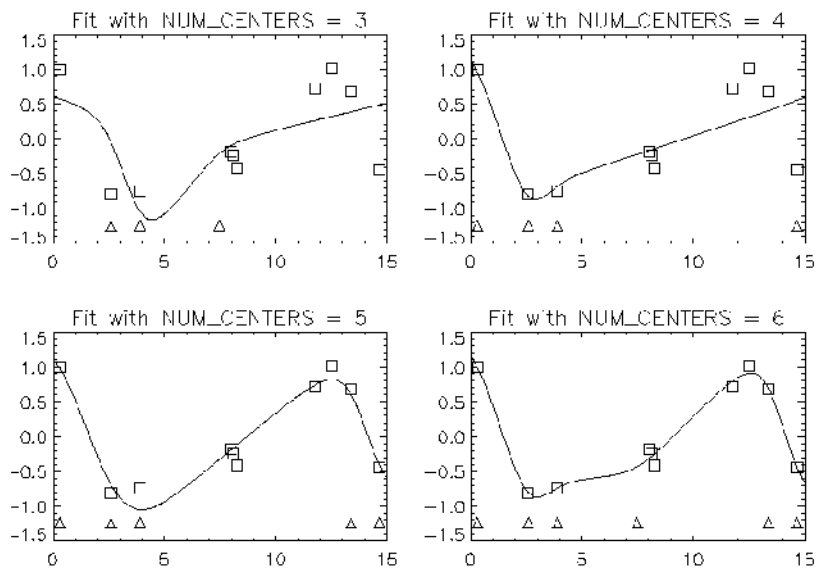


Figure 6-16: Fits using Differential Values for Num_centers

Example 2: Fitting Noisy Data with User-supplied Radial Function

This example fits the same data as the first example, but the user supplies the radial function and sets `RATIO_CENTERS` to zero. The radial function used in this example is $\phi(r) = \ln(1 + r^2)$. Four plots are generated using different values for `num_centers` as shown in [Figure 6-17](#). The plots generated by running this example are included after the code. Note that the triangles represent the placement of the centers.

```

FUNCTION user_fcn, distance
    ; Define the radial function.
    RETURN, ALOG(1 + distance^2)
END

PRO radbf_ex2
    ; Set up parameters.
    !P.Multi = [0, 2, 2]
    ndata = 10
    noise_size = .05
    xydata = DBLARR(1, ndata)
    fdata = DBLARR(ndata)
    IMSL_RANDOMOPT, Set = 234579
    ; Set the random number seed.
    noise = 1 - 2 * IMSL_RANDOM(ndata, /Double)
    ; Generate the noisy data.
    xydata(0, *) = 15 * IMSL_RANDOM(ndata)
    fdata = REFORM(COS(xydata(0,*)) + noise_size * noise, ndata)
    FOR i = 0, 3 DO BEGIN
        ; Loop on different values of num_centers.
        num_centers = ndata/3 + i
        radial_struct = IMSL_RADBF(xydata, fdata, $
            num_centers, Ratio_Centers = 0, Basis = 'user_fcn')
        ; Compute the fit.
        a = DBLARR(1, 100)
        a(0, *) = 15 * FINDGEN(100)/99.
        fit = IMSL_RADBE(a, radial_struct)
        ; Evaluate fit.
        title = 'Fit with NUM_CENTERS = ' + $
            STRCOMPRESS(num_centers, /Remove_All)
        PLOT, xydata(0,*), fdata, Title = title, $
            Psym = 6, Yrange = [-1.25, 1.25]
        ; Plot results.
        OPLOT, a(0, *), fit
        OPLOT, radial_struct.CENTERS, $
            MAKE_ARRAY(num_centers, Value = -1.25), Psym = 5
    END

```

END

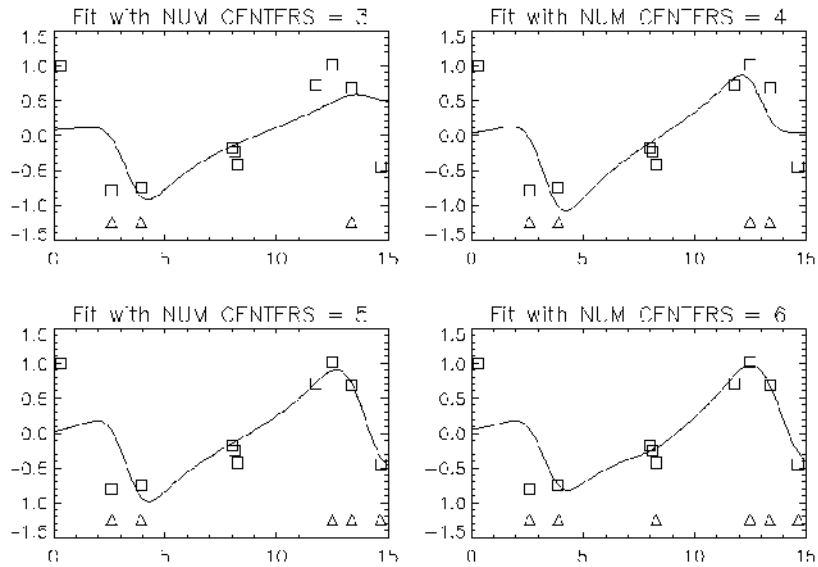


Figure 6-17: Fit using a User-Defined Radial Function

Example 3: Fitting a Surface to Three-dimensional Scattered Data

This example fits a surface to scattered data. The scattered data is generated using the function $f(x, y) = \exp(\ln(y + 1) \sin(x))$. The plots generated by running this example are included after the code as shown in [Figure 6-18](#) and [Figure 6-19](#).

```

FUNCTION f, x1, x2
    ; This function generates the scattered data function values.
    RETURN, EXP(ALOG10(x2 + 1)) * SIN(x1)
END

PRO radbf_ex3
    ; Set up initial parameters.
    IMSL_RANDOMOPT, Set = 123457
    ndata          = 50
    num_centers    = ndata
    xydata         = DBLARR(2, ndata)
    fdata          = DBLARR(ndata)

```

```

xrange          = 8
yrange          = 5
xydata(0,*) = xrange * IMSL_RANDOM(ndata, /Double)
xydata(1,*) = yrange * IMSL_RANDOM(ndata, /Double)
fdata(*)       = f(xydata(0, *), xydata(1, *))
; Generate data.
radial_struct = IMSL_RADBF(xydata, fdata, num_centers, Ratio=0)
; Compute fit using IMSL_RADBF.
WINDOW, /Free
; Plot results.
nx = 25
ny = 25
; Variables nx and ny are coarseness of the plotted surfaces.
xyfit          = DBLARR(2, nx * ny)
xyfit(0, *) = xrange * (FINDGEN(nx * ny)/ny)/(nx - 1)
xyfit(1, *) = yrange * (FINDGEN(nx * ny) MOD ny)/(ny - 1)

zfit = TRANSPOSE(REFORM(IMSL_RADBE(xyfit, Radial_Struct), $
    ny, nx))
; Use TRANSPOSE and REFORM in order to get the results
; into a form that SURFACE can use.
xt = xrange * FINDGEN(nx)/(nx-1)
yt = yrange * FINDGEN(ny)/(ny-1)
SURFACE, zfit, xt, yt, /Save, Zrange = [MIN(zfit), MAX(zfit)]
PLOTS, xydata(0, *), xydata(1, *), fdata, $
    /T3d, Psym = 4, Symsize = 2
; Plot the original data points over the surface plot.
WINDOW, /Free
orig = DBLARR(nx, ny)
FOR i = 0, (nx-1) DO FOR j = 0, (ny-1) DO $
    orig(i, j) = f(xt(i), yt(j))
SURFACE, orig, xt, yt, Zrange = [MIN(zfit), MAX(zfit)]
; Plot original function.
END

```

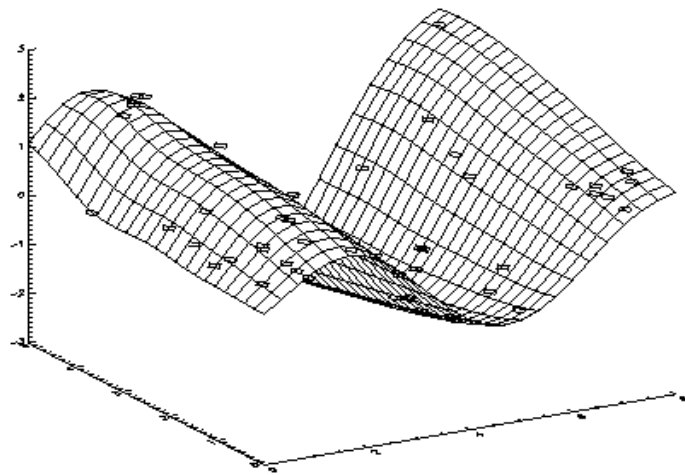


Figure 6-18: Surface Fit to Scattered Data

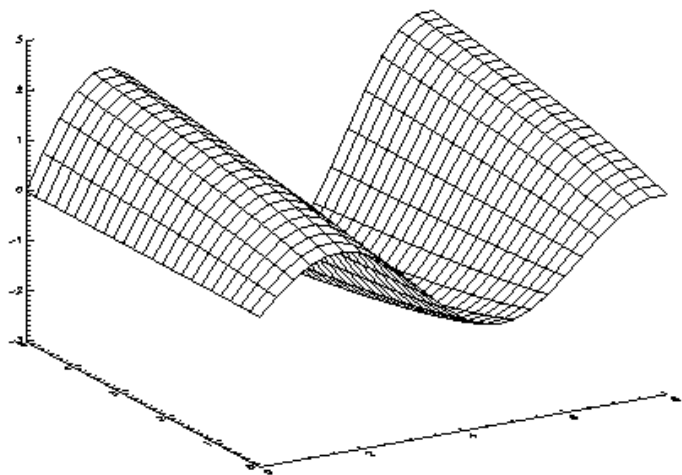


Figure 6-19: Function used to Generate Scattered Data

Version History

6.4	Introduced

IMSL_RADBE

The IMSL_RADBE function evaluates a radial-basis fit computed by IMSL_RADBF.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_RADBE(*abscissa*, *radial_fit*)

Return Value

An array containing the values of the radial-basis fit at the desired values.

Arguments

abscissa

Two-dimensional array containing the abscissa of the data points at which the fit is evaluated. Argument *abscissa* (*i*, *j*) is the abscissa value of the *j*-th data point in the *i*-th dimension.

radial_fit

Radial-basis structure to be used for the evaluation.

Discussion

The IMSL_RADBE function evaluates a radial-basis fit from data generated by [IMSL_RADBF](#).

Example

See “[IMSL_RADBF](#)” on page 266 for examples using IMSL_RADBE.

Version History

6.4	Introduced



Chapter 7

Quadrature

This section contains the following topics:

Overview: Quadrature	280	Quadrature Routines	283

Overview: Quadrature

This section introduces some of the mathematical concepts used in the IDL Advanced Math and Stats integration routines.

Univariate and Bivariate Quadrature

The first function in this chapter, [IMSL_INTFCN](#), is designed to compute approximations to integrals of the following form:

$$\int_a^b f(x)w(x)dx$$

or:

$$\int_a^b \int_{g(x)}^{h(x)} f(x, y) dx dy$$

The weight function w is used to incorporate known singularities (either algebraic or logarithmic) or to incorporate oscillations. The default action of this function assumes univariate quadrature, a weight function $w(x) = 1$, and the existence of endpoint singularities. Even if no endpoint singularities exist, the default method is still effective for general-purpose integration. If more efficiency is desired, then a more specialized method can be specified through the use of specific parameter and keyword combinations. The available methods can be summarized as follows, where the description refers to subsections of the documentation for the [IMSL_INTFCN](#) function:

- $w(x) = 1$
- Integration of a function with endpoint singularities (default method)
- Integration of a function based on Gauss-Kronrod rules
- Integration of a function with singular points given
- Integration of a function over an infinite or semi-infinite interval
- Integration of a smooth function using a nonadaptive method
- Integration of a two-dimensional iterated integral
- $w(x) = \sin \omega x$ or $w(x) = \cos \omega x$
- Integration of a function containing a sine or cosine factor

- Computing the Fourier sine or cosine transform
- $w(x) = (x-a)^\alpha (b-x)^\beta \ln(x-a) \ln(b-x)$, where the $\ln$ factors are optional
- Integration of functions with algebraic-logarithmic singularities
- $w(x) = 1/(x-c)$
- Integrals in the Cauchy principle value sense

The `IMSL_INTFCN` function returns an estimated answer R and provides keywords to specify a requested absolute error ϵ , the requested relative error ρ , and a named variable in which to return an estimate of the error E . These numbers are related in the equation:

$$\left| \int_a^b f(x)w(x)dx - R \right| \leq E \leq \max \left\{ \epsilon, \rho \left| \int_a^b f(x, y)dydx \right| \right\}$$

One situation that arises in univariate quadrature concerns the approximation of integrals when only tabular data is given. The functions above do not directly address this question. However, the standard method for handling this problem is to interpolate the data then integrate the interpolant. This can be accomplished by using the IDL Advanced Math and Stats spline interpolation functions with the spline integration function, “[IMSL_SPINTEG](#)” on page 230.

Multivariate Quadrature

Two functions, [IMSL_INTFCN](#) and [IMSL_INTFCNHYPER](#), have been included in this chapter that can be used to approximate certain multivariate integrals.

`IMSL_INTFCN` can be called with additional parameters and keywords to return an approximation to a two-dimensional iterated integral of the form:

$$\int_a^b \int_{g(x)}^{h(x)} f(x, y)dydx$$

The `IMSL_INTFCNHYPER` function returns an approximation to the integral of a function of n variables over a hyper-rectangle as shown in the equation:

$$\int_{a_0}^{b_0} \dots \int_{a_{n-1}}^{b_{n-1}} f(x_0, \dots, x_{n-1})dx_{n-1} \dots dx_0$$

When working with two-dimensional, tensor-product tabular data, use the IDL Advanced Math and Stats spline interpolation the [IMSL_BSINTERP](#) function, followed by the spline integration the [IMSL_SPINTEG](#) function.

Gauss Quadrature

For a fixed number of nodes, N , the Gauss quadrature rule is the unique rule that integrates polynomials of degree less than $2N$. These quadrature rules can be easily computed using the `IMSL_GQUAD` procedure, which produces the points $\{x_i\}$ and weights $\{w_i\}$ for $i = 1, \dots, N$ that satisfy:

$$\int_a^b f(x)w(x)dx = \sum_{i=1}^N f(x_i)w_i$$

for all functions f that are polynomials of degree less than $2N$. The weight functions w can be selected from [Table 7-1](#):

$w(x)$	Interval	Name
1	$(-1, 1)$	Legendre
$1/(\sqrt{1-x^2})$	$(-1, 1)$	Chebyshev 1st kind
$\sqrt{1-x^2}$	$(-1, 1)$	Chebyshev 2nd kind
e^{-x^2}	$(-\infty, \infty)$	Hermite
$(1+x)^\alpha(1-x)^\beta$	$(-1, 1)$	Jacobi
$e^{-x}x^a$	$(0, \infty)$	Generalized Laguerre
$1/\cosh(x)$	$(-\infty, \infty)$	Hyperbolic cosine

Table 7-1: Weight Functions

Where permissible, `IMSL_GQUAD` also computes Gauss-Radau and Gauss-Lobatto quadrature rules.

Quadrature Routines

Univariate and Bivariate Quadrature

[IMSL_INTFCN](#)—Integration of a user-defined univariate or bivariate function.

Arbitrary Dimension Quadrature

[IMSL_INTFCNHYPER](#)—Iterated integral on a hyper-rectangle.

[IMSL_INTFCN_QMC](#)—Integrates a function on a hyper-rectangle using a Quasi Monte Carlo method.

Gauss Quadrature

[IMSL_GQUAD](#)—Gauss quadrature formulas.

Differentiation

[IMSL_FCN_DERIV](#)—First, second, or third derivative of a function.

IMSL_INTFCN

The IMSL_INTFCN function integrates a user-supplied function. Using different combinations of keywords and parameters, several types of integration can be performed including the following:

- [IMSL_INTFCN: Functions with Endpoint Singularities \(the default method\)](#)
- [IMSL_INTFCN: Functions Based on Gauss-Kronrod Rules](#)
- [IMSL_INTFCN: Functions with Singular Points Given](#)
- [IMSL_INTFCN: Functions with Algebraic-logarithmic Singularities](#)
- [IMSL_INTFCN: Functions Over an Infinite or Semi-infinite Interval](#)
- [IMSL_INTFCN: Functions Containing a Sine or Cosine Factor](#)
- [IMSL_INTFCN: Computation of Fourier Sine or Cosine Transforms](#)
- [IMSL_INTFCN: Integrals in the Cauchy Principle Value Sense](#)
- [IMSL_INTFCN: Smooth Functions Using Nonadaptive Rule](#)
- [IMSL_INTFCN: Two-dimensional Iterated Integrals](#)

Different types of integration are specified by supplying different sets of parameters and keywords to the IMSL_INTFCN function. Refer to the discussion that pertains to the type of integration you wish to perform for the corresponding function syntax.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_INTFCN(f, a, b [, /DOUBLE] [, ERR_ABS=value]
[, ERR_EST=variable] [, ERR_REL=value] [, MAX_SUBINTER=value]
[, N_SUBINTER=variable] [, N_EVALS=variable])
```

Return Value

An estimate of the desired integral. If no value can be computed, the floating-point value NaN (Not a Number) is returned.

Arguments

f

A scalar string specifying the name of a user-supplied function to be integrated. The function f accepts one scalar parameter and returns a single scalar of the same type.

a

A scalar expression specifying the lower limit of integration.

b

A scalar expression specifying the upper limit of integration.

Keywords

The following keywords can be used in any combination with each method of integration except the nonadaptive method, which is triggered by the keyword SMOOTH.

DOUBLE

Set this keyword to perform computations using double precision.

ERR_ABS

Set this keyword to a value specifying the accuracy desired. Default: $\text{ERR\_ABS} = \text{SQRT}(\epsilon)$, where ϵ is the machine precision.

ERR_EST

Set this keyword equal to a named variable that will contain an estimate of the absolute value of the error.

ERR_REL

Set this keyword to a value specifying the relative accuracy desired. Default: $\text{ERR\_REL} = \text{SQRT}(\epsilon)$, where ϵ is the machine precision

MAX_SUBINTER

Set this keyword equal to the number of subintervals allowed. Default: $\text{MAX\_SUBINTER} = 500$.

N_SUBINTER

Set this keyword equal to a named variable that will contain the number of subintervals generated.

N_EVALS

Set this keyword equal to a named variable that will contain the number of evaluations of *Function*.

Discussion of Default Method

The default method used by `IMSL_INTFCN` is a general-purpose integrator that uses a globally adaptive scheme to reduce the absolute error. It subdivides the interval $[a, b]$ and uses a 21-point Gauss-Kronrod rule to estimate the integral over each subinterval. The error for each subinterval is estimated by comparison with the 10-point Gauss quadrature rule. The subinterval with the largest estimated error is then bisected, and the same procedure is applied to both halves. The bisection process is continued until either the error criterion is satisfied, the roundoff error is detected, the subintervals become too small, or the maximum number of subintervals allowed is reached. This method uses an extrapolation procedure known as the ϵ -algorithm. This method is based on the subroutine QAGS by Piessens et al. (1983).

Should the default method fail to produce acceptable results, consider one of the more specialized methods available by using method-specific keywords for this function.

Example

An estimate of:

$$\int_0^3 x^2 dx$$

is computed, then compared to the actual value.

```
.RUN
; Define the function to be integrated.
FUNCTION f, x
    RETURN, x^2
END

ans = IMSL_INTFCN('f', 0, 3)
; Call IMSL_INTFCN to compute the integral.
PM, 'Computed Answer:', ans
; Output the results.
Computed Answer:
```

```

          9.00000
PM, 'Exact - Computed:', 3^2 - ans
Exact - Computed:
          0.00000

```

Errors

The integration methods supported by IMSL_INTFCN may generate any of the following errors.

Warning Errors

MATH_ROUNDOFF_CONTAMINATION—Roundoff error, preventing the requested tolerance from being achieved, has been detected.

MATH_PRECISION_DEGRADATION—Degradation in precision has been detected.

MATH_EXTRAPOLATION_ROUNDOFF—Roundoff error in the extrapolation table, preventing requested tolerance from being achieved, has been detected.

MATH_EXTRAPOLATION_PROBLEMS—Extrapolation table, constructed for convergence acceleration of the series formed by the integral contributions of the cycles, does not converge to the requested accuracy.

MATH_BAD_INTEGRAND_BEHAVIOR—Bad integrand behavior occurred in one or more cycles.

Fatal Errors

MATH_DIVERGENT—Integral is probably divergent or slowly convergent.

MATH_MAX_SUBINTERVALS—Maximum number of subintervals allowed has been reached.

MATH_MAX_CYCLES—Maximum number of cycles allowed has been reached.

MATH_MAX_STEPS—Maximum number of steps allowed have been taken. The integrand is too difficult for this routine.

Version History

6.4	Introduced

IMSL_INTFCN: Functions Based on Gauss-Kronrod Rules

This version of the [IMSL_INTFCN](#) function integrates functions using a globally adaptive scheme based on Gauss-Kronrod rules.

Note

The **RULE** keyword must be supplied to use this integration method.

Syntax

Result = IMSL_INTFCN(*f*, *a*, *b*, RULE = {1-6} [, [RULE=value](#)])

Return Value

The value of:

$$\int_a^b f(x) dx$$

is returned. If no value can be computed, the floating-point value NaN (Not a Number) is returned.

Arguments

f

A scalar string specifying the name of a user-supplied function to be integrated. The function *f* accepts one scalar parameter and returns a single scalar of the same type.

a

A scalar expression specifying the lower limit of integration.

b

A scalar expression specifying the upper limit of integration.

Keywords

In addition to the global IMSL_INTFCN keywords listed in the main section under “Keywords” on page 285, the following keywords are available:

RULE

Set this keyword equal to an integer representing the Gauss-Kronrod rule to use. Possible values are:

Rule	Gauss-Kronrod Rule
1	7-15 points
2	10-21 points
3	15-31 points
4	20-41 points
5	25-51 points
6	30-61 points

Table 7-2: Corresponding Gauss-Kronrod Rules

Discussion

This method is a general-purpose integrator that uses a globally adaptive scheme to reduce the absolute error. It subdivides the interval $[a, b]$ and uses a $(2k+1)$ -point Gauss-Kronrod rule to estimate the integral over each subinterval. The error for each subinterval is estimated by comparison with the k -point Gauss quadrature rule. The subinterval with the largest estimated error is then bisected, and the same procedure is applied to both halves. The bisection process is continued until either the error criterion is satisfied, roundoff error is detected, the subintervals become too small, or the maximum number of subintervals allowed is reached. This method is based on the subroutine QAG by Piessens et al. (1983).

If this method is used, the function should be coded to protect endpoint singularities if they exist.

Example

The value of:

$$\int_0^1 \sin(1/x) dx$$

is computed. Since the integrand is oscillatory, RULE = 6 is used. The exact value is 0.50406706. The values of the actual and estimated error are machine dependent.

```
.RUN
; Define the function to be integrated.
FUNCTION f, x
    RETURN, SIN(1/x)
END

ans = IMSL_INTFCN('f', 0, 1, RULE=6)
; Call IMSL_INTFCN, to compute the integral based on the
; specified Gauss-Kronrod rule.
PM, 'Computed Answer:',ans
; Output the results.
Computed Answer:
    0.504051
    exact = .50406706
PM, 'EXACT - COMPUTED:', exact - ans
    Exact - Computed:
    1.62125e-05
```

Errors

See “[Errors](#)” on page 287.

IMSL_INTFCN: Functions with Singular Points Given

This version of the [IMSL_INTFCN](#) function integrates functions with singularity points given.

Note

The SING_PTS keyword must be supplied to use this integration method.

Syntax

Result = IMSL_INTFCN(*f*, *a*, *b*, SING_PTS=*points* [, [SING_PTS](#)=*vector*])

Return Value

The value of:

$$\int_a^b f(x) dx$$

is returned. If no value can be computed, the floating-point value NaN (Not a Number) is returned.

Arguments

f

A scalar string specifying the name of a user-supplied function to be integrated. The function *f* accepts one scalar parameter and returns a single scalar of the same type.

a

A scalar expression specifying the lower limit of integration.

b

A scalar expression specifying the upper limit of integration.

Keywords

In addition to the global IMSL_INTFCN keywords listed in the main section under “Keywords” on page 285, the following keywords are available:

SING_PTS

Set this keyword equal to a vector of abscissa values for the singularities. The values should be interior to the interval $[a, b]$.

Discussion

This method is a special-purpose integrator that uses a globally adaptive scheme to reduce the absolute error. It subdivides the interval $[a, b]$ into $N+1$ user-supplied subintervals, where N is the number of singular points, and uses a 21-point Gauss-Kronrod rule to estimate the integral over each subinterval. The error for each subinterval is estimated by comparison with the 10-point Gauss quadrature rule. The subinterval with the largest estimated error is then bisected, and the same procedure is applied to both halves. The bisection process is continued until either the error criterion is satisfied, the roundoff error is detected, the subintervals become too small, or the maximum number of subintervals allowed is reached. This method uses an extrapolation procedure known as the ϵ -algorithm. This method is based on the subroutine QAGP by Piessens et al. (1983).

Example

The value of:

$$\int_0^3 x^3 \ln|(x^2 - 1)(x^2 - 2)| dx = 61 \ln 2 + \frac{77}{4} \ln 7 - 27$$

is computed. The values of the actual and estimated error are machine dependent. Note that this subfunction never evaluates the user-supplied function at the user-supplied breakpoints.

```
.RUN
; Define the function to be integrated.
FUNCTION f, x
    RETURN, x^3 * ALOG(ABS((x^2 - 1) * $(
        (x^2 - 2)))
END

ans = IMSL_INTFCN('f', 0, 3, $
Sing_Pts = [1, SQRT(2)], N_Evals = nevals)
; Call IMSL_INTFCN using keyword Sing_Pts to specify
; the singular points.
```



```
PM, 'Computed Answer:', ans
; Output the results.
Computed Answer:
      52.7408
exact = 61 * ALOG(2) + (77/4.) * ALOG(7) - 27
PM, 'Exact - Computed:', exact - ans
Exact - Computed:
      -2.67029e-05
PM, 'Number of Function Evaluations:', nevals
Number of Function Evaluations:
      819
```

Errors

See “[Errors](#)” on page 287.

IMSL_INTFCN: Functions with Algebraic-logarithmic Singularities

This version of the [IMSL_INTFCN](#) function integrates functions with algebraic-logarithmic singularities.

Note

The *Alpha* and *Beta* arguments must be supplied to use this integration method.

Syntax

```
Result = IMSL_INTFCN(f, a, b, Alpha, Beta,  
/ALGEBRAIC | /ALG_LEFT_LOG | /ALG_LOG | /ALG_RIGHT_LOG)
```

Return Value

The value of:

$$\int_a^b f(x)w(x)dx$$

is returned, where $w(x)$ is defined by one of the keywords below. If no value can be computed, the floating-point value NaN (Not a Number) is returned.

Arguments

f

A scalar string specifying the name of a user-supplied function to be integrated. The function *f* accepts one scalar parameter and returns a single scalar of the same type.

a

A scalar expression specifying the lower limit of integration.

b

A scalar expression specifying the upper limit of integration.

Alpha

The strength of the singularity at *a*. Must be greater than -1 .

Beta

Strength of the singularity at b . Must be greater than -1 .

Keywords

In addition to the global IMSL_INTFCN keywords listed in the main section under “Keywords” on page 285, exactly one of the following keywords may be specified:

ALGEBRAIC

Set this keyword to use the weight function $(x-a)^\alpha(b-x)^\beta$. This is the default weight function for this method.

ALG_LEFT_LOG

Set this keyword to use the weight function $(x-a)^\alpha(b-x)^\beta \log(x-a)$.

ALG_LOG

Set this keyword to use the weight function $(x-a)^\alpha(b-x)^\beta \log(x-a) \log(x-b)$.

ALG_RIGHT_LOG

Set this keyword to use the weight function $(x-a)^\alpha(b-x)^\beta \log(x-b)$.

Discussion

This method is a special-purpose integrator that uses a globally adaptive scheme to reduce the absolute error. It computes integrals whose integrands have the special form $w(x)f(x)$, where $w(x)$ is a weight function. A combination of modified Clenshaw-Curtis and Gauss-Kronrod formulas is employed. This method is based on the subroutine QAWS, which is fully documented by Piessens et al. (1983).

If this method is used, the function should be coded to protect endpoint singularities if they exist.

Example

The value of:

$$\int_0^1 [(1+x)(1-x)]^{1/2} \ln(x) dx = \frac{(3 \ln(2) - 4)}{9}$$

is computed.

```
.RUN
; Define the function to be integrated.
FUNCTION f, x
    RETURN, Sqrt((1 + x))
END

ans = $
IMSL_INTFCN('f', 0, 1, /Alg_Left_Log, 1.0, .5 )
; Call IMSL_INTFCN with keyword Alg_Left_Log set and values for the
; method parameters alpha and beta.
PM, 'Computed Answer:', ans
; Output the results.
; Computed Answer: -0.213395
exact = (3 * ALOG(2) - 4)/9
PM, 'Exact - Computed:', exact - ans
; Exact - Computed: 1.49012e-08
```

Errors

See “[Errors](#)” on page 287.

IMSL_INTFCN: Functions Over an Infinite or Semi-infinite Interval

This version of the [IMSL_INTFCN](#) function integrates functions over an infinite or semi-infinite interval.

Note

One of the `INF_INF`, `INF_BOUND`, or `BOUND_INF` keywords must be supplied to use this integration method.

Syntax

Result = IMSL_INTFCN(*f*, /[INF_INF](#))

or

Result = IMSL_INTFCN(*f*, *Bound*, /[INF_BOUND](#) | /[BOUND_INF](#))

Return Value

The value of:

$$\int_a^b f(x) dx$$

is returned, where *a* and *b* are appropriate integration limits. If no value can be computed, the floating-point value NaN (Not a Number) is returned.

Arguments

f

A scalar string specifying the name of a user-supplied function to be integrated. The function *f* accepts one scalar parameter and returns a single scalar of the same type.

Bound

A scalar value specifying the finite limit of integration. If either of the keywords `INF_BOUND` or `BOUND_INF` are specified, this argument is required.

Keywords

In addition to the global IMSL_INTFCN keywords listed in the main section under “Keywords” on page 285, exactly one of the following keywords may be specified:

INF_INF

Set this keyword to integrate f over the range $(-\infty, \infty)$.

INF_BOUND

Set this keyword to integrate f over the range $(-\infty, \text{bound})$.

BOUND_INF

Set this keyword to integrate f over the range (bound, ∞) .

Discussion

This method is a special-purpose integrator that uses a globally adaptive scheme to reduce the absolute error. It initially transforms an infinite or semi-infinite interval into the finite interval $[0, 1]$. It then uses the same strategy that is used when integrating functions with singularity points given (see “IMSL_INTFCN: Functions with Singularity Points Given” on page 291). This method is based on the subroutine QAGI by Piessens et al. (1983).

If this method is used, the function should be coded to protect endpoint singularities if they exist.

Example

The value of:

$$\int_0^{\infty} \frac{\ln(x)}{1 + (10x)^2} dx = \frac{-\pi \ln(10)}{20}$$

is computed.

```
.RUN
; Define the function to be integrated.
FUNCTION f, x
    RETURN, ALOG(x)/(1 + (10 * x)^2)
END

ans = IMSL_INTFCN('f', 0, /Bound_Inf)
; Call IMSL_INTFCN with keyword Bound_Inf set. Notice that
; only lower limit of integration is given.
```

```
PM, 'Computed Answer:', ans
; Output the results.
Computed Answer:
    -0.361689
exact = -!Pi * ALOG(10)/20
PM, 'Exact - Computed:', exact - ans
Exact - Computed:
    5.96046e-08
```

Errors

See [“Errors”](#) on page 287.

IMSL_INTFCN: Functions Containing a Sine or Cosine Factor

This version of the [IMSL_INTFCN](#) function integrates functions containing a sine or a cosine factor.

Note

The *Omega* argument and one of the SINE, or COSINE keywords must be supplied to use this integration method.

Syntax

Result = IMSL_INTFCN(*f*, *a*, *b*, *Omega*
, /SINE | /COSINE [, MAX_MOMENTS=*value*])

Return Value

The value of:

$$\int_a^b f(x)w(\omega x)dx$$

where the weight function $w(\omega x)$ is defined by the keywords below, is returned. If no value can be computed, the floating-point value NaN (Not a Number) is returned.

Arguments

f

A scalar string specifying the name of a user-supplied function to be integrated. The function *f* accepts one scalar parameter and returns a single scalar of the same type.

a

A scalar expression specifying the lower limit of integration.

b

A scalar expression specifying the upper limit of integration.

Omega

A scalar expression specifying the frequency of the trigonometric weighting function.

Keywords

In addition to the global IMSL_INTFCN keywords listed in the main section under “Keywords” on page 285, the following keywords may be specified:

SINE

Set this keyword to use $\sin(\omega x)$ for the integration weight function. If SINE is supplied, COSINE must not be present.

COSINE

Set this keyword to use $\cos(\omega x)$ for the integration weight function. If COSINE is supplied, SINE must not be present.

MAX_MOMENTS

Set this keyword equal to a scalar expression specifying an upper bound on the number of Chebyshev moments that can be stored. Increasing (decreasing) this number may increase (decrease) execution speed and space used. Default: $\text{MAX\_MOMENTS} = 21$

Discussion

This method is a special-purpose integrator that uses a globally adaptive scheme to reduce the absolute error. It computes integrals whose integrands have the special form $w(x)f(x)$, where $w(x)$ is either $\cos(\omega x)$ or $\sin(\omega x)$. Depending on the length of the subinterval in relation to the size of ω , either a modified Clenshaw-Curtis procedure or a Gauss-Kronrod 7/15 rule is employed to approximate the integral on a subinterval. This method is based on the subroutine QAWO by Piessens et al. (1983).

If this method is used, the function should be coded to protect endpoint singularities if they exist.

Example

The value of:

$$\int_0^1 x^2 \sin(3\pi x) dx$$

is computed. The exact answer is:

$$\frac{(3\pi)^2 - 4}{(3\pi)^3}$$

```
.RUN
; Define the function to be integrated.
FUNCTION f, x
    RETURN, x^2
END

ans = IMSL_INTFCN('f', 0, 1, 3 * !Pi, /Sine)
; Call IMSL_INTFCN with Sine set and value for method
; parameter omega.
PM, 'Computed Answer:', ans
; Output the results.
Computed Answer:
    0.101325
exact = ((3 * !Pi)^2 - 2)/((3 * !pi)^3) - 2/(3 * !Pi)^3
PM, 'Exact - Computed:', exact - ans
Exact - Computed:
    0.00000
```

Errors

See “[Errors](#)” on page 287.

IMSL_INTFCN: Computation of Fourier Sine or Cosine Transforms

This version of the [IMSL_INTFCN](#) function computes Fourier sine or cosine transforms.

Note

The *Omega* argument and one of the SINE, or COSINE keywords must be supplied to use this integration method.

Syntax

```
Result = IMSL_INTFCN(f, a, Omega, /SINE | /COSINE
[, MAX_MOMENTS=value] [, N_CYCLES=variable])
```

Return Value

The value of:

$$\int_a^{\infty} f(x)w(\omega x)dx$$

where the weight function $w(\omega x)$ is defined by the keywords below, is returned. If no value can be computed, the floating-point value NaN (Not a Number) is returned.

Arguments

f

A scalar string specifying the name of a user-supplied function to be integrated. The function f accepts one scalar parameter and returns a single scalar of the same type.

a

A scalar expression specifying the lower limit of integration.

Omega

A scalar expression specifying the frequency of the trigonometric weighting function.

Keywords

In addition to the global IMSL_INTFCN keywords listed in the main section under “Keywords” on page 285, the following keywords may be specified:

SINE

Set this keyword to use $\sin(\omega x)$ for the integration weight function. If SINE is supplied, COSINE must not be present.

COSINE

Set this keyword to use $\cos(\omega x)$ for the integration weight function. If COSINE is supplied, SINE must not be present.

MAX_MOMENTS

Set this keyword equal to a scalar expression specifying an upper bound on the number of Chebyshev moments that can be stored. Increasing (decreasing) this number may increase (decrease) execution speed and space used. Default:

MAX_MOMENTS = 21

N_CYCLES

Set this keyword equal to a named variable that will contain the number of cycles.

Discussion

This method is a special-purpose integrator that uses a globally adaptive scheme to reduce the absolute error. It computes integrals whose integrands have the special form $w(x)f(x)$, where $w(x)$ is either $\cos(\omega x)$ or $\sin(\omega x)$. The integration interval is always semi-infinite of the form $[a, \text{infinity}]$. This method is based on the subroutine QAWF by Piessens et al. (1983).

If this method is used, the function should be coded to protect endpoint singularities if they exist.

Example

The value of:

$$\int_0^{\infty} \frac{\cos\left(\frac{\pi x}{2}\right)}{\sqrt{x}} dx = 1$$

is computed. Notice that the function is coded to protect for the singularity at zero.

```
.RUN
; Define the function to be integrated.
FUNCTION f, x
    IF (x EQ 0) THEN RETURN, x $
    ELSE RETURN, 1/SQRT(x)
END

ans = IMSL_INTFCN('f', 0, !Pi/2, /Cosine)
; Call IMSL_INTFCN with keyword Cosine set and a value for
; the method specific parameter omega.
PM, 'Computed Answer:', ans
; Output the results.
Computed Answer:
    1.00000
exact = 1.0
    PM, 'Exact - Computed:', exact - ans
Exact - Computed:
    -1.19209e-007
```

Errors

See “[Errors](#)” on page 287.

IMSL_INTFCN: Integrals in the Cauchy Principle Value Sense

This version of the [IMSL_INTFCN](#) function computes integrals of the form:

$$\int_a^b \frac{f(x)}{x - c} dx$$

in the Cauchy principal value sense.

Note

The *Singular_Pt* argument and the CAUCHY keyword must be supplied to use this integration method.

Syntax

Result = IMSL_INTFCN(*f*, *a*, *b*, *Singular_Pt*, /CAUCHY)

Return Value

The value of:

$$\int_a^b \frac{f(x)}{x - c} dx$$

is returned. If no value can be computed, the floating-point value NaN (Not a Number) is returned.

Arguments

f

A scalar string specifying the name of a user-supplied function to be integrated. The function *f* accepts one scalar parameter and returns a single scalar of the same type.

a

A scalar expression specifying the lower limit of integration.

b

A scalar expression specifying the upper limit of integration.

Singular_Pt

A scalar expression specifying the singular point. The singular point must not equal a or b .

Keywords

In addition to the global IMSL_INTFCN keywords listed in the main section under “Keywords” on page 285, the following keywords may be specified:

CAUCHY

Set this keyword to compute the specified integral in the Cauchy principal value sense.

Discussion

This method uses a globally adaptive scheme in an attempt to reduce the absolute error. It computes integrals whose integrands have the special form $w(x)f(x)$, where $w(x) = 1/(x - \text{Singular\_Pt})$. If *Singular_Pt* lies in the interval of integration, then the integral is interpreted as a Cauchy principal value. A combination of modified Clenshaw-Curtis and Gauss-Kronrod formulas is employed. The method is an implementation of the subroutine QAWC by Piessens et al. (1983).

If this method is used, the function should be coded to protect endpoint singularities if they exist.

Example

The Cauchy principal value of:

$$\int_{-1}^5 \frac{1}{x(5x^3 + 6)} dx = \frac{\ln(25/631)}{18}$$

is computed.

```
.RUN
; Define the function to be integrated.
FUNCTION f, x
    RETURN, 1/(5 * x^3 + 6)
END

ans = IMSL_INTFCN('f', -1, 5, 0, /Cauchy)
; Call IMSL_INTFCN with keyword Cauchy set.
PM, 'Computed Answer:', ans
```

```
; Output the results.  
Computed Answer:  
    -0.0899440  
exact = ALOG(125/631.)/18  
    PM, 'Exact - Computed:', exact - ans  
Exact - Computed:  
    1.49012e-08
```

Errors

See “[Errors](#)” on page 287.

IMSL_INTFCN: Smooth Functions Using Nonadaptive Rule

This version of the [IMSL_INTFCN](#) function integrates smooth functions using a nonadaptive rule.

Note

The SMOOTH keyword must be supplied to use this integration method.

Syntax

```
Result = IMSL_INTFCN(f, a, b, [, /SMOOTH] [, /DOUBLE] [, ERR_ABS=value]
[, ERR_EST=variable] [, ERR_REL=value])
```

Return Value

The value of:

$$\int_a^b f(x) dx$$

is returned. If no value can be computed, the floating-point value NaN (Not a Number) is returned.

Arguments

f

A scalar string specifying the name of a user-supplied function to be integrated. The function *f* accepts one scalar parameter and returns a single scalar of the same type.

a

A scalar expression specifying the lower limit of integration.

b

A scalar expression specifying the upper limit of integration.

Keywords

Because this integration method is nonadaptive, the global IMSL_INTFCN keywords listed in the main section do not apply. A complete list of the available keywords is given below. This method requires the use of keyword SMOOTH.

SMOOTH

Set this keyword to use a nonadaptive rule to compute the integral.

DOUBLE

Set this keyword to perform computations using double precision.

ERR_ABS

Set this keyword to a value specifying the accuracy desired. Default: $\text{ERR\_ABS}=\text{SQRT}(\epsilon)$, where ϵ is the machine precision

ERR_EST

Set this keyword equal to a named variable that will contain an estimate of the absolute value of the error.

ERR_REL

Set this keyword to a value specifying the relative accuracy desired. Default: $\text{ERR\_REL}=\text{SQRT}(\epsilon)$, where ϵ is the machine precision

Discussion

This method is designed to integrate smooth functions. It implements a nonadaptive quadrature procedure based on nested Paterson rules of order 10, 21, 43, and 87. These rules are positive quadrature rules with degree of accuracy 19, 31, 64, and 130, respectively. This method applies these rules successively, estimating the error until either the error estimate satisfies the user-supplied constraints or the last rule is applied.

This method is not very robust, but for certain smooth functions, it can be efficient. This method is based on the subroutine QNG by Piessens et al. (1983). If this method is used, the function should be coded to protect endpoint singularities if they exist.

Example

The value of:

$$\int_0^2 x e^x dx = e^2 + 1$$

is computed.

```
.RUN
; Define the function to integrate.
FUNCTION f, x
    RETURN, x * EXP(x)
END

ans = IMSL_INTFCN('f', 0, 2, /Smooth)
; Call IMSL_INTFCN with keyword Smooth set.
PM, 'Computed Answer:', ans
Computed Answer:
      8.38906
exact = EXP(2) + 1
      PM, 'Exact - Computed:', exact - ans
Exact - Computed:
      9.53674e-07
```

Errors

See “[Errors](#)” on page 287.

IMSL_INTFCN: Two-dimensional Iterated Integrals

This version of the [IMSL_INTFCN](#) function integrates two-dimensional iterated integrals.

Note

The `TWO_DIMENSIONAL` keyword must be supplied to use this integration method.

Syntax

Result = IMSL_INTFCN(*f*, *a*, *b*, *g*, *h*, /[TWO_DIMENSIONAL](#))

Return Value

The value of:

$$\int_a^b \int_{g(x)}^{h(x)} f(x, y) dy dx$$

is returned. If no value can be computed, the floating-point value NaN (Not a Number) is returned.

Arguments

f

A scalar string specifying the name of a user-supplied function to be integrated. The function *f* accepts one scalar parameter and returns a single scalar of the same type.

a

A scalar expression specifying the lower limit of integration.

b

A scalar expression specifying the upper limit of integration.

g

Scalar string specifying the name of a user-supplied IDL Advanced Math and Stats function used to evaluate the lower limit of the inner integral. Function *g* accepts one scalar parameter and returns a single scalar of the same type.

h

Scalar string specifying the name of a user-supplied IDL Advanced Math and Stats function used to evaluate the upper limit of the inner integral. Function *h* accepts one scalar parameter and returns a single scalar of the same type.

Keywords

In addition to the global IMSL_INTFCN keywords listed in the main section under “[Keywords](#)” on page 285, the following keyword must be specified:

TWO_DIMENSIONAL

Set this keyword to integrate a two-dimensional iterated integral.

Discussion

This method approximates the following two-dimensional iterated integral:

$$\int_a^b \int_{g(x)}^{h(x)} f(x, y) dy dx$$

The lower-numbered rules are used for less smooth integrands, while the higher-order rules are more efficient for smooth (oscillatory) integrands.

If this method is used, the function should be coded to protect endpoint singularities if they exist.

Example

This example computes the value of the integral:

$$\int_0^1 \int_x^{2x} \sin(x + y) dy dx$$

```
.RUN
; Define the function to be integrated.
FUNCTION f, x, y
```

```

        RETURN, SIN(x + y)
    END

    .RUN
    ; Define the function for the lower limit of the inner integral.
    FUNCTION g, x
        RETURN, x
    END

    .RUN
    ; Define the function for the upper limit of the inner integral.
    FUNCTION h, x
        RETURN, 2 * x
    END

    ans = IMSL_INTFCN('f',0,1,'g','h',/Two_Dimensional)
    ; Call IMSL_INTFCN with keyword Two_Dimensional set and the names
    ; of the functions defining the limits of the inner integral.
    PM, 'Computed Answer:', ans
    Computed Answer:
        0.407609
    exact = -SIN(3)/3 + SIN(2)/2
    PM, 'Exact - Computed:', exact - ans
    Exact - Computed:
        -5.96046e-08

```

Errors

See “[Errors](#)” on page 287.

IMSL_INTFCNHYPER

The IMSL_INTFCNHYPER function integrates a function on a hyper-rectangle as follows:

$$\int_{a_0}^{b_0} \dots \int_{a_{n-1}}^{b_{n-1}} f(x_0, \dots, x_{n-1}) dx_{n-1} \dots dx_0$$

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_INTFCNHYPER(f, a, b [, ERR_ABS=value] [, ERR_EST=variable]
    [, ERR_REL=value] [, MAX_EVALS=value])
```

Return Value

The value of the hyper-rectangle function is returned. If no value can be computed, the floating-point value NaN (Not a Number) is returned.

Arguments

f

A scalar string specifying the name of a user-supplied function to be integrated. The function *f* accepts an array of data points at which the function is to be evaluated and returns the scalar value of the function.

a

A vector specifying the lower limit of integration.

b

A vector specifying the upper limit of integration.

Keywords

ERR_ABS

Set this keyword to a value specifying the accuracy desired. Default:
 $\text{ERR\_ABS} = \text{SQRT}(\epsilon)$, where ϵ is the machine precision

ERR_EST

Set this keyword equal to a named variable that will contain an estimate of the absolute value of the error.

ERR_REL

Set this keyword to a value specifying the relative accuracy desired. Default:
 $\text{ERR\_REL} = \text{SQRT}(\epsilon)$, where ϵ is the machine precision

MAX_EVALS

Set this keyword to a scalar value specifying the number of evaluations allowed. Default: $\text{MAX\_EVALS} = 1,000,000$ for $n \leq 2$ and $\text{MAX\_EVALS} = 256^n$ for $n > 2$, where n is the number of independent variables of f .

Discussion

The `IMSL_INTFCNHYPER` function approximates the following n -dimensional iterated integral:

$$\int_{a_0}^{b_0} \dots \int_{a_{n-1}}^{b_{n-1}} f(x_0, \dots, x_{n-1}) dx_{n-1} \dots dx_0$$

An estimate of the error is returned in the optional keyword `ERR_EST`. The approximation is achieved by iterated applications of product Gauss formulas. The integral is first estimated by a two-point, tensor-product formula in each direction. Then, for $(i = 0, \dots, n-1)$, the function calculates a new estimate by doubling the number of points in the i -th direction, but halving the number immediately afterwards if the new estimate does not change appreciably. This process is repeated until either one complete sweep results in no increase in the number of sample points in any dimension, the number of Gauss points in one direction exceeds 256, or the number of function evaluations needed to complete a sweep exceeds `MAX_EVALS`.

Example

This example computes the integral of:

$$e^{-(x_0^2 + x_1^2 + x_2^2)}$$

on an expanding cube. The values of the error estimates are machine dependent. The exact integral over R is $\pi^{3/2}$.

```
.RUN
; Define the function to be integrated.
FUNCTION f, x
    RETURN, EXP(-TOTAL(x^2))
END

limit = !Pi^1.5
; Compute the exact value of the integral.
PM, '    Limit:', limit
    Limit:      5.56833
FOR i = 1, 6 DO BEGIN $
    a = [-i/2., -i/2., -i/2.] &$
    b = [i/2., i/2., i/2.] &$
    ans = IMSL_INTFCNHYPHER('f', a, b) &$
    PRINT, 'integral = ', ans, ' limit = ', limit
; Compute values of the integral over expanding cubes and
; output the results after each call to IMSL_INTFCNHYPHER.
    integral = 0.785213 limit = 5.56833
    integral = 3.33231 limit = 5.56833
    integral = 5.02107 limit = 5.56833
    integral = 5.49055 limit = 5.56833
    integral = 5.56135 limit = 5.56833
    integral = 5.56771 limit = 5.56833
```

Errors

Warning Errors

MATH_MAX_EVALS_TOO_LARGE—The keyword **MAX_EVALS** was set too large.

Fatal Errors

MATH_NOT_CONVERGENT—Maximum number of function evaluations has been reached, and convergence has not been attained.

Version History

6.4	Introduced

IMSL_INTFCN_QMC

The IMSL_INTFCN_QMC function integrates a function on a hyper-rectangle using a quasi-Monte Carlo method.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_INTFCN_QMC(f, a, b [, BASE=value] [, /DOUBLE]
    [, ERR_ABS=value] [, ERR_EST=variable] [, ERR_REL=value]
    [, MAX_EVALS=value] [, SKIP=value])
```

Return Value

The value of:

$$\int_{a_0}^{b_0} \dots \int_{a_{n-1}}^{b_{n-1}} f(x_0, \dots, x_{n-1}) dx_{n-1} \dots dx_0$$

is returned. If no value can be computed, the floating-point value NaN (Not a Number) is returned.

Arguments

f

A scalar string specifying the name of a user-supplied function to be integrated. The function f accepts an array of data points at which the function is to be evaluated and returns the scalar value of the function.

a

A vector specifying the lower limit of integration.

b

A vector specifying the upper limit of integration.

Keywords

BASE

Set this keyword equal to the value of BASE used to compute the Faure sequence.

DOUBLE

Set this keyword to perform computations using double precision.

ERR_ABS

Set this keyword to a value specifying the accuracy desired. Default:
 $\text{ERR\_ABS}=1 \times e^{-4}$.

ERR_EST

Set this keyword equal to a named variable that will contain an estimate of the absolute value of the error.

ERR_REL

Set this keyword to a value specifying the relative accuracy desired. Default:
 $\text{ERR\_REL}=1 \times e^{-4}$.

MAX_EVALS

Set this keyword equal to the number of evaluations allowed. If MAX_EVALS is not supplied, the number of evaluations is unlimited.

SKIP

Set this keyword equal to the value of SKIP used to compute the Faure sequence.

Discussion

Integration of functions over hypercubes by direct methods, such as [IMSL_INTFCNHYPER](#), is practical only for fairly low dimensional hypercubes. This is because the amount of work required increases exponential as the dimension increases.

An alternative to direct methods is Monte Carlo, in which the integral is evaluated as the value of the function averaged over a sequence of randomly chosen points. Under mild assumptions on the function, this method will converge like $1/n^{1/2}$, where n is the number of points at which the function is evaluated.

It is possible to improve on the performance of Monte Carlo by carefully choosing the points at which the function is to be evaluated. Randomly distributed points tend to be non-uniformly distributed. The alternative to a sequence of random points is a low-discrepancy sequence. A low-discrepancy sequence is one that is highly uniform.

This function is based on the low-discrepancy Faure sequence, as computed by [IMSL_FAURE_NEXT_PT](#).

Example

```

FUNCTION F, x
  S = 0.0
  sign = -1.0
  FOR i = 0, N_ELEMENTS(x)-1 DO BEGIN
    prod = 1.0
    FOR j = 0, i DO BEGIN
      prod = prod*x(j)
    END
    S = S + sign*prod
    sign = -sign
  END
  RETURN, S
END

ndim = 10
a = FLTARR(ndim)
a(*) = 0
b = FLTARR(ndim)
b(*) = 1
result = IMSL_INTFCN_QMC( 'f', a, b)
PM, result
-0.333010

```

Version History

6.4	Introduced

IMSL_GQUAD

The IMSL_GQUAD procedure computes a Gauss, Gauss-Radau, or Gauss-Lobatto quadrature rule with various classical weight functions.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
IMSL_GQUAD, n, weights, points [, /CHEBY_FIRST] [, /CHEBY_SECOND]
[, /COSH] [, /DOUBLE] [, /HERMITE] [, JACOBI=vector]
[, LAGUERRE=parameter] [, FIXED_POINTS=vector]
```

Arguments

n

The number of quadrature points.

weights

A named variable that will contain an array of length n containing the quadrature weights.

points

A named variable that will contain an array of length n containing quadrature points. The default action of this routine is to produce the Gauss Legendre points and weights.

Keywords

CHEBY_FIRST

Set this keyword to compute the Gauss points and weights using the weight function:

$$1/\sqrt{1-x^2}$$

on the interval $(-1, 1)$.

CHEBY_SECOND

Set this keyword to compute the Gauss points and weights using the weight function:

$$\sqrt{1 - x^2}$$

on the interval $(-1, 1)$.

COSH

Set this keyword to compute the Gauss points and weights using the weight function $1/\cosh(x)$ on the interval $(-\infty, \infty)$.

DOUBLE

Set this keyword to perform computations using double precision.

HERMITE

Set this keyword to compute the Gauss points and weights using the weight function $\exp(-x^2)$ on the interval $(-\infty, \infty)$.

JACOBI

Set this keyword equal to a two-element vector containing the parameters α and β to be used in the weight function $(1-x)^\alpha(1+x)^\beta$. If this keyword is present, IMSL_GQUAD computes the Gauss points and weights using the weight function $(1-x)^\alpha(1+x)^\beta$ on the interval $(-1, 1)$.

LAGUERRE

Set this keyword equal to a scalar parameter α to be used in the weight function $\exp(-x)x^\alpha$. If this keyword is present, IMSL_GQUAD computes the Gauss points and weights using the weight function $\exp(-x)x^\alpha$ on the interval $(0, \infty)$.

FIXED_POINTS

Set this keyword equal to a one- or two-element vector specifying the fixed points.

- If **FIXED_POINTS** is a scalar or one-element vector, IMSL_GQUAD computes the Gauss-Radau points and weights using the specified weight function and the fixed point. This formula integrates polynomials of degree less than $2N-1$ exactly.
- If **FIXED_POINTS** is a two-element vector, IMSL_GQUAD computes the Gauss-Lobatto points and weights using the specified weight function and the

fixed points. This formula integrates polynomials of degree less than $2N-2$ exactly.

Discussion

The IMSL_GQUAD procedure produces the points and weights for the Gauss, Gauss-Radau, or Gauss-Lobatto quadrature formulas for some of the most popular weights. The default weight is the weight function identically equal to 1 on the interval $(-1, 1)$. In fact, it is slightly more general than this suggests because the extra one or two points that can be specified do not have to lie at the endpoints of the interval. This procedure is a modification of the subroutine GAUSSQUADRULE (Golub and Welsch 1969).

In the default case, the procedure returns points in $x = \text{points}$ and weights in $w = \text{weights}$ so that:

$$\int_a^b f(x)w(x)dx = \sum_{i=0}^{N-1} f(x_i)w_i$$

for all functions f that are polynomials of degree less than $2N$.

If the keyword FIXED_POINTS is specified, then one or two of the above x_i is equal to the values specified by FIXED_POINTS. In general, the accuracy of the above quadrature formula degrades when n increases. The quadrature rule integrates all functions f that are polynomials of degree less than $2N - F$, where F is the number of fixed points.

Example

This example computes the three-point Gauss Legendre quadrature points and weights, then uses them to approximate the integrals as follows:

$$\int_{-1}^1 x^i dx \quad i=0, \dots, 6$$

Notice that the integrals are exact for the first six monomials, but the last approximation is in error. In general, the Gauss rules with k -points integrate polynomials with degree less than $2k$ exactly.

```
IMSL_GQUAD, 3, weights, points
; Call IMSL_GQUAD to get the weights and points.
error = FLTARR(7)
; Define an array to hold the errors.
FOR i = 0, 6 DO error(i) = $
    (TOTAL(weights*(points^i)) - (1-(i MOD 2))*2./(i+1))
```



```
        ; Compute the errors for seven monomials.  
PM, 'Error:', error  
; Output the results.  
Error:  
    -2.38419e-07  
     2.68221e-07  
    -5.96046e-08  
     2.08616e-07  
     2.98023e-08  
     1.78814e-07  
    -0.0457142
```

Version History

6.4	Introduced

IMSL_FCN_DERIV

The IMSL_FCN_DERIV function computes the first, second, or third derivative of a user-supplied function.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_FCN_DERIV(f, x [, /DOUBLE] [, ORDER=value]  
[, STEPSIZE=value] [, TOLERANCE=value])
```

Return Value

An estimate of the first, second or third derivative of f at x . If no value can be computed, NaN is returned.

Arguments

f

A scalar string specifying a user-supplied function whose derivative at x will be computed.

x

The point at which the derivative will be evaluated.

Keywords

DOUBLE

Set this keyword to perform computations using double precision.

ORDER

Set this keyword equal to the order of the desired derivative (1, 2 or 3). Default: ORDER = 1

STEPSIZE

Set this keyword equal to the beginning value used to compute the size of the interval for approximating the derivative. STEPSIZE must be chosen small enough that f is defined and reasonably smooth in the interval $(x - 4.0*\text{STEPSIZE}, x + 4.0*\text{STEPSIZE})$, yet large enough to avoid roundoff problems. Default: $\text{STEPSIZE} = 0.01$

TOLERANCE

Set this keyword equal to the relative error desired in the derivative estimate. Convergence is assumed when $(2/3) |d_2 - d_1| < \text{TOLERANCE}$, for two successive derivative estimates, d_1 and d_2 . Default: $\text{TOLERANCE} = \sqrt[4]{\epsilon}$ where ϵ is machine epsilon.

Discussion

The `IMSL_FCN_DERIV` function produces an estimate to the first, second, or third derivative of a function. The estimate originates from first computing a spline interpolant to the input function using values within the interval $(x - 4.0*\text{STEPSIZE}, x + 4.0*\text{STEPSIZE})$, then differentiating the spline at x .

Examples

Example 1

This example obtains the approximate first derivative of the function $f(x) = -2\sin(3x/2)$ at the point $x = 2$.

```
FUNCTION fcn, x
    f = -2*SIN(1.5*x)
    RETURN, f
END

deriv1 = IMSL_FCN_DERIV('fcn', 2.0)
PRINT, "f'(x)    = ", deriv1
f'(x)    =      2.97008
```

Example 2

This example obtains the approximate first, second, and third derivative of the function $f(x) = -2\sin(3x/2)$ at the point $x = 2$.

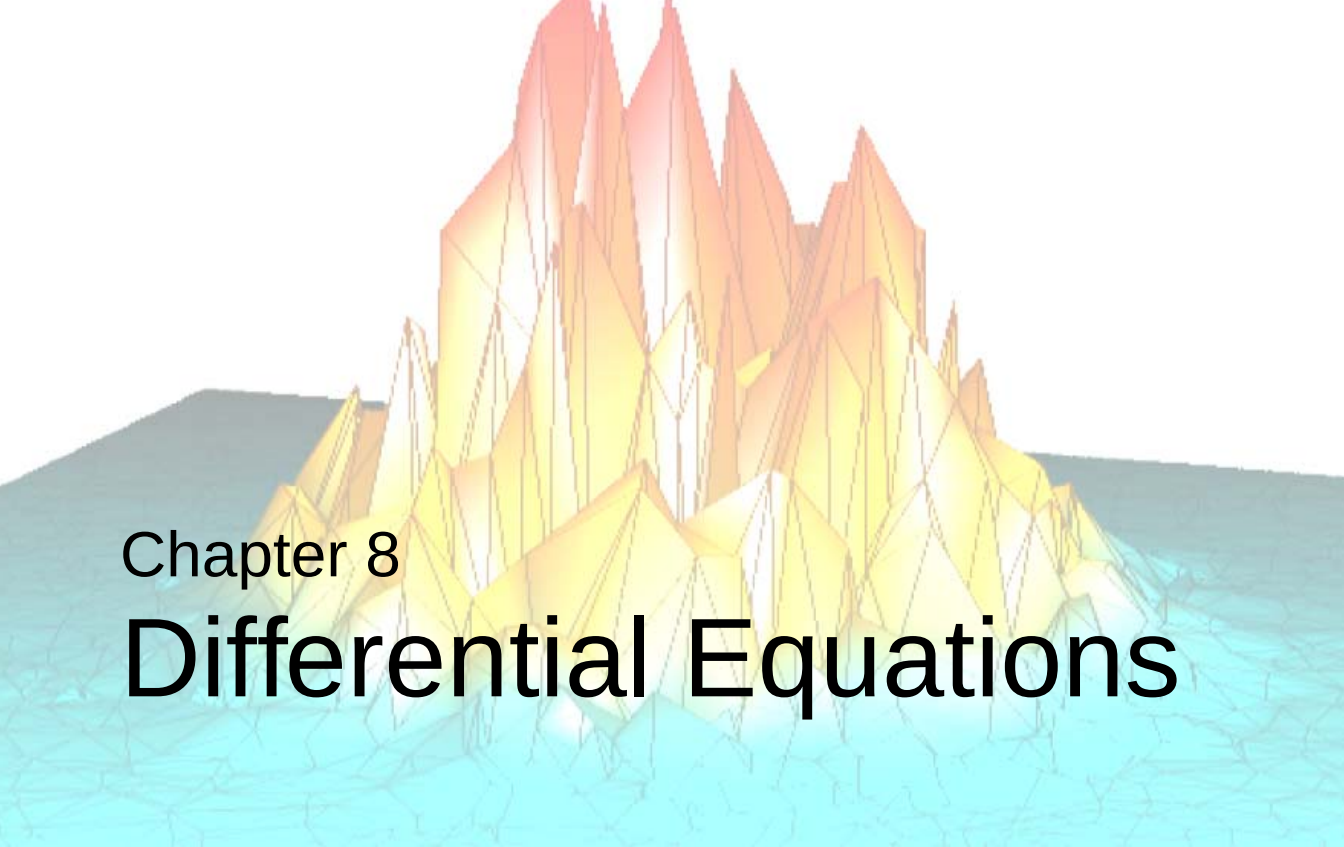
```
FUNCTION fcn, x
    f = -2*SIN(1.5*x)
```

```
        RETURN,  f
    END

    deriv1 = IMSL_FCN_DERIV('fcn', 2.0, /Double)
    deriv2 = IMSL_FCN_DERIV('fcn', 2.0, ORDER = 2, /Double)
    deriv3 = IMSL_FCN_DERIV('fcn', 2.0, ORDER = 3, /Double)
    PRINT, "f'(x)  = ", deriv1, ',  error =', $
        ABS(deriv1 + 3.0*COS(1.5*2.0))
    f'(x)  =      2.9699775,  error =   1.1094893e-07
    PRINT, "f''(x) = ", deriv2, ',  error =', $
        ABS(deriv2 - 4.5*SIN(1.5*2.0))
    f''(x) =      0.63504004,  error =   5.1086361e-08
    PRINT, "f'''(x) = ", deriv3, ',  error =', $
        ABS(deriv3 - 6.75*COS(1.5*2.0))
    f'''(x) =      -6.6824494,  error =   1.1606068e-08
```

Version History

6.4	Introduced



Chapter 8

Differential Equations

This section contains the following topics:

Overview: Differential Equations	330	Differential Equations Routines	332

Overview: Differential Equations

This section introduces some of the mathematical concepts used with IDL Advanced Math and Stats.

Ordinary Differential Equations

An *ordinary differential equation* is an equation involving one or more dependent variables called y_i , one independent variable, t , and derivatives of the y_i with respect to t .

In the *initial value problem* (IVP), the initial or starting values of the dependent variables y_i at a known value $t = t_0$ are given. Values of $y_i(t)$ for $t > t_0$ or $t < t_0$ are required.

The IMSL_ODE function solves the IVP for ODEs of the form:

$$\frac{dy_i}{dt} = y_i' = f_i(t, y_0 \dots y_{N-1}) \quad i = 0, \dots, N-1$$

with $y_i(t = t_0)$ specified. Here, f_i is a user-supplied function that must be evaluated at any set of values $(t, y_0, \dots, y_{N-1})$, $i = 0, \dots, N-1$.

The previous problem statement is abbreviated by writing it as a *system* of first-order ODEs, $y(t) = [y_0(t), \dots, y_{N-1}(t)]^T$, $f(t, y) = [f_0(t, y), \dots, f_{N-1}(t, y)]^T$, so that the problem becomes $y' = f(t, y)$ with initial values $y(t_0)$.

The system:

$$\frac{dy}{dt} = y' = f(t, y)$$

is said to be stiff if some of the eigenvalues of the Jacobian matrix:

$$\{(\partial y_i')/(\partial y_j)\}$$

are large and negative. This is frequently the case for differential equations modeling the behavior of physical systems such as chemical reactions proceeding to equilibrium where subspecies effectively complete their reactions in different epochs. An alternate model concerns discharging capacitors such that different parts of the system have widely varying decay rates (or *time constants*).

Users typically identify stiff systems by the fact that certain numerical differential equation solvers, such as the Runge-Kutta-Verner fifth-order and sixth-order method, are inefficient or they fail completely. Special methods are often required. The most common inefficiency is that a large number of evaluations of $f(t, y)$ and, hence, an excessive amount of computer time are required to satisfy the accuracy and stability requirements of the software. In such cases, the keyword `R_K_V` should not be

specified when using the IMSL_ODE function. For more about stiff systems, see Gear (1971, Chapter 11) or Shampine and Gear (1979).

Partial Differential Equations

The routine IMSL_PDE_MOL solves the IVP problem for systems of the form:

$$\frac{\partial u_i}{\partial t} = f_i \left(x, t, u_1, \dots, u_n, \frac{\partial u_1}{\partial x}, \dots, \frac{\partial u_N}{\partial x}, \frac{\partial^2 u_1}{\partial x^2}, \dots, \frac{\partial^2 u_N}{\partial x^2} \right)$$

subject to the boundary conditions:

$$\alpha_1^{(i)} u_i(a) + \beta_1^{(i)} \frac{\partial u_i}{\partial x}(a) = \gamma_1(t)$$

$$\alpha_2^{(i)} u_i(b) + \beta_2^{(i)} \frac{\partial u_i}{\partial x}(b) = \gamma_2(t)$$

and subject to the initial conditions:

$$u_i(x, t = t_0) = g_i(x)$$

for $i = 1, \dots, N$. Here, $f_i, g_i, :$

$$\alpha_j^{(i)}, \text{ and } \beta_j^{(i)}$$

are user-supplied, $j = 1, 2$.

The routine IMSL_POISSON2D solves Laplace's, Poisson's, or Helmholtz's equation in two dimensions. This routine uses a fast Poisson method to solve a PDE of the form:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + cu = f(x, y)$$

over a rectangle, subject to boundary conditions on each of the four sides. The scalar constant c and the function f are user specified.

Differential Equations Routines

[IMSL_ODE](#)—Adams-Gear or Runge-Kutta method.

[IMSL_PDE_MOL](#)—Solves a system of partial differential equations using the method of lines.

[IMSL_POISSON2D](#)—Solves Poisson's or Helmholtz's equation on a two-dimensional rectangle.

IMSL_ODE

The IMSL_ODE function solves an initial value problem, which is possibly stiff, using the Adams-Gear methods for ordinary differential equations. Using keywords, the Runge-Kutta-Verner fifth-order and sixth-order method can be used if you know the problem is not stiff.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_ODE(t, y, f[, /DOUBLE] [, FLOOR=value] [, HINIT=value]
[, HMAX=value] [, HMIN=value] [, MAX_EVALS=value]
[, MAX_STEPS=value] [, N_STEPS=variable] [, N_EVALS=variable]
[, NORM=value] [, R_K_V=value] [, SCALE=value] [, TOLERANCE=value]
[, JACOBIAN=string] [, MAX_ORD=value] [, METHOD=value]
[, MITER=value] [, N_JEVALS=value] )
```

Return Value

A two-dimensional array containing the approximate solutions for each specified value of the independent variable. The elements (*i*, *) are the solutions for the *i*-th variable.

Arguments

t

One-dimensional array containing values of the independent variable. Parameter *t*(0) should contain the initial independent variable value, and the remaining elements of *t* should be filled with values of the independent variable at which a solution is desired.

y

Array containing the initial values of the dependent variables.

f

Scalar string specifying a user-supplied function to evaluate the right-hand side. This function takes two parameters, t and y , where t is the current value of the independent variable and y is defined above.

The return value of this function is an array defined by the following equation:

$$f(t, y) = \frac{dy}{dt} = y'$$

Keywords**DOUBLE**

If present and nonzero, double precision is used.

FLOOR

Used with `IMSL_NORM`. Provides a positive lower bound for the error norm option with value 2. Default: `FLOOR = 1.0`

HINIT

Scalar value used for the initial value for the step size h . Steps are applied in the direction of integration. Default: $HINIT = 0.001 | t(i+1) - t(i) |$

HMAX

Scalar value used as the maximum value for the step size h . If keyword `R_K_V` is set, $HMAX = 2.0$ is used. Default: largest machine-representable number

HMIN

Scalar value used as the minimum value for the step size h . Default: $HMIN = 0.0$

MAX_EVALS

Integer value used in the maximum number of function evaluations allowed per time step. Default: `MAX_EVALS = no enforced limit`

MAX_STEPS

Integer value used in the maximum number of steps allowed per time step. Default: `MAX_STEPS = 500`

N_STEPS

Named variable into which the array containing the number of steps taken at each value of t is stored.

N_EVALS

Named variable into which the array containing the number of function evaluations used at each value of t is stored.

NORM

Switch determining the error norm. In the following, e_i is the absolute value of the error estimate for y_i .

- 0—Minimum of the absolute error and the relative error equals the maximum of $e_i/\max(|y_i|, 1)$ for $i = 0, \dots, N\_ELEMENTS(y) - 1$.
- 1—Absolute error, equals $\max_i e_i$.
- 2—The error norm is $\max_i (e_i/w_i)$, where $w_i = \max(|y_i|, Floor)$.
- Default: $NORM = 0$.

R_K_V

If present and nonzero, uses the Runge-Kutta-Verner fifth-order and sixth-order method.

SCALE

Scalar value used as a measure of the scale of the problem, such as an approximation to the Jacobian along the trajectory. Default: $SCALE = 1$

TOLERANCE

Scalar value used to set the tolerance for error control. An attempt is made to control the norm of the local error such that the global error is proportional to $TOLERANCE$. Default: $TOLERANCE = 0.001$

Adams Gear (Default) Method Only**JACOBIAN**

Scalar string specifying a user-supplied function to evaluate the Jacobian matrix. This function takes three parameters, x , y , and $yprime$, where x and y are defined in the

description of the user-supplied function f of the Arguments section and y_{prime} is the array returned by the user-supplied function f . The return value of this function is a two-dimensional array containing the partial derivatives. Each derivative $\partial y'_i / \partial y_j$ is evaluated at the provided (x, y) values and is returned in array location (i, j) .

MAX_ORD

Defines the highest order formula of implicit Adams type or BDF type to use.
Default: value 12 for Adams formulas; value 5 for BDF formulas

METHOD

Chooses the class of integration methods:

- 1—Uses implicit Adams method.
- 2—Uses backward differentiation formula (BDF) methods.
- Default: METHOD = 2.

MITER

Chooses the method for solving the formula equations:

- 1—Uses function iteration or successive substitution.
- 2—Uses chord or modified Newton method and a user-supplied Jacobian matrix.
- 3—Same as 2 except Jacobian is approximated within the function by divided differences.
- Default: MITER = 3.

Adams Gear (Default) Method Only

N_JEVALS

Named variable into which the array containing the number of Jacobian function evaluations used at each value of t is stored. The values returned are nonzero only if the keyword JACOBIAN is also used.

Discussion

The IMSL_ODE function finds an approximation to the solution of a system of first-order differential equations of the form:

$$\frac{dy}{dt} = y' = f(t, y)$$

with given initial conditions for y at the starting value for t . The function attempts to keep the global error proportional to a user-specified tolerance. The proportionality depends on the differential equation and the range of integration.

The function returns a two-dimensional array with the (i, j) -th component containing the i -th approximate solution at the j -th time step. Thus, the returned matrix has dimension (N_ELEMENTS (y), N_ELEMENTS (t)). It is important to notice here that the initial values of the problem also are included in this two-dimensional matrix.

The code is based on using backward differentiation formulas not exceeding order five as outlined in Gear (1971) and implemented by Hindmarsh (1974). There is an optional use of the code that employs implicit Adams formulas. This use is intended for nonstiff problems with expensive functions $y' = f(t, y)$.

If the keyword R_K_V is set, the IMSL_ODE function uses the Runge-Kutta-Verner fifth-order and sixth-order method and is efficient for nonstiff systems where the evaluations of $f(t, y)$ are not expensive. The code is based on an algorithm designed by Hull et al. (1976) and Jackson et al. (1978) and uses Runge-Kutta formulas of order five and six developed by J.H. Verner.

Examples

Example 1

This is a mildly stiff example problem (F2) from the test set of Enright and Pryce (1987).

$$\begin{aligned} y'_0 &= -y_0 - y_0 y_1 + k_0 y_1 \\ y'_1 &= -k_1 y_1 + k_2 (1 - y_1) y_0 \\ y_0(0) &= 1 \\ y_1(0) &= 0 \\ k_0 &= 294. \\ k_1 &= 3. \\ k_2 &= 0.01020408 \end{aligned}$$

```
.RUN
; Define function f.
FUNCTION f, t, y
    RETURN, [-y(0) - y(0) * y(1) + 294. * y(1), $
            -3.*y(1) + 0.01020408*(1. - y(1)) * y(0)]
END
```

```

yp = IMSL_ODE([0, 120, 240], [1, 0], 'f')
; Call the IMSL_ODE code with the values of the independent
; variable at which a solution is desired and the initial
; conditions.
PM, yp, FORMAT = '(3f10.6)', $
    Title = '      y(0)      y(120)      y(240) '
; Output results.
    y(0)      y(120)      y(240)
    1.000000   0.514591   0.392082
    0.000000   0.001749   0.001333

```

Now solve the same problem but with a user supplied Jacobian.

```

.RUN
; Define function f.
FUNCTION f, t, y
    RETURN, [-y(0)-y(0)*y(1)+294.0*y(1), $
            -3.0*y(1)+0.01020408*(1.0-y(1))*y(0)]
END

.RUN
FUNCTION jacob, x, y, dydx
    dydx = [ [-y(1)-1, 0.01020408*(1-y(1))], $
            [294-y(0), -0.01020408*y(0)-3] ]
    RETURN, dydx
END

yp = IMSL_ODE( [0,120,240], [1,0], 'f', JACOBIAN='jacob', MITER=2)
PM, yp, FORMAT='(3f10.6)', TITLE='      y(0)      y(120)      y(240) '

```

Example 2: Runge-Kutta Method

This example solves:

$$\frac{dy}{dt} = -y$$

over the interval $[0, 1]$ with the initial condition $y(0) = 1$ using the Runge-Kutta-Verner fifth-order and sixth-order method. The solution is $y(t) = e^{-t}$.

```

.RUN
; Define function f.
FUNCTION f, t, y
    RETURN, -y
END

yp = IMSL_ODE([0, 1], [1], 'f', /R_K_V)
; Call IMSL_ODE with the keyword R_K_V set.

```

```
PM, yp, Title = 'Solution'
; Output results.
Solution
      1.00000      0.367879
PM, yp(1) - EXP(-1), Title = 'Error'
Error
      0.00000
```

Example 3: Predator-Prey Problem

Consider a predator-prey problem with rabbits and foxes. Let r be the density of rabbits, and let f be the density of foxes. In the absence of any predator-prey interaction, the rabbits would increase at a rate proportional to their number, and the foxes would die of starvation at a rate proportional to their number. Mathematically, the model without species interaction is approximated by the following equations:

$$r' = 2r$$

$$f' = -f$$

With species interaction, the rate at which the rabbits are consumed by the foxes is assumed to equal the value $2rf$. The rate at which the foxes increase because they are consuming the rabbits, is equal to rf . Thus, the model differential equations to be solved are as follows:

$$r' = 2r - 2rf$$

$$f' = -f + rf$$

For illustration, the initial conditions are taken to be $r(0) = 1$ and $f(0) = 3$. The interval of integration is $0 \leq t \leq 40$. In the program, $y(0) = r$ and $y(1) = f$. The IMSL_ODE function is then called with 100 time values from 0 to 40. The results are shown in [Figure 8-1](#).

```
.RUN
; Define the function f.
FUNCTION f, t, y
    yp = y
    yp(0) = 2 * y(0) * (1 - y(1))
    yp(1) = -y(1) * (1 - y(0))
    RETURN, yp
END

y = [1, 3]
; Set the initial values and time values.
t = 40 * FINDGEN(100)/99
y = IMSL_ODE(t, y, 'f', /R_K_V)
; Call IMSL_ODE with R_K_V set to use the Runge-Kutta method.
PLOT, y(0, *), y(1, *), Psym = 2, XTitle = 'Density of Rabbits', $
    YTitle = 'Density of Foxes'
; Plot the result.
```

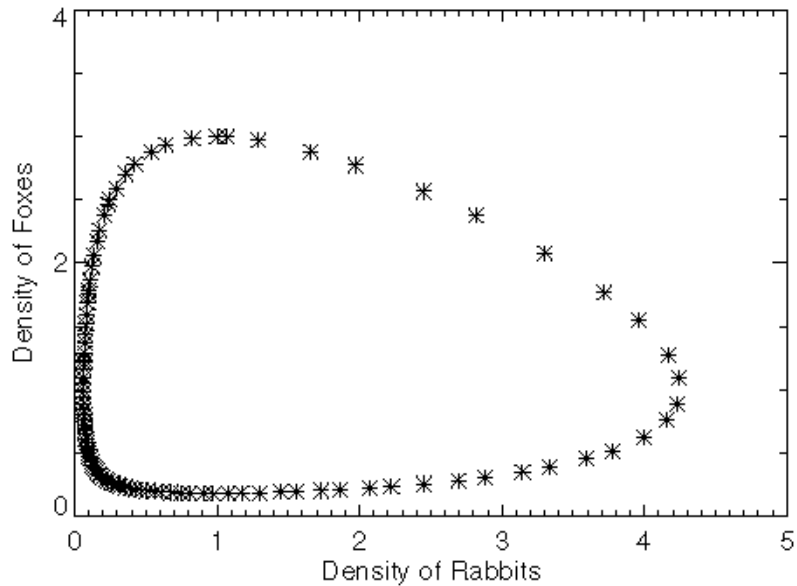



Figure 8-1: Predator-Prey Plot

Example 4: Stiff Problems and Changing Defaults

This problem is a stiff example (F5) from the test set of Enright and Pryce (1987). An initial step size of $h = 10^{-7}$ is suggested by these authors. When solving a problem that is known to be stiff, using double precision is advisable. The IMSL_ODE function is forced to use the suggested initial step size and double precision by using keywords.

$$y'_0 = k_0 (-k_1 y_0 y_1 + k_2 y_3 - k_3 y_0 y_2)$$

$$y'_1 = -k_0 k_1 y_0 y_1 + k_4 y_3$$

$$y'_2 = k_0 (-k_3 y_0 y_2 + k_5 y_3)$$

$$y'_3 = k_0 (k_1 y_0 y_1 - k_2 y_3 + k_3 y_0 y_2)$$

$$y_0(0) = 3.365 \times 10^{-7}$$

$$y_1(0) = 8.261 \times 10^{-3}$$

$$y_2(0) = 1.641 \times 10^{-3}$$

$$y_3(0) = 9.380 \times 10^{-6}$$

$$k_0 = 10^{11}$$

$$k_1 = 3.$$

$$k_2 = 0.0012$$

$$k_3 = 9.$$

$$k_4 = 2 \times 10^7$$

$$k_5 = 0.001$$

The results are shown in [Figure 8-2](#).

```
.RUN
; Define the function.
FUNCTION f, t, y
    k = [1d11, 3., .0012, 9., 2d7, .001]
    yp = [k(0)*(-k(1)*y(0)*y(1)+k(2)*y(3)- $
        k(3)*y(0)*y(2)), -k(0)*k(1)*y(0)*y(1)+ $
        k(4)*y(3), k(0)*(-k(3)*y(0)*y(2) + $
        k(5)*y(3)), k(0)* (k(1)*y(0)*y(1)- $
        k(2)*y(3)+k(3)*y(0)*y(2))]
    RETURN, yp
END

t = FINDGEN(500)/5e6
; Set up the values of the independent variable.
y = [3.365e-7, 8.261e-3, 1.641e-3, 9.380e-6]
; Set the initial values.
y = IMSL_ODE(t, y, 'f', Hinit = 1d-7, /Double)
; Call IMSL_ODE.
!P.Multi = [0, 2, 2]
!P.Font = 0
PLOT, t, y(0, *), Title = '!8y!I0!5', XTICKS=2
PLOT, t, y(1, *), Title = '!8y!I1!5', XTICKS=2
PLOT, t, y(2, *), Title = '!8y!I2!5', XTICKS=2
PLOT, t, y(3, *), Title = '!8y!I3!5', XTICKS=2
; Plot each variable on a separate axis.
```

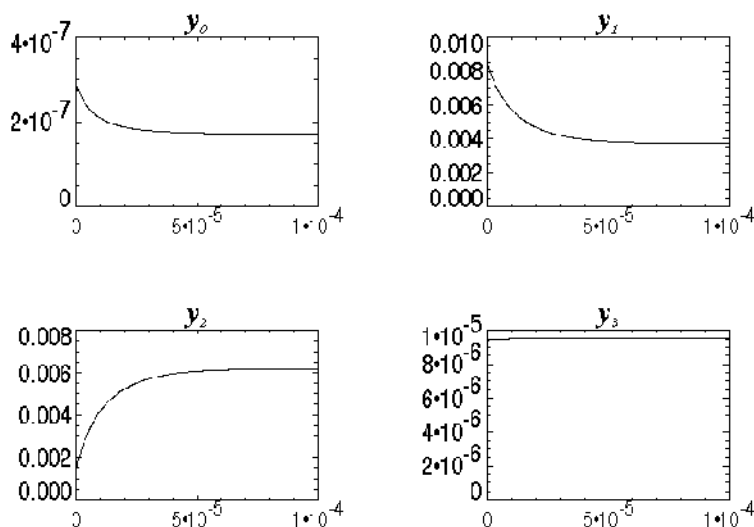


Figure 8-2: Plot for Each Variable

Example 5: Strange Attractors—The Rossler System

This example illustrates a strange attractor. The strange attractor used is the Rossler system, a simple model of a truncated Navier-Stokes equation. The Rossler system is given by relation below.

$$y'_0 = -y_1 - y_2$$

$$y'_1 = y_0 + a y_1$$

$$y'_2 = b + y_0 y_2 - c y_2$$

The initial conditions and constants are shown below.

$$y_0(0) = 1$$

$$y_1(0) = 0$$

$$y_2(0) = 0$$

$$a = 0.2$$

$$b = 0.2$$

$$c = 5.7$$

The results are shown in [Figure 8-3](#).

```
.RUN
; Define function f.
FUNCTION f, t, y
    COMMON constants, a, b, c
    ; Define some common variables.
    yp = y
    yp(0) = -y(1) - y(2)
    yp(1) = y(0) + a * y(1)
    yp(2) = b + y(0) * y(2) - c * y(2)
    RETURN, yp
END

COMMON constants, a, b, c
    a = .2
    b = .2
    c = 5.7
; Assign values to the common variables.
ntime = 5000
; Set the number of values of the independent variable.
time_range = 200
; Set the range of the independent variable to 0, ..., 200.
max_steps = 20000
; Allow up to 20,000 steps per value of the independent variable.
t = FINDGEN(ntime)/(ntime - 1) * time_range
y = [1, 0, 0]
; Set the initial conditions.
y = IMSL_ODE(t, y, 'f', Max_Steps = max_steps, /Double)
; Call IMSL_ODE using keywords Max_Steps and Double.
!P.Charsize = 1.5
SURFACE, FINDGEN(2, 2), /Nodata, $
XRange = [MIN(y(0, *)), MAX(y(0, *))], $
YRange = [MIN(y(1, *)), MAX(y(1, *))], $
ZRange = [MIN(y(2, *)), MAX(y(2, *))], $
XTitle = '!6y!i0', YTitle = 'y!i1', $
ZTitle = 'y!i2', Az = 25, /Save
PLOTS, y(0, *), y(1, *), y(2, *), /T3d
; Set up axes to plot solution. SURFACE draws the axes and defines
; the transformation used in PLOTS. The transformation is saved
; using keyword Save in SURFACE, then applied in PLOTS using T3d.
```

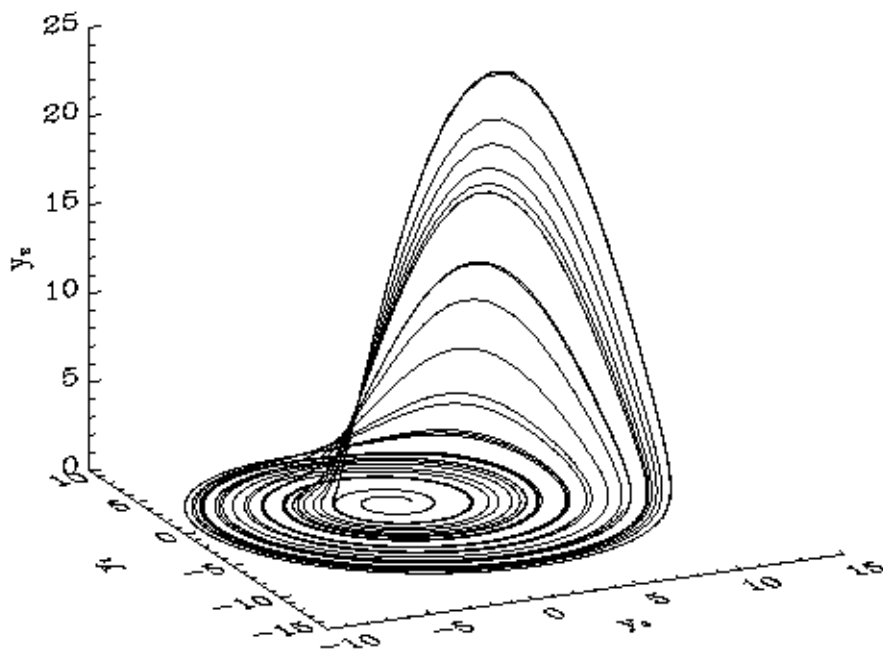


Figure 8-3: Rossler System Plot

Example 6: Coupled, Second-order System

Consider the two-degrees-of-freedom system represented by the model (and corresponding free-body diagrams) in [Figure 8-4](#). Assuming y_1 is greater than y_0 causes the spring k_1 to be in tension, as seen by the tensile force $k_1 (y_1 - y_0)$.

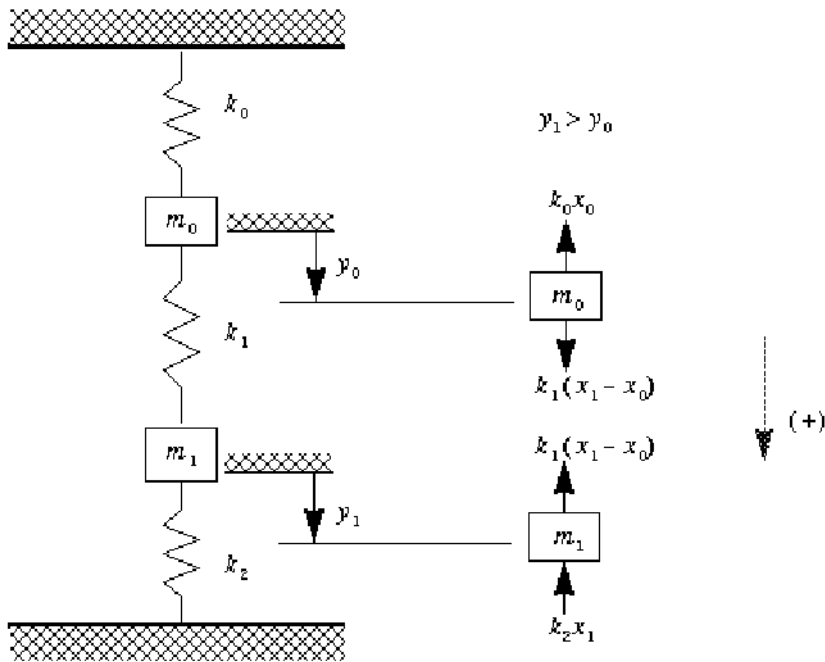


Figure 8-4: Two-Degrees of Freedom System

Note

If y_0 is taken to be greater than y_1 , then spring k_1 is in compression, with the spring force $k_1(y_0 - y_1)$. Both methods give correct results when a summation of forces is written.

The differential equations of motion for the system are written as follows:

$$m_0 \ddot{y}_0 = -k_0 y_0 + k_1 (y_1 - y_0)$$

$$m_1 \ddot{y}_1 = -k_1 (y_1 - y_0) - k_2 y_1$$

Thus:

$$\ddot{y}_0 = -\left(\frac{k_0 + k_1}{m_0}\right)y_0 + \left(\frac{k_1}{m_0}\right)y_1$$

$$\ddot{y}_1 = \left(\frac{k_1}{m_1}\right)y_0 - \left(\frac{k_1 + k_2}{m_1}\right)y_1$$

If given the mass and spring constant values:

$$m_0 = m_1 = 1 \text{ kg}$$

$$k_0 = k_1 = k_2 = 1000 \frac{\text{N}}{\text{m}}$$

the following is true:

$$\ddot{y}_0 = (-2000)y_0 + (1000)y_1$$

$$\ddot{y}_1 = (1000)y_0 - 2000y_1$$

Now, in order to convert this problem into one which IMSL_ODE can be used to solve, choose the following variables:

$$z(0) = y_0$$

$$z(1) = y_1$$

$$z(2) = \dot{y}_0$$

$$z(3) = \dot{y}_1$$

$$k(0) = -2000$$

$$k(1) = 1000$$

which yields the following equations:

$$\dot{y}_0 = z(2)$$

$$\dot{y}_1 = z(3)$$

$$\ddot{y}_0 = k(0)z(0) + k(1)z(1)$$

$$\ddot{y}_1 = k(1)z(0) + k(0)z(1)$$

The last four equations are the object of the return values of the user-supplied function in the exact order as previously specified.

The example below loops through four different sets of initial values for z . The results are shown in [Figure 8-5](#).

```
.RUN
; Define a function.
FUNCTION f, t, z
    k = [-2000, 1000]
    RETURN, [z(2), z(3), k(0) * z(0) + k(1) * $
            z(1), k(1) * z(0) + k(0) * z(1)]
END

.RUN
t = FINDGEN(1000)/999
; Independent variable, t, is between 0 and 1.
!P.MULTI = [0, 2, 2]
; Place all four plots in one window.
FOR i = 0, 3 DO BEGIN
    z = [1, i/3., 0, 0]
    z = IMSL_ODE(t, z, 'f', Max_Steps = 1000, Hinit = 0.001, /R_K_V)
    PLOT, t, z(0, *), Thick = 2, Title = 'Displacement of Mass'
    ; Plot the displacement of m0 as a solid line.
    OPLOT, t, z(1, *), Linestyle = 1, Thick = 2
    ; Overplot the displacement of m1 as a dotted line.
ENDFOR
END
```

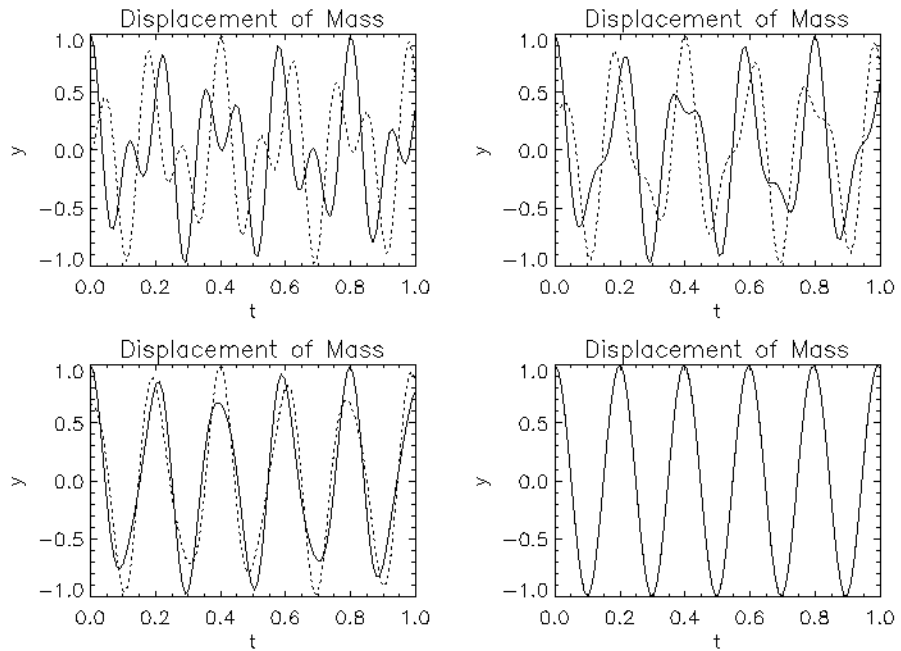



Figure 8-5: Second Order Systems with Differential Initial Values

The displacement for m_0 is the solid line, and the dotted line represents the displacement for m_1 . Note that when the initial conditions for:

$$\dot{y}_0 \text{ and } \dot{y}_1$$

are equal, the displacement of the masses is equal for all values of the independent variable (as seen in the fourth plot). Also, the two principal modes of this problem occur when the following is true:

$$\dot{y}_0 = \dot{y}_1 = 1$$

$$\ddot{y}_0 = 1, \ddot{y}_1 = 1$$

Errors

Fatal Errors

- MATH_ODE_TOO_MANY_EVALS—Completion of the next step would make the number of function evaluations #, but only # evaluations are allowed.
- MATH_ODE_TOO_MANY_STEPS—Maximum number of steps allowed; # used. The problem may be stiff.
- MATH_ODE_FAIL—Unable to satisfy the error requirement. *TOLERANCE* = # may be too small.

Version History

6.4	Introduced

IMSL_PDE_MOL

The IMSL_PDE_MOL function solves a system of partial differential equations of the form $u_t = f(x, t, u, u_x, u_{xx})$ using the method of lines. The solution is represented with cubic Hermite polynomials.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_PDE_MOL(t, y, xbreak, f_ut, f_bc [, /DOUBLE]
                    [, DERIV_INIT=array] [, HINIT=value] [, TOLERANCE=value] )
```

Return Value

Three-dimensional array of size $npde$ by nx by $N\_ELEMENTS(t)$ containing the approximate solutions for each specified value of the independent variable.

Arguments

t

One-dimensional array containing values of independent variable. Element $t(0)$ should contain the initial independent variable value (the initial time, t_0) and the remaining elements of t should be values of the independent variable at which a solution is desired.

y

Two-dimensional array of size $npde$ by nx containing the initial values, where $npde$ is the number of differential equations and nx is the number of mesh points or lines. It must satisfy the boundary conditions.

xbreak

One-dimensional array of length nx containing the breakpoints for the cubic Hermite splines used in the x discretization. The points in $xbreak$ must be strictly increasing. The values $xbreak(0)$ and $xbreak(nx - 1)$ are the endpoints of the interval.

f_ut

Scalar string specifying an user-supplied function to evaluate u_t . Function f_ut accepts the following arguments:

- **npde**—Number of equations.
- **x**—Space variable, x .
- **t**—Time variable, t .
- **u**—One-dimensional array of length $npde$ containing the dependent values, u .
- **ux**—One-dimensional array of length $npde$ containing the first derivatives, u_x .
- **uxx**—One-dimensional array of length $npde$ containing the second derivative, u_{xx} .

The return value of this function is an one-dimensional array of length $npde$ containing the computed derivatives u_t .

f_bc—Scalar string specifying user-supplied procedure to evaluate boundary conditions. The boundary conditions accepted by IMSL_PDE_MOL are:

$$\alpha_k u_k + \beta_k \frac{\partial u_k}{\partial x} = \gamma_k$$

Note

Users must supply the values α_k and β_k , which determine the values γ_k . Since γ_k can depend on t values, γ_k' also are required.

- **npde**—Number of equations. (Input)
- **x**—Space variable, x . (Input)
- **t**—Time variable, t . (Input)
- **alpha**—Named variable into which an one-dimensional array of length $npde$ containing the α_k values is stored. (Output)
- **beta**—Named variable into which an one-dimensional array of length $npde$ containing the β_k values is stored. (Output)

gammap

Named variable into which an one-dimensional array of length $npde$ containing the derivatives is stored. (Output):

$$\frac{d\gamma_k}{dt} = \gamma'_k$$

Keywords**DOUBLE**

If present and nonzero, double precision is used.

DERIV_INIT

Two-dimensional array that supplies the derivative values $u_x(x, t(0))$. This derivative information is input as:

$$\text{Deriv\_Init}(k, i) = \frac{\partial u_k}{\partial u_x}(x, t(0))$$

Default: Derivatives are computed using cubic spline interpolation

HINIT

Initial step size in the t integration. This value must be nonnegative. If HINIT is zero, an initial step size of $0.001|t_{i+1} - t_i|$ will be arbitrarily used. The step will be applied in the direction of integration. Default: HINIT = 0.0

TOLERANCE

Differential equation error tolerance. An attempt to control the local error in such a way that the global relative error is proportional to TOLERANCE. Default: TOLERANCE = $100.0 \cdot \epsilon$, where ϵ is machine epsilon.

Discussion

Let $M = npde$, $N = nx$ and $x_i = xbreak(i)$. The routine `IMSL_PDE_MOL` uses the method of lines to solve the partial differential equation system:

$$\frac{\partial u_k}{\partial t} = f_k \left(x, t, u_1, \dots, u_M, \frac{\partial u_1}{\partial x}, \dots, \frac{\partial u_M}{\partial x}, \frac{\partial^2 u_1}{\partial x^2}, \dots, \frac{\partial^2 u_M}{\partial x^2} \right)$$

with the initial conditions:

$$u_k = u_k(x, t) \text{ at } t = t_0, \text{ where } t_0 = t(0)$$

and the boundary conditions:

$$\alpha_k u_k + \beta_k \frac{\partial u_k}{\partial x} = \gamma_k \quad \text{at } x = x_1 \text{ and at } x = x_N$$

for $k = 1, \dots, M$.

Cubic Hermite polynomials are used in the x variable approximation so that the trial solution is expanded in the series:

$$\hat{u}_k(x, t) = \sum_{i=1}^N (a_{i,k}(t)\phi_i(x) + b_{i,k}(t)\psi_i(x))$$

where $\phi_i(x)$ and $\psi_i(x)$ are standard basis functions for cubic Hermite polynomials with the knots $x_1 < x_2 < \dots < x_N$. These are piecewise cubic polynomials with continuous first derivatives. At the breakpoints, they satisfy:

$$\begin{aligned} \phi_i(x_1) &= \delta_{i1} & \psi_i(x_1) &= 0 \\ \frac{d\phi_i}{dx}(x_1) &= 0 & \frac{d\psi_i}{dx}(x_1) &= \delta_{i1} \end{aligned}$$

According to the collocation method, the coefficients of the approximation are obtained so that the trial solution satisfies the differential equation at the two Gaussian points in each subinterval:

$$\begin{aligned} p_{2j-1} &= x_j + \frac{3-\sqrt{3}}{6}(x_{j+1} - x_j) \\ p_{2j} &= x_j + \frac{3+\sqrt{3}}{6}(x_{j+1} + x_j) \end{aligned}$$

for $j = 1, \dots, N$. The collocation approximation to the differential equation is:

$$\frac{da_{i,k}}{dt} \phi_i(p_j) + \frac{db_{i,k}}{dt} \psi_i(p_j) = f_k(p_j, t, \hat{u}_1(p_j), \dots, \hat{u}_M(p_j), \dots, (\hat{u}_1)_{xx}(p_j), \dots, (\hat{u}_M)_{xx}(p_j))$$

for $k = 1, \dots, M$ and $j = 1, \dots, 2(N-1)$.

This is a system of $2M(N-1)$ ordinary differential equations in $2M N$ unknown coefficient functions, $a_{i,k}$ and $b_{i,k}$. This system can be written in the matrix-vector form as $A dc/dt = F(t, y)$ with $c(t_0) = c_0$ where c is a vector of coefficients of length $2M N$ and c_0 holds the initial values of the coefficients. The last $2M$ equations are obtained by differentiating the boundary conditions:

$$\alpha_k \cdot \frac{da_k}{dt} + \beta_k \cdot \frac{db_k}{dt} = \frac{d\gamma_k}{dt}$$

for $k = 1, \dots, M$.

The initial conditions $u_k(x, t_0)$ must satisfy the boundary conditions. Also, the $\gamma_k(t)$ must be continuous and have a smooth derivative, or the boundary conditions will not be properly imposed for $t > t_0$.

If $\alpha_k = \beta_k = 0$, it is assumed that no boundary condition is desired for the k -th unknown at the left endpoint. A similar comment holds for the right endpoint. Thus, collocation is done at the endpoint. This is generally a useful feature for systems of first-order partial differential equations.

If the number of partial differential equations is $M = 1$ and the number of breakpoints is $N = 4$, then:

$$A = \begin{bmatrix} \alpha_1 & \beta_1 & & & & & & \\ \phi_1(p_1) & \psi_1(p_1) & \phi_2(p_1) & \psi_2(p_1) & & & & \\ \phi_1(p_2) & \psi_1(p_2) & \phi_2(p_2) & \psi_2(p_2) & & & & \\ & & \phi_3(p_3) & \psi_3(p_3) & \phi_4(p_3) & \psi_4(p_3) & & \\ & & \phi_3(p_4) & \psi_3(p_4) & \phi_4(p_4) & \psi_4(p_4) & & \\ & & & & \phi_5(p_5) & \psi_5(p_5) & \phi_6(p_5) & \psi_6(p_5) \\ & & & & \phi_5(p_6) & \psi_5(p_6) & \phi_6(p_6) & \psi_6(p_6) \\ & & & & & & \alpha_4 & \beta_4 \end{bmatrix}$$

The vector c is:

$$c = [a_1, b_1, a_2, b_2, a_3, b_3, a_4, b_4]^T$$

and the right-side F is:

$$F = [\gamma_1(x_1), f(p_1), f(p_2), f(p_3), f(p_4), f(p_5), f(p_6), \gamma_1(x_1)]^T$$

If $M > 1$, then each entry in the above matrix is replaced by an $M \times M$ diagonal matrix. The element α_1 is replaced by $\text{diag}(\alpha_{1,1}, \dots, \alpha_{1,M})$. The elements α_N , β_1 and β_N are handled in the same manner. The $\phi_i(p_j)$ and $\psi_i(p_j)$ elements are replaced by $\phi_i(p_j)I_M$ and $\psi_i(p_j)I_M$ where I_M is the identity matrix of order M . See Madsen and Sincovec (1979) for further details about discretization errors and Jacobian matrix structure.

The input array y contains the values of the $a_{k,i}$. The initial values of the $b_{k,i}$ are obtained by using the IDL Advanced Math and Stats cubic spline routine [IMSL_CSINTERP](#) to construct functions:

$$\hat{u}_k(x, t_0)$$

such that:

$$\hat{u}_k(x_i, t_0) = a_{ki}$$

The IDL Advanced Math and Stats routine [IMSL_SPVALUE](#) is used to approximate the values:

$$\frac{d\hat{u}_k}{dx}(x_i, t_0) \equiv b_{k,i}$$

There is an optional use of [IMSL_PDE_MOL](#) that allows the user to provide the initial values of $b_{k,i}$.

The order of matrix A is $2MN$ and its maximum bandwidth is $6M - 1$. The band structure of the Jacobian of F with respect to c is the same as the band structure of A . This system is solved using a modified version of [IMSL_ODE](#). Some of the linear solvers were removed. Numerical Jacobians are used exclusively. The algorithm is unchanged. Gear's BDF method is used as the default because the system is typically stiff.

Four examples of PDEs are now presented that illustrate how users can interface their problems with [IMSL_PDE_MOL](#). The examples are small and not indicative of the complexities that most practitioners will face in their applications.

Examples

Example 1

This equation solves the normalized linear diffusion PDE, $u_t = u_{xx}$, $0 \leq x \leq 1, t > t_0$. The initial values are $t_0 = 0$, $u(x, t_0) = u_0 = 1$. There is a “zero-flux” boundary condition at $x = 1$, namely $u_x(1, t) = 0, (t > t_0)$. The boundary value of $u(0, t)$ is abruptly changed from u_0 to the value $u_1 = 0.1$. This transition is completed by $t = t_\delta = 0.09$.

Due to restrictions in the type of boundary conditions successfully processed by `IMSL_PDE_MOL`, it is necessary to provide the derivative boundary value function γ at $x = 0$ and at $x = 1$. The function γ at $x = 0$ makes a smooth transition from the value u_0 at $t = t_0$ to the value u_1 at $t = t_\delta$. The transition phase for γ is computed by evaluating a cubic interpolating polynomial. For this purpose, the function subprogram `IMSL_SPVALUE`. The interpolation is performed as a first step in the user-supplied procedure `f_bc`. The function and derivative values $\gamma(t_0) = u_0$, $\gamma'(t_0) = 0$, $\gamma(t_\delta) = u_1$, and $\gamma'(t_\delta) = 0$, are used as input to routine `IMSL_CSINTERP` to obtain the coefficients evaluated by `IMSL_SPVALUE`. Notice that $\gamma'(t) = 0, t > t_\delta$. The evaluation routine `IMSL_SPVALUE` will not yield this value so logic in the procedure `f_bc` assigns $\gamma'(t) = 0, t > t_\delta$.

Save the following code as `pde_mol_example1`, then compile and run:

```
FUNCTION f_ut, npde, x, t, u, ux, uxx
; Define the PDE
  ut = uxx
  RETURN, ut
END

PRO f_bc, npde, x, t, alpha, beta, gammap
COMMON ex1_pde, first, ppoly
  first = 1
  alpha = FLTARR(npde)
  beta = FLTARR(npde)
  gammap = FLTARR(npde)
  delta = 0.09
; Compute interpolant first time only
IF (first EQ 1) THEN BEGIN
  first = 0
  ppoly = IMSL_CSINTERP([0.0, delta], [1.0, 0.1], $
    ileft = 1, left = 0.0,  iright = 1, right = 0.0)
ENDIF
; Define the boundary conditions.
IF (x EQ 0.0) THEN BEGIN
  alpha(0) = 1.0
  beta(0) = 0.0
```

```

        gammap(0) = 0.0
        ; If in the boundary layer, compute nonzero gamma prime
        IF (t LE delta) THEN gammap(0) = $
            IMSL_SPVALUE(t, ppoly, xderiv = 1)
        END ELSE BEGIN
            ; These are for x = 1
            alpha(0) = 0.0
            beta(0) = 1.0
            gammap(0) = 0.0
        END
    RETURN
END

PRO pde_mol_example1
    COMMON ex1_pde, first, ppoly
    npde = 1
    nx = 8
    nstep = 10
    ; Set breakpoints and initial conditions
    xbreak = FINDGEN(nx)/(nx - 1)
    y = FLTARR(npde, nx)
    y(*) = 1.0
    ; Initialize the solver
    t = FINDGEN(nstep)/(nstep) + 0.1
    t = [0.0, t*t]
    ; Solve the problem
    res = IMSL_PDE_MOL(t, y, xbreak, 'f_ut', 'f_bc')
    num = INDGEN(8) + 1
    FOR i = 1, 10 DO BEGIN
        PRINT, 'solution at t = ', t(i)
        PRINT, num, FORMAT = '(8I7)'
        PM, res(0, *, i), FORMAT = '(8F7.4)'
    ENDFOR
END

```

IDL Prints:

```

solution at t =      0.0100000
      1      2      3      4      5      6      7      8
0.9691 0.9972 0.9999 1.0000 1.0000 1.0000 1.0000 1.0000
solution at t =      0.0400000
      1      2      3      4      5      6      7      8
0.6247 0.8708 0.9624 0.9908 0.9981 0.9997 1.0000 1.0000
solution at t =      0.0900000
      1      2      3      4      5      6      7      8
0.1000 0.4602 0.7169 0.8671 0.9436 0.9781 0.9917 0.9951
solution at t =      0.160000
      1      2      3      4      5      6      7      8
0.1000 0.3130 0.5071 0.6681 0.7893 0.8708 0.9168 0.9315
solution at t =      0.250000

```

```

      1      2      3      4      5      6      7      8
0.1000 0.2567 0.4045 0.5354 0.6428 0.7224 0.7710 0.7874
solution at t =      0.360000
      1      2      3      4      5      6      7      8
0.1000 0.2176 0.3292 0.4292 0.5125 0.5751 0.6139 0.6270
solution at t =      0.490000
      1      2      3      4      5      6      7      8
0.1000 0.1852 0.2661 0.3386 0.3992 0.4448 0.4731 0.4827
solution at t =      0.640000
      1      2      3      4      5      6      7      8
0.1000 0.1588 0.2147 0.2648 0.3066 0.3381 0.3577 0.3643
solution at t =      0.810000
      1      2      3      4      5      6      7      8
0.1000 0.1387 0.1754 0.2083 0.2358 0.2565 0.2694 0.2738
solution at t =      1.000000
      1      2      3      4      5      6      7      8
0.1000 0.1242 0.1472 0.1678 0.1850 0.1980 0.2060 0.2087

```

Example 2

This example solves Problem C from Sincovec and Madsen (1975). The equation is of diffusion-convection type with discontinuous coefficients. This problem illustrates a simple method for programming the evaluation routine for the derivative, u_t . Note that the weak discontinuities at $x = 0.5$ are not evaluated in the expression for u_t . The results are shown in [Figure 8-6](#). The problem is defined as:

$$u_t = \partial u / \partial t = \partial / \partial x (D(x) \partial u / \partial x) - v(x) \partial u / \partial x$$

$$x \in [0, 1], t > 0$$

$$D(x) = \begin{cases} 5 & \text{if } 0 \leq x < 0.5 \\ 1 & \text{if } 0.5 < x \leq 1.0 \end{cases}$$

$$v(x) = \begin{cases} 1000.0 & \text{if } 0 \leq x < 0.5 \\ 1 & \text{if } 0.5 < x \leq 1.0 \end{cases}$$

$$u(x, 0) = \begin{cases} 1 & \text{if } x = 0 \\ 0 & \text{if } x < 0 \end{cases}$$

$$u(0, t) = 1, \quad u(1, t) = 0$$

Save the following code as `pde_mol_example2`, then compile and run:

```

FUNCTION f_ut, npde, x, t, u, ux, uxx
; Define the PDE

```

```

    ut = FLTARR(npde)
    IF (x LE 0.5) THEN BEGIN
        d = 5.0
        v = 1000.0
    END ELSE BEGIN
        d = 1.0
        v = 1.0
    END
    ut(0) = d*uxx(0) - v*ux(0)
    RETURN, ut
END

PRO f_bc, npde, x, t, alpha, beta, gammap
; Define the Boundary Conditions
alpha = FLTARR(npde)
beta = FLTARR(npde)
gammap = FLTARR(npde)
alpha(0) = 1.0
beta(0) = 0.0
gammap(0) = 0.0
RETURN
END

PRO pde_mol_example2
    npde = 1
    nx = 100
    nstep = 10
; Set breakpoints and initial conditions
    xbreak = FINDGEN(nx)/(nx - 1)
    y = FLTARR(npde, 100)
    y(*) = 0.0
    y(0) = 1.0
; Initialize the solver
    mach = IMSL_MACHINE(/FLOAT)
    tol = SQRT(mach.MAX_REL_SPACE)
    hinit = 0.01*tol
    PRINT, 'tol = ', tol, ' and hinit = ', hinit
    t = [0.0, FINDGEN(nstep)/(nstep) + 0.1]
; Solve the problem
    res = IMSL_PDE_MOL(t, y, xbreak, 'f_ut', 'f_bc', $
        tolerance = tol, hinit = hinit)
; Plot results at current  $t_i=t_{i+1}$ 
    PLOT, xbreak, res(0,*,10), psym = 3, yrange=[0, 1.25], $
        title = 'Solution at t = 1.0'
END

```

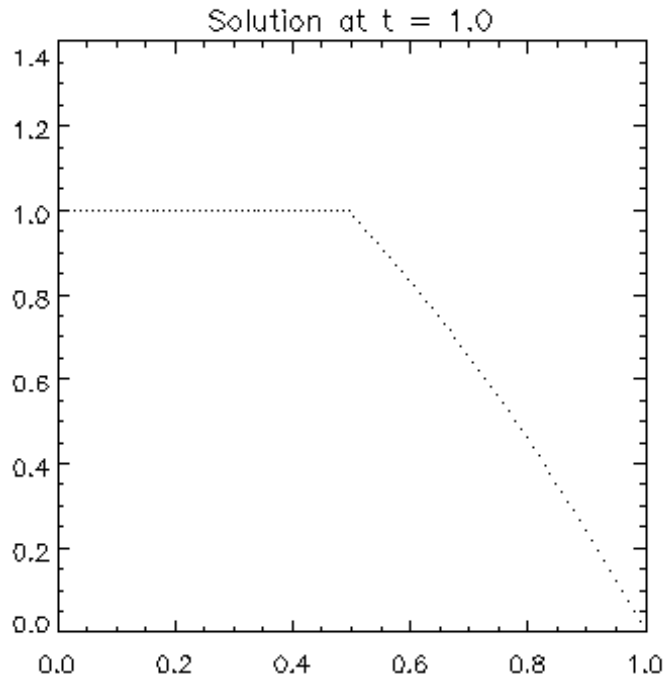


Figure 8-6: Diffusion-Convection Type with Discontinuous Coefficients

Example 3

In this example, using `IMSL_PDE_MOL`, the linear normalized diffusion PDE $u_t = u_{xx}$ is solved but with an optional use that provides values of the derivatives, u_x , of the initial data. Due to errors in the numerical derivatives computed by spline interpolation, more precise derivative values are required when the initial data is $u(x, 0) = 1 + \cos[(2n - 1)\pi x]$, $n > 1$. The boundary conditions are “zero flux” conditions $u_x(0, t) = u_x(1, t) = 0$ for $t > 0$. Note that the initial data is compatible with these end conditions since the derivative function:

$$u_x(x, 0) = \frac{du(x, 0)}{dx} = -(2n - 1)\pi \sin[(2n - 1)\pi x]$$

vanishes at $x = 0$ and $x = 1$.

This optional usage signals that the derivative of the initial data is passed by the user.

Save the following code as `pde_mol_example3`, then compile and run:

```

FUNCTION f_ut, npde, x, t, u, ux, uxx
; Define the PDE
ut = FLTARR(npde)
ut(0) = uxx(0)
RETURN, ut
END

PRO f_bc, npde, x, t, alpha, beta, gammap
; Define the boundary conditions
alpha = FLTARR(npde)
beta = FLTARR(npde)
gammap = FLTARR(npde)
alpha(0) = 0.0
beta(0) = 1.0
gammap(0) = 0.0
RETURN
END

PRO pde_mol_example3
npde = 1
nx = 10
nstep = 10
arg = 9.0*!Pi
; Set breakpoints and initial conditions
xbreak = FINDGEN(nx)/(nx - 1)
y = FLTARR(npde, nx)
y(0, *) = 1.0 + COS(arg*xbreak)
di = y
di(0, *) = -arg*SIN(arg*xbreak)
; Initialize the solver
mach = IMSL_MACHINE(/FLOAT)
tol = SQRT(mach.MAX_REL_SPACE)
t = [FINDGEN(nstep + 1)*(nstep*0.001)/(nstep)]
; Solve the problem
res = IMSL_PDE_MOL(t, y, xbreak, 'f_ut', 'f_bc', $
    Tolerance = tol, Deriv_Init = di)
; Print results at every other  $t_i = t_{i+1}$ 
FOR i = 2, 10, 2 DO BEGIN
    PRINT, 'solution at t = ', t(i)
    PM, res(0, *, i), FORMAT = '(10F10.4)'
    PRINT, 'derivative at t = ', t(i)
    PM, di(0, *, i)
    PRINT
ENDFOR
END

```

IDL Prints:

```

solution at t =      0.00200000
    1.2329    0.7671    1.2329    0.7671    1.2329
    0.7671    1.2329    0.7671    1.2329    0.7671
derivative at t =      0.00200000
    0.00000    9.58505e-07    7.96148e-09    1.25302e-06
   -1.61002e-07    1.91968e-06   -1.60244e-06    3.85856e-06
   -4.83314e-06    2.02301e-06

solution at t =      0.00400000
    1.0537    0.9463    1.0537    0.9463    1.0537
    0.9463    1.0537    0.9463    1.0537    0.9463
derivative at t =      0.00400000
    0.00000    6.64098e-07   -5.12883e-07    8.55131e-07
   -6.11177e-07   -2.76893e-06    7.84288e-08    2.97113e-06
   -2.32777e-07    2.02301e-06

solution at t =      0.00600000
    1.0121    0.9879    1.0121    0.9879    1.0121
    0.9879    1.0121    0.9879    1.0121    0.9879
derivative at t =      0.00600000
    0.00000    7.42109e-07   -5.29244e-08   -1.98559e-07
   -1.19702e-06   -8.66795e-07    1.17180e-07    7.09625e-07
    4.31432e-07    2.02301e-06

solution at t =      0.00800000
    1.0027    0.9973    1.0027    0.9973    1.0027
    0.9973    1.0027    0.9973    1.0027    0.9973
derivative at t =      0.00800000
    0.00000    3.56892e-07   -3.80790e-07   -9.99308e-07
   -1.96765e-07    7.72356e-07    8.50576e-08    1.11979e-07
    4.74838e-07    2.02301e-06

solution at t =      0.0100000
    1.0008    0.9992    1.0008    0.9992    1.0008
    0.9992    1.0008    0.9992    1.0008    0.9992
derivative at t =      0.0100000
    0.00000    2.40533e-07   -4.27171e-07   -1.25933e-06
    3.60702e-08    6.42627e-07   -1.00818e-07    2.08207e-07
    1.12973e-06    2.02301e-06

```

Example 4

In this example, consider the linear normalized hyperbolic PDE, $u_{tt} = u_{xx}$, the “vibrating string” equation. This naturally leads to a system of first order PDEs. Define a new dependent variable $u_t = v$. Then, $v_t = u_{xx}$ is the second equation in the system. Take as initial data $u(x, 0) = \sin(\pi x)$ and $u_t(x, 0) = v(x, 0) = 0$. The ends of the string are fixed so $u(0, t) = u(1, t) = v(0, t) = v(1, t) = 0$. The exact solution to this problem is $u(x, t) = \sin(\pi x) \cos(\pi t)$. Residuals are computed at the output values of t for $0 < t \leq 2$. Output is obtained at 200 steps in increments of 0.01.

Even though the sample code `IMSL_PDE_MOL` gives satisfactory results for this PDE, users should be aware that for *nonlinear problems*, “shocks” can develop in the solution. The appearance of shocks may cause the code to fail in unpredictable ways. See Courant and Hilbert (1962), pp 488-490, for an introductory discussion of shocks in hyperbolic systems.

Save the following code as `pde_mol_example4`, then compile and run:

```

FUNCTION f_ut, npde, x, t, u, ux, uxx
; Define the PDE
ut = FLTARR(npde)
ut(0) = u(1)
ut(1) = uxx(0)
RETURN, ut
END

PRO f_bc, npde, x, t, alpha, beta, gammap
; Define the boundary conditions
alpha = FLTARR(npde)
beta = FLTARR(npde)
gammap = FLTARR(npde)
alpha(0) = 1
alpha(1) = 1
beta(0) = 0
beta(1) = 0
gammap(0) = 0
gammap(1) = 0
RETURN
END

PRO pde_mol_example4
npde = 2
nx = 10
nstep = 200
; Set breakpoints and initial conditions
xbreak = FINDGEN(nx)/(nx - 1)
y = FLTARR(npde, nx)
y(0, *) = SIN(!Pi*xbreak)
y(1, *) = 0
di = y
di(0, *) = !Pi*COS(!Pi*xbreak)
di(1, *) = 0.0
; Initialize the solver
mach = IMSL_MACHINE(/FLOAT)
tol = SQRT(mach.MAX_REL_SPACE)
t = [0.0, 0.01 + FINDGEN(nstep)*2.0/(nstep)]
; Solve the problem
u = IMSL_PDE_MOL(t, y, xbreak, 'f_ut', 'f_bc', $
Tolerance = tol, Deriv_Init = di)

```



```

err = 0.0
pde_error = FLTARR(nstep)
FOR j = 1, N_ELEMENTS(t) - 1 DO BEGIN
  FOR i = 0, nx - 1 DO BEGIN
    err = (err) > (u(0, i, j) - $
      SIN(!Pi*xbreak(i))*COS(!Pi*t(j)))
  ENDFOR
ENDFOR
PRINT, 'Maximum error in u(x, t) = ', err
END

```

IDL Prints:

```
Maximum error in u(x, t) = 0.000626385
```

Version History

6.4	Introduced

IMSL_POISSON2D

The IMSL_POISSON2D function solves Poisson's or Helmholtz's equation on a two-dimensional rectangle using a fast Poisson solver based on the HODIE finite-difference scheme on a uniform mesh.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_POISSON2D(rhs_pde, rhs_bc, coef_u, nx, ny, ax, bx, ay, by, bc_type  
[, /DOUBLE] [, ORDER=value])
```

Return Value

Two-dimensional array of size *nx* by *ny* containing solution at the grid points.

Arguments

rhs_pde

Scalar string specifying the name of the user-supplied function to evaluate the right-hand side of the partial differential equation at a scalar value *x* and scalar value *y*.

rhs_bc

Scalar string specifying the name of the user-supplied function to evaluate the right-hand side of the boundary conditions, on side *side*, at scalar value *x* and scalar value *y*. The value of *side* will be one of the integer values shown in [Table 8-1](#).

Integer	Side
0	right side
1	bottom side

Table 8-1: Integer Values

Integer	Side
2	left side
3	top side

*Table 8-1: Integer Values***coef_u**

Value of the coefficient of u in the differential equation.

nx

Number of grid lines in the x -direction. nx must be at least 4. See “[Discussion](#)” on page 369 section for further restrictions on nx .

ny

Number of grid lines in the y -direction. ny must be at least 4. See “[Discussion](#)” on page 369 section for further restrictions on ny .

ax

The value of x along the left side of the domain.

bx

The value of x along the right side of the domain.

ay

The value of y along the bottom of the domain.

by

The value of y along the top of the domain.

bc_type

One-dimensional array of size 4 indicating the type of boundary condition on each side of the domain or that the solution is periodic. The sides are numbered as shown in [Table 8-2](#).

Array	Side	Location
<i>bc_type</i> (0)	right	$x = bx$
<i>bc_type</i> (1)	bottom	$y = ay$
<i>bc_type</i> (2)	left	$x = ax$
<i>bc_type</i> (3)	top	$y = by$

Table 8-2: Side Numbering

The three possible boundary condition types are shown in [Table 8-3](#).

Type	Condition
<i>bc_type</i> (<i>i</i>) = 1	Dirichlet condition. Value of u is given.
<i>bc_type</i> (<i>i</i>) = 2	Neuman condition. Value of du/dx is given (on the right or left sides) or du/dy (on the bottom or top of the domain).
<i>bc_type</i> (<i>i</i>) = 3	Periodic condition.

Table 8-3: Boundary Condition Types

Keywords

DOUBLE

If present and nonzero, double precision is used.

ORDER

Order of accuracy of the finite-difference approximation. It can be either 2 or 4.
Default: ORDER = 4

Discussion

Let $c = \text{coef\_}u$, $a_x = ax$, $b_x = bx$, $a_y = ay$, $b_y = by$, $n_x = nx$ and $n_y = ny$.

IMSL_POISSON2D is based on the code HFFT2D by Boisvert (1984). It solves the equation:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + cu = p$$

on the rectangular domain $(a_x, b_x) \times (a_y, b_y)$ with a user-specified combination of Dirichlet (solution prescribed), Neumann (first-derivative prescribed), or periodic boundary conditions. The sides are numbered clockwise, starting with the right side, as shown in [Figure 8-7](#).

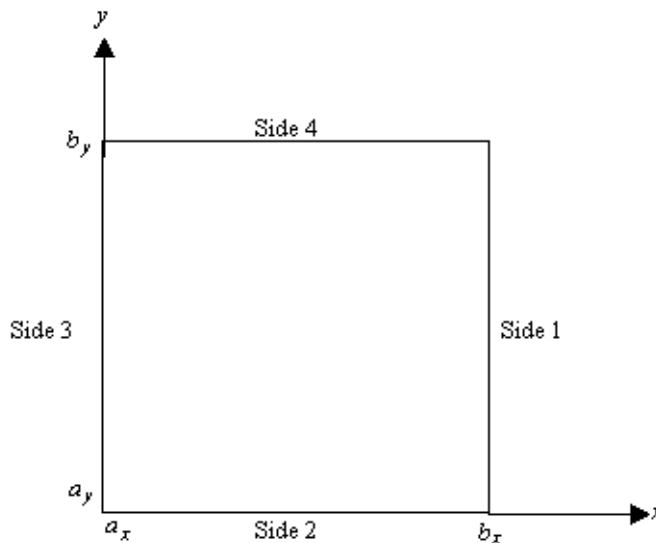


Figure 8-7: Side Numbering

When $c = 0$ and only Neumann or periodic boundary conditions are prescribed, then any constant may be added to the solution to obtain another solution to the problem. In this case, the solution of minimum ∞ -norm is returned.

The solution is computed using either a second- or fourth-order accurate finite-difference approximation of the continuous equation. The resulting system of linear

algebraic equations is solved using fast Fourier transform techniques. The algorithm relies on the fact that $n_x - 1$ is highly composite (the product of small primes). For details of the algorithm, see Boisvert (1984). If $n_x - 1$ is highly composite then the execution time of `IMSL_POISSON2D` is proportional to $n_x n_y \log_2 n_x$. If evaluations of $p(x, y)$ are inexpensive, then the difference in running time between `ORDER = 2` and `ORDER = 4` is small.

The grid spacing is the distance between the (uniformly spaced) grid lines. It is given by the formulas $hx = (bx - ax)/(nx - 1)$ and $hy = (by - ay)/(ny - 1)$. The grid spacings in the x and y directions must be the same, i.e., nx and ny must be such that hx is equal to hy . Also, as noted above, nx and ny must be at least 4. To increase the speed of the fast Fourier transform, $nx - 1$ should be the product of small primes. Good choices are 17, 33, and 65.

If `-coef_u` is nearly equal to an eigenvalue of the Laplacian with homogeneous boundary conditions, then the computed solution might have large errors.

Example

This example solves the equation:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + 3u = -2\sin(x + 2y) + 16e^{2x + 3y}$$

with the boundary conditions:

$$\frac{\partial u}{\partial y} = 2\cos(x + 2y) + 3e^{2x + 3y}$$

on the bottom side and:

$$u = \sin((x + 2y) + e^{2x + 3y})$$

on the other three sides. The domain is the rectangle $[0, 1/4] \times [0, 1/2]$. The output of `IMSL_POISSON2D` is a 17 x 33 table of values. The functions `IMSL_SPVALUE` are used to print a different table of values.

```
FUNCTION rhs_pde, x, y
; Define the right side of the PDE
f = (-2.0*SIN(x + 2.0*y) + 16.0*EXP(2.0*x + 3.0*y))
RETURN, f
END

FUNCTION rhs_bc, side, x, y
; Define the boundary conditions
IF (side EQ 1) THEN $
; Bottom side
```

```

      f = 2.0*COS(x + 2.0*y) + 3.0*EXP(2.0*x + 3.0*y) $
ELSE $
  ; All other sides, 0, 2, 3
  f = SIN(x + 2.0*y) + EXP(2.0*x + 3.0*y)
RETURN, f
END

PRO print_results, x, y, utable
  FOR j = 0, 4 DO FOR i = 0, 4 DO $
    PRINT, x(i), y(j), utable(i, j), $
    ABS(utable(i, j) - SIN(x(i) + 2.0*y(j)) - $
    EXP(2.0*x(i) + 3.0*y(j)))
  END
  nx = 17
  nxtable = 5
  ny = 33
  nytable = 5
  ; Set rectangle size
  ax = 0.0
  bx = 0.25
  ay = 0.0
  by = 0.5
  ; Set boundary conditions
  bc_type = [1, 2, 1, 1]
  ; Coefficient of u
  coef_u = 3.0
  ; Solve the PDE
  u = IMSL_POISSON2D('rhs_pde', 'rhs_bc', coef_u, nx, ny, ax, $
    bx, ay, by, bc_type)
  ; Set up for interpolation
  xdata = ax + (bx - ax)*FINDGEN(nx)/(nx - 1)
  ydata = ay + (by - ay)*FINDGEN(ny)/(ny - 1)
  ; Compute interpolant
  sp = IMSL_BSINTERP(xdata, ydata, u)
  x = ax + (bx - ax)*FINDGEN(nxtable)/(nxtable - 1)
  y = ay + (by - ay)*FINDGEN(nytable)/(nytable - 1)
  utable = IMSL_SPVALUE(x, y, sp)
  ; Print computed answer and absolute on nxtabl by nytabl grid
  PRINT, '          X          Y          U          Error'
  print_results, x, y, utable

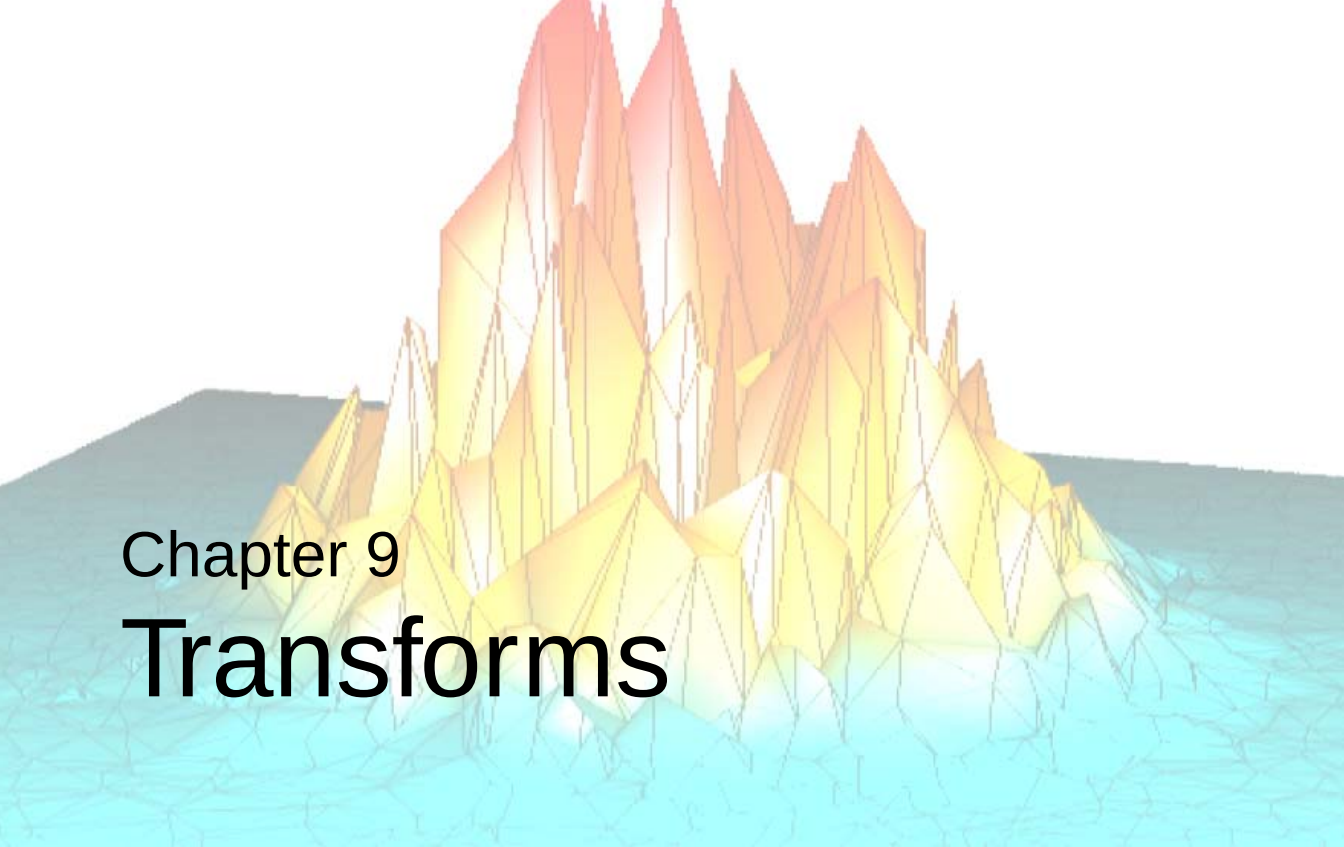
```

X	Y	U	Error
0.00000	0.00000	1.00000	0.00000
0.0625000	0.00000	1.19560	4.88758e-06
0.125000	0.00000	1.40869	7.39098e-06
0.187500	0.00000	1.64139	4.88758e-06
0.250000	0.00000	1.89613	1.19209e-07
0.00000	0.125000	1.70240	1.19209e-07
0.0625000	0.125000	1.95615	6.55651e-06
0.125000	0.125000	2.23451	9.53674e-06

0.187500	0.125000	2.54067	6.67572e-06
0.250000	0.125000	2.87830	0.00000
0.00000	0.250000	2.59643	4.76837e-07
0.0625000	0.250000	2.93217	9.05991e-06
0.125000	0.250000	3.30337	1.31130e-05
0.187500	0.250000	3.71482	8.82149e-06
0.250000	0.250000	4.17198	2.38419e-07
0.00000	0.375000	3.76186	2.38419e-07
0.0625000	0.375000	4.21634	9.05991e-06
0.125000	0.375000	4.72261	1.31130e-05
0.187500	0.375000	5.28776	8.58307e-06
0.250000	0.375000	5.91989	4.76837e-07
0.00000	0.500000	5.32316	4.76837e-07
0.0625000	0.500000	5.95199	0.00000
0.125000	0.500000	6.65687	4.76837e-07
0.187500	0.500000	7.44826	0.00000
0.250000	0.500000	8.33804	1.43051e-06

Version History

6.4	Introduced



Chapter 9

Transforms

This section contains the following topics:

Overview: Transforms	374	Transforms Routines	376

Overview: Transforms

This section introduces some of the mathematical concepts used with IDL Advanced Math and Stats.

Fast Fourier Transforms

A fast Fourier transform (FFT) is a discrete Fourier transform that is computed efficiently. The straightforward method for computing the Fourier transform takes approximately n^2 operations, where n is the number of points in the transform, while the FFT (which computes the same values) takes approximately $n \log n$ operations. The algorithms in this chapter are modeled after the Cooley-Tukey (1965) algorithm. These functions are most efficient for integers that are highly composite, that is, integers that are a product of the small primes 2, 3, and 5.

For the [IMSL_FFTCOMP](#) function, there is a corresponding initialization function ([IMSL_FFTINIT](#)). Use [IMSL_FFTINIT](#) only when repeatedly transforming one-dimensional sequences of the same data type and length. In this situation, the initialization function computes the initial setup once; subsequently, the user calls the main function with the appropriate keyword. This may result in substantial computational savings. In addition to the one-dimensional transformation described above, the [IMSL_FFTCOMP](#) function also can be used to compute a complex two-dimensional FFT and its inverse.

Continuous Versus Discrete Fourier Transform

There is a close connection between the discrete Fourier transform and the continuous Fourier transform. The continuous Fourier transform is defined by Brigham (1974) as follows:

$$\hat{f}(\omega) = (Ff)(\omega) = \int_{-\infty}^{\infty} f(t) e^{-2\pi i \omega t} dt$$

Begin by making the following approximation:

$$\begin{aligned}\hat{f}(\omega) &\approx \int_{-T/2}^{T/2} f(t) e^{-2\pi i \omega t} dt \\ &= \int_0^T f(t - T/2) e^{-2\pi i \omega (t - T/2)} dt \\ &= e^{\pi i \omega T} \int_0^T f(t - T/2) e^{-2\pi i \omega t} dt\end{aligned}$$

If the last integral approximated using the rectangle rule with spacing $h = T/n$, the result is given below:

$$\hat{f}(\omega) \approx e^{\pi i \omega T} h \sum_{k=0}^{n-1} e^{-2\pi i \omega k h} f(kh - T/2)$$

Finally, setting $\omega = j / T$ for $j = 0, \dots, n-1$ yields:

$$\hat{f}(j/T) \approx e^{\pi i j} h \sum_{k=0}^{n-1} e^{-2\pi i j(k/n)} f(kh - T/2) = (-1)^j h \sum_{k=0}^{n-1} e^{-2\pi i j(k/n)} f_k^h$$

where the vector $f^h = (f(-T/2), \dots, f((n-1)h - T/2))$. Thus, after scaling the components by $(-1)^j h$, the discrete Fourier transform as computed in `IMSL_FFTCOMP` (with input f^h) is related to an approximation of the continuous Fourier transform by the above formula.

If the function f is expressed as a function, then the continuous Fourier transform:

$$\hat{f}$$

can be approximated using the IDL Advanced Math and Stats `IMSL_INTFCN` function to compute a Fourier transform as described in “[IMSL_INTFCN](#)” on page 284.

Transforms Routines

[IMSL_FFTCOMP](#)—Real or complex FFT.

[IMSL_FFTINIT](#)—Real or complex FFT initialization.

[IMSL_CONVOL1D](#)—Compute discrete convolution.

[IMSL_CORR1D](#)—Compute discrete correlation.

[IMSL_LAPLACE_INV](#)—Approximate inverse Laplace transform of a complex function.

IMSL_FFTCOMP

The IMSL_FFTCOMP function computes the discrete Fourier transform of a real or complex sequence. Using keywords, a real-to-complex transform or a two-dimensional complex Fourier transform can be computed.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_FFTCOMP(a [, COSINE=value] [, SINE=value] [, /DOUBLE]  
[, COMPLEX=value] [, BACKWARD=value] [, INIT_PARAMS=array] )
```

Return Value

Transformed sequence. If *A* is one-dimensional, type of *A* determines whether the real or complex transform is computed, where *A* is array *a*. If *A* is two-dimensional, complex transform is always computed.

Arguments

a

Array containing the periodic sequence.

Keywords

COSINE

If present and nonzero, then IMSL_FFTCOMP computes the discrete Fourier cosine transformation of an even sequence.

SINE

If present and nonzero, then IMSL_FFTCOMP computes the discrete Fourier sine transformation of an odd sequence.

DOUBLE

If present and nonzero, double precision is used.

COMPLEX

If present and nonzero, the complex transform is computed. If A is complex, this keyword is not required to ensure that a complex transform is computed. If A is real, it is promoted to complex internally.

BACKWARD

If present and nonzero, the backward transform is computed. See “[Discussion](#)” below for more details on this option.

INIT_PARAMS

Array containing parameters used when computing a one-dimensional FFT. If `IMSL_FFTCOMP` is used repeatedly with arrays of the same length and data type, it is more efficient to compute these parameters only once with a call to the [IMSL_FFTINIT](#) function.

Discussion

The `IMSL_FFTCOMP` function’s default action is to compute the FFT of array A , with the type of FFT performed dependent upon the data type of the input array A . (If A is a one-dimensional real array, the real FFT is computed; if A is a one-dimensional complex array, the complex FFT is computed; and if A is a two-dimensional real or complex array, the complex FFT is computed.) If the complex FFT of a one-dimensional real array is desired, the keyword `COMPLEX` should be specified. The keywords `SINE` and `COSINE` allow `IMSL_FFTCOMP` to be used to compute the discrete Fourier sine or cosine transformation of a one dimensional real array. The remainder of this section is divided into separate discussions of the various uses of `IMSL_FFTCOMP`.

Case 1: One-dimensional Real FFT

If A is one-dimensional and real, the `IMSL_FFTCOMP` function computes the discrete Fourier transform of a real array of length $n = N\_ELEMENTS(a)$. The method used is a variant of the Cooley-Tukey algorithm, which is most efficient when n is a product of small prime factors. If n satisfies this condition, then the computational effort is proportional to $n \log n$.

By default, IMSL_FFTCOMP computes the forward transform. If n is even, the forward transform is as follows:

$$q_{2m-1} = \sum_{k=0}^{n-1} p_k \cos \frac{2\pi km}{n}$$

$$q_{2m} = - \sum_{k=0}^{n-1} p_k \sin \frac{2\pi km}{n}$$

$$q_0 = \sum_{k=0}^{n-1} p_k$$

If n is odd, q_m is defined as above for m from 1 to $(n-1)/2$.

Let f be a real-valued function of time. Suppose f is sampled at n equally spaced time intervals of length Δ seconds starting at time t_0 :

$$p_i = f(t_0 + i\Delta) \quad i = 0, 1, \dots, n-1$$

Assume that n is odd for the remainder of the discussion for the case in which A is real. The IMSL_FFTCOMP function treats this sequence as if it were periodic of period n . In particular, it assumes that $f(t_0) = f(t_0 + n\Delta)$. Hence, the period of the function is assumed to be $T = n\Delta$. The above transform is inverted for the following:

$$p_m = \frac{1}{n} \left[q_0 + 2 \sum_{k=0}^{(n-3)/2} q_{2k+1} \cos \frac{2\pi(k+1)m}{n} - 2 \sum_{k=0}^{(n-3)/2} q_{2k+2} \sin \frac{2\pi(k+1)m}{n} \right]$$

This formula can be interpreted in the following manner: The coefficients q produced by IMSL_FFTCOMP determine an interpolating trigonometric polynomial to the data. That is, if the equations are defined as:

$$g(t) = \frac{1}{n} \left[q_0 + 2 \sum_{k=0}^{(n-3)/2} q_{2k+1} \cos \frac{2\pi(k+1)(t-t_0)}{n\Delta} - 2 \sum_{k=0}^{(n-3)/2} q_{2k+2} \sin \frac{2\pi(k+1)(t-t_0)}{n\Delta} \right]$$

$$g(t) = \frac{1}{n} \left[q_0 + 2 \sum_{k=0}^{(n-3)/2} q_{2k+1} \cos \frac{2\pi(k+1)(t-t_0)}{T} - 2 \sum_{k=0}^{(n-3)/2} q_{2k+2} \sin \frac{2\pi(k+1)(t-t_0)}{T} \right]$$

then the result is as follows:

$$f(t_0 + i\Delta) = g(t_0 + i\Delta)$$

Now suppose the dominant frequencies are to be obtained. Form the array P of length $(n + 1) / 2$ as follows:

$$P_0 = |q_0|$$

$$P_k = \sqrt{q_{2k+1}^2 + q_{2k}^2} \quad k = 1, 2, \dots, (n - 1)/2$$

These numbers correspond to the energy in the spectrum of the signal. In particular, P_k corresponds to the energy level at the following frequency:

$$\frac{k}{T} = \frac{k}{n\Delta} \quad k = 0, 1, \dots, \frac{n-1}{2}$$

Furthermore, note that there are only:

$$(n + 1)/2 \approx T/(2\Delta)$$

resolvable frequencies when n observations are taken. This is related to the Nyquist phenomenon, which is induced by discrete sampling of a continuous signal. Similar relations hold for the case when n is even.

If the keyword BACKWARD is specified, the backward transform is computed. If n is even, the backward transform is as follows:

$$q_m = p_0 + (-1)^{m+1} p_{n-1} + 2 \sum_{k=0}^{n/2-2} p_{2k+1} \cos \frac{2\pi(k+1)m}{n} - 2 \sum_{k=0}^{n/2-2} p_{2k+2} \sin \frac{2\pi(k+1)m}{n}$$

If n is odd, the following is true:

$$q_m = p_0 + 2 \sum_{k=0}^{(n-3)/2} p_{2k+1} \cos \frac{2\pi(k+1)m}{n} - 2 \sum_{k=0}^{(n-3)/2} p_{2k+2} \sin \frac{2\pi(k+1)m}{n}$$

The backward Fourier transform is the unnormalized inverse of the forward Fourier transform.

IMSL_FFTCOMP is based on the real FFT in FFTPACK, which was developed by Paul Swarztrauber at the National Center for Atmospheric Research.

Case 2: One-dimensional Complex FFT

If A is one-dimensional and complex, the IMSL_FFTCOMP function computes the discrete Fourier transform of a complex array of size $n = N\_ELEMENTS(a)$. The method used is a variant of the Cooley Tukey algorithm, which is most efficient when

n is a product of small prime factors. If n satisfies this condition, the computational effort is proportional to $n \log n$.

By default, `IMSL_FFTCOMP` computes the forward transform as in the equation below:

$$q_j = \sum_{m=0}^{n-1} p_m e^{(-2\pi i m j)/n}$$

Note, the Fourier transform can be inverted as follows:

$$p_m = \frac{1}{n} \sum_{j=0}^{n-1} q_j e^{2\pi i j (m/n)}$$

This formula reveals the fact that, after properly normalizing the Fourier coefficients, you have coefficients for a trigonometric interpolating polynomial to the data.

If the keyword `BACKWARD` is used, the following computation is performed:

$$q_j = \sum_{m=0}^{n-1} p_m e^{2\pi i m (j/n)}$$

Furthermore, the relation between the forward and backward transforms is that they are unnormalized inverses of each other. In other words, the following code fragment begins with an array p and concludes with an array $p_2 = np$:

```
q  = IMSL_FFTCOMP(p)
p2 = IMSL_FFTCOMP(q, /Backward)
```

Case 3: Two-dimensional FFT

If A is two-dimensional and real or complex, the `IMSL_FFTCOMP` function computes the discrete Fourier transform of a two-dimensional complex array of size $n \times m$ where $n = N\_ELEMENTS(a(*, 0))$ and $m = N\_ELEMENTS(a(0, *))$. The method used is a variant of the Cooley-Tukey algorithm, which is most efficient when both n and m are a product of small prime factors. If n and m satisfy this condition, then the computational effort is proportional to $nm \log nm$.

By default, given a two-dimensional array, `IMSL_FFTCOMP` computes the forward transform as in the following equation:

$$q_{jk} = \sum_{s=0}^{n-1} \sum_{t=0}^{m-1} p_{st} e^{-2\pi i js/n} e^{-2\pi i kt/m}$$

Note, the Fourier transform can be inverted as follows:

$$p_{jk} = \frac{1}{nm} \sum_{s=0}^{n-1} \sum_{t=0}^{m-1} q_{st} e^{2\pi i js/n} e^{2\pi i kt/m}$$

This formula reveals the fact that, after properly normalizing the Fourier coefficients, you have the coefficients for a trigonometric interpolating polynomial to the data.

If the keyword `BACKWARD` is used, the following computation is performed:

$$p_{jk} = \sum_{s=0}^{n-1} \sum_{t=0}^{m-1} q_{st} e^{2\pi i js/n} e^{2\pi i kt/m}$$

Case 4: Cosine Transform of a Real Sequence:

If the keyword `COSINE` is present and nonzero, the `IMSL_FFTCOMP` function computes the discrete Fourier cosine transform of a real vector of size N . The method used is a variant of the Cooley-Tukey algorithm, which is most efficient when $N-1$ is a product of small prime factors. If N satisfies this condition, then the computational effort is proportional to $N \log N$. Specifically, given an N -vector p , `IMSL_FFTCOMP` returns in q :

$$q_m = 2 \sum_{n=1}^{N-2} p_n \sin\left(\frac{mn\pi}{N-1}\right) + s_0 + s_{N-1}(-1)^m$$

where p = array a and q = result.

Finally, note that the Fourier cosine transform is its own (unnormalized) inverse.

Case 5: Sine Transform of a Real Sequence

If the keyword SINE is present and nonzero, the IMSL_FFTCOMP function computes the discrete Fourier sine transform of a real vector of size N . The method used is a variant of the Cooley-Tukey algorithm, which is most efficient when $N + 1$ is a product of small prime factors. If N satisfies this condition, then the computational effort is proportional to $N \log N$. Specifically, given an N -vector p , IMSL_FFTCOMP returns in q :

$$q_m = 2 \sum_{n=0}^{N-2} p_n \sin\left(\frac{(m+1)(n+1)\pi}{N+1}\right)$$

where p = array a and q = result.

Finally, note that the Fourier sine transform is its own (unnormalized) inverse.

Examples

Example 1

This example uses a pure cosine wave as a data array, and its Fourier series is recovered. The Fourier series is an array with all components zero except at the appropriate frequency where it has an $n/2$.

```
n = 7
; Fill up the data array with a pure cosine wave.
p = COS(FINDGEN(n) * 2 * !Pi/n)
PM, p
    1.00000
    0.623490
   -0.222521
   -0.900969
   -0.900969
   -0.222521
    0.623490
q = IMSL_FFTCOMP(p)
; Call IMSL_FFTCOMP to compute the FFT.
PM, q, FORMAT = '(f8.3)'
; Output results.
0.000
3.500
```

```

0.000
-0.000
-0.000
0.000
-0.000

```

Example 2: Resolving Dominant Frequencies

The following procedure demonstrates how the FFT can be used to resolve the dominant frequency of a signal. Call `IMSL_FFTCOMP` with a data vector of length $n = 15$, filled with pure, exponential signals of increasing frequency and decreasing strength. Using the computed FFT, the relative strength of the frequencies is resolved. It is important to note that for an array of length n , at most $(n + 1)/2$ frequencies can be resolved using the computed FFT.

```

.RUN
PRO power_spectrum
  n = 15
  ; Define the length of the signal.
  num_freq = n/2 + (n MOD 2)
  z = COMPLEX(0, FINDGEN(n) * 2 * !Pi/n)
  p = COMPLEXARR(n)
  FOR i = 0, num_freq - 1 DO p = p + EXP(i * z)/(i + 1)
  ; Fill up the data array.
  q = IMSL_FFTCOMP(p)
  ; Compute the FFT.
  power = FLTARR(num_freq)
  IF ((n MOD 2) EQ 0) THEN BEGIN
    power(0) = ABS(q(0))^2
    FOR i = 1, (num_freq - 2) DO $
      power(i) = q(i) * CONJ(q(i)) + q(n-i-1) * CONJ(q(n-i-1))
    power(num_freq - 1) = q(num_freq - 1) * CONJ(q(num_freq - 1))
  ENDIF
  ; Determine the strengths of the frequencies. The method is
  ; dependent upon whether n is even or odd.
  IF ((n MOD 2) EQ 1) THEN BEGIN
    FOR i = 1, (num_freq - 1) DO power(i) = $
      q(i)^2 + q(n - i)^2
    power(0) = q(0)^2
  ENDIF
  PRINT, '    frequency    strength' &$
  PRINT, '    -----    -----' &$
  FOR i = 0, 7 DO PRINT, i, power(i)
  ; Display frequencies and strengths.
END

frequency    strength
-----

```

0	225.000
1	56.2500
2	25.0000
3	14.0625
4	9.00000
5	6.25000
6	4.59183
7	3.51562

Example 3: Computing a Two-dimensional FFT

This example computes the forward transform of a two-dimensional matrix followed by the backward transform. Notice that the process of computing the forward transform followed by the backward transform multiplies the entries of the original matrix by the product of the lengths of the two dimensions.

```

n = 4
m = 5
p = COMPLEXARR(n, m)
FOR i = 0, n - 1 DO BEGIN &$
  z = COMPLEX(0, 2 * i * 2 * !Pi/n) &$
  FOR j = 0, m - 1 DO BEGIN &$
    w = COMPLEX(0, 5 * j * 2 * !Pi/m) &$
    p(i, j) = EXP(z) * EXP(w) &$
  ENDFOR &$
ENDFOR
q = IMSL_FFTCOMP(p)
p2 = IMSL_FFTCOMP(q, /Backward)
FORMAT = ' (4 ("( ",f6.2," ",f5.2," ")",2x)) '
PM, p, FORMAT = format, TITLE = 'p'
p
( 1.0, 0.0)( 1.0, 0.0)( 1.0, 0.0)( 1.0, 0.0)
( 1.0, 0.0)(-1.0,-0.0)(-1.0,-0.0)(-1.0,-0.0)
(-1.0,-0.0)(-1.0,-0.0)( 1.0, 0.0)( 1.0, 0.0)
( 1.0, 0.0)( 1.0, 0.0)( 1.0, 0.0)(-1.0,-0.0)
(-1.0,-0.0)(-1.0,-0.0)(-1.0,-0.0)(-1.0,-0.0)
PM, q, FORMAT = format, TITLE = 'q = IMSL_FFTCOMP(p) '
q = IMSL_FFTCOMP(p)
( 0.0, 0.0)(-0.0, 0.0)( 0.0, 0.0)(-0.0, 0.0)
( 0.0, 0.0)(-0.0,-0.0)( 0.0,-0.0)( 0.0,-0.0)
( 0.0, 0.0)(-0.0, 0.0)(20.0, 0.0)(-0.0,-0.0)
(-0.0,-0.0)( 0.0,-0.0)( 0.0,-0.0)( 0.0,-0.0)
( 0.0, 0.0)(-0.0, 0.0)(-0.0,-0.0)(-0.0,-0.0)
PM, p2, FORMAT = format, TITLE = 'p2 = IMSL_FFTCOMP(q, /BACKWARD) '
p2 = IMSL_FFTCOMP(q, /Backward)
( 20., 0.)( 20., 0.)( 20., 0.)( 20., 0.)
( 20., 0.)(-20.,-0.)(-20.,-0.)(-20.,-0.)
(-20.,-0.)(-20.,-0.)( 20., 0.)( 20., 0.)
( 20., 0.)( 20., 0.)( 20., 0.)(-20.,-0.)

```

(-20.,-0.) (-20.,-0.) (-20.,-0.) (-20.,-0.)

Version History

6.4	Introduced

IMSL_FFTINIT

The IMSL_FFTINIT function computes the parameters for a one-dimensional FFT to be used in the IMSL_FFTCOMP function with the keyword INIT_PARAMS.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_FFTINIT(n [, /DOUBLE] [, COMPLEX=value] [, SINE=value]  
[, COSINE=value])
```

Return Value

A one-dimensional array of length $2n + 15$ that can then be used by IMSL_FFTCOMP when the optional parameter INIT_PARAMS is specified.

Arguments

n

Length of the sequence to be transformed.

Keywords

DOUBLE

If present and nonzero, double precision is used and the returned array is double precision. This keyword does not have an effect if the initialization is being computed for a complex FFT.

COMPLEX

If present and nonzero, the parameters for a complex transform are computed.

SINE

If present and nonzero, then parameters for a discrete Fourier cosine transformation are returned. See the [IMSL_FFTCOMP](#) keyword SINE.

COSINE

If present and nonzero, then parameters for a discrete Fourier cosine transformation are returned. See the [IMSL_FFTCOMP](#) keyword SINE.

Discussion

The `IMSL_FFTINIT` function should be used when many calls are to be made to [IMSL_FFTCOMP](#) without changing the data type of the array and the length of the sequence. The default action of `IMSL_FFTINIT` is to compute the parameters necessary for a real FFT. If parameters for a complex FFT are needed, the keyword `COMPLEX` should be specified.

The `IMSL_FFTINIT` function is based on the routines `RFFTI` and `RFFTI` in `FFTPACK`, which was developed by Paul Swarztrauber at the National Center for Atmospheric Research.

Example

In this example, two distinct, real FFTs are computed by calling `IMSL_FFTINIT` once, then calling `IMSL_FFTCOMP` twice.

```
.RUN
n = 7
; Define the length of the signals.
init_params = IMSL_FFTINIT(7)
; Initialize the parameters by calling IMSL_FFTINIT.
FOR j = 0, 2 DO BEGIN
    p = COS(j * FINDGEN(n) * 2 * !Pi/n)
    q = IMSL_FFTCOMP(p, Init_Params = init_params)
    PM, 'p', 'q', FORMAT = '(7x, a1, 10x, a1)'
    FOR i = 0, n - 1 DO PM, p(i), q(i), FORMAT = '(f10.5, f10.2)'
ENDFOR
END

; For each pass through loop, compute a real FFT of an array of
; length n and output both original signal and computed FFT.
      p          q
1.00000      7.00
1.00000      0.00
1.00000      0.00
```


1.00000	0.00
1.00000	0.00
1.00000	-0.00
1.00000	0.00
p	q
1.00000	0.00
0.62349	3.50
-0.22252	0.00
-0.90097	-0.00
-0.90097	-0.00
-0.22252	0.00
0.62349	-0.00
p	q
1.00000	-0.00
-0.22252	0.00
-0.90097	-0.00
0.62349	3.50
0.62349	-0.00
-0.90097	0.00
-0.22252	0.00

Version History

6.4	Introduced

IMSL_CONVOL1D

The IMSL_CONVOL1D function computes the discrete convolution of two one-dimensional arrays.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_CONVOL1D(*x*, *y* [, **DIRECT**=*value*] [, **PERIODIC**=*value*])

Return Value

A one-dimensional array containing the discrete convolution of *x* and *y*.

Arguments

x

One-dimensional array.

y

One-dimensional array.

Keywords

DIRECT

If present and nonzero, causes the computations to be done by the direct method instead of the FFT method regardless of the size of the vectors passed in.

PERIODIC

If present and nonzero, then a *circular convolution* is computed.

Discussion

The IMSL_CONVOL1D function computes the discrete convolution of two sequences x and y .

Let n_x be the length of x , and n_y denote the length of y . If the keyword PERIODIC is set, then $n_z = \max\{n_x, n_y\}$, otherwise n_z is set to the smallest whole number, $n_z \geq n_x + n_y - 1$, of the form:

$$n_z = 2^\alpha 3^\beta 5^\gamma \quad : \alpha, \beta, \gamma \text{ nonnegative integers}$$

The arrays x and y are then zero-padded to a length n_z . Then, we compute:

$$z_i = \sum_{j=0}^{n_z-1} x_{i-j} y_j$$

where the index on x is interpreted as a nonnegative number between 0 and $n_z - 1$.

The technique used to compute the z_i 's is based on the fact that the (complex discrete) Fourier transform maps convolution into multiplication. Thus, the Fourier transform of z is given by:

$$\hat{z}(n) = \hat{x}(n) \hat{y}(n)$$

where the following equation is true:

$$\hat{z}(n) = \sum_{m=0}^{n_z-1} z_m e^{-2\pi i m n / n_z}$$

The technique used here to compute the convolution is to take the discrete Fourier transform of x and y , multiply the results together component-wise, and then take the inverse transform of this product. It is very important to make sure that n_z is the product of small primes if PERIODIC is set. If n_z is a product of small primes, then the computational effort will be proportional to $n_z \log(n_z)$. If PERIODIC is not set, then n_z is chosen to be a product of small primes.

We point out that if x and y are not complex, then no complex transforms of x or y are taken, since a real transforms can simulate the complex transform above. Such a strategy is six times faster and requires less space than when using the complex transform.

Example

This example computes simple moving-average digital filter plots of 5-point and 25-point moving-average filters of noisy data. Results are shown in figures [Figure 9-1](#) and [Figure 9-2](#).

```

PRO Convolve_ex1
  IMSL_RANDOMOPT, SET = 1234579L
  ; Set the random number seed.
  ny = 100
  t = FINDGEN(ny)/(ny-1)
  y = SIN(2*!PI*t) + .5*IMSL_RANDOM(ny, /Uniform) -.25
  ; Define a 1-period sine wave with added noise.
  win=0
  FOR nfltr = 5, 25, 20 DO BEGIN
    nfltr_str = strcompress(nfltr,/Remove_All)
    fltr = fltarr(nfltr)
    fltr(*) = 1./nfltr
    ; Define the NFLTR-point moving average array.
    z = IMSL_CONVOLVE1D(fltr, y, /Periodic)
    ; Convolve filter and signal, using keyword Periodic.
    WINDOW, win++
    PLOT, y, LINESTYLE = 1, TITLE = nfltr_str + $
      '-point Moving Average'
    OPLOT, shift(z, -nfltr/2)
  ENDFOR
END

```

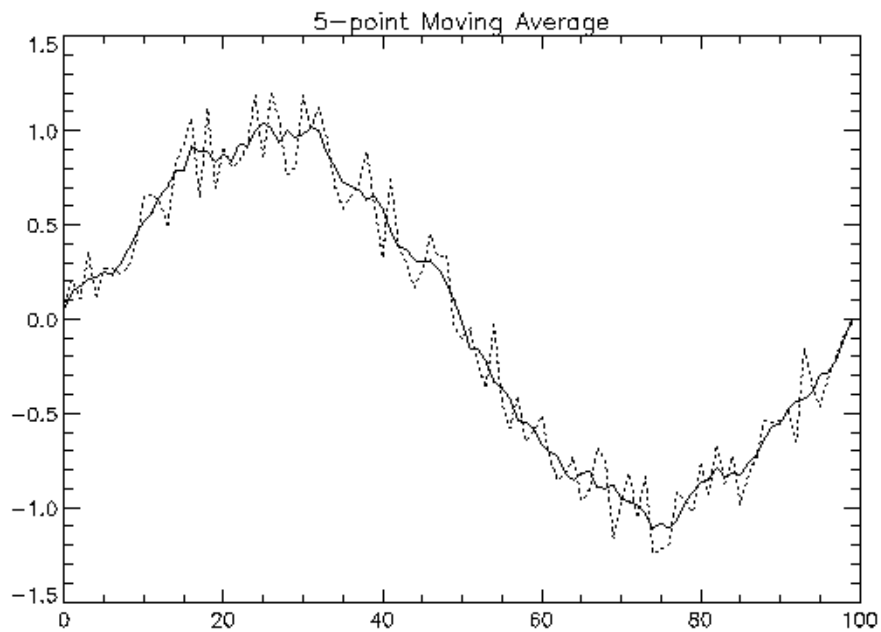


Figure 9-1: 5 Point Moving Average

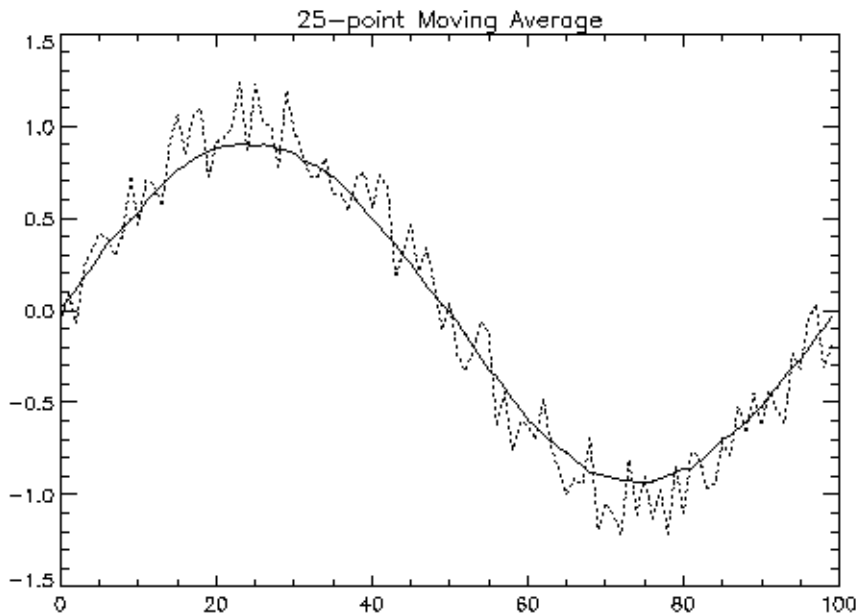


Figure 9-2: 25 Point Moving Average

Version History

6.4	Introduced

IMSL_CORR1D

The IMSL_CORR1D function computes the discrete correlation of two one-dimensional arrays.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_CORR1D(*x* [, *y*] [, PERIODIC=*value*])

Return Value

A one-dimensional array containing the discrete convolution of *x* and *x*, or *x* and *y* if *y* is supplied.

Arguments

x

One-dimensional array.

y

(Optional) One-dimensional array.

Keywords

PERIODIC

If present and nonzero, then the input data is periodic.

Discussion

The IMSL_CORR1D function computes the discrete correlation of two sequences *x* and *y*. If only one argument is passed, then IMSL_CORR1D computes the discrete correlation of *x* and *x*.

More precisely, let n be the length of x and y . If PERIODIC is set, then $n_z = n$, otherwise n_z is set to the smallest whole number, $n_z \geq 2n - 1$, of the form:

$$n_z = 2^\alpha 3^\beta 5^\gamma \quad : \alpha, \beta, \gamma \text{ nonnegative integers}$$

The arrays x and y are then zero-padded to a length n_z . Then, we compute:

$$z_i = \sum_{j=0}^{n_z-1} x_{i+j} y_j$$

where the index on x is interpreted as a positive number between 0 and $n_z - 1$.

The technique used to compute the z_i 's is based on the fact that the (complex discrete) Fourier transform maps correlation into multiplication. Thus, the Fourier transform of z is given by:

$$\hat{z}_j = \hat{x}_j \overline{\hat{y}_j}$$

where the following equation is true:

$$\hat{z}_j = \sum_{m=0}^{n_z-1} z_m e^{-2\pi i m j / n_z}$$

Thus, the technique used here to compute the correlation is to take the discrete Fourier transform of x and the conjugate of the discrete Fourier transform of y , multiply the results together component-wise, and then take the inverse transform of this product. It is very important to make sure that n_z is the product of small primes if the keyword PERIODIC is selected. If n_z is the product of small primes, then the computational effort will be proportional to $n_z \log(n_z)$. If PERIODIC is not set, then a good value is chosen for n_z so that the Fourier transforms are efficient and $n_z \geq 2n - 1$. This will mean that both vectors may be padded with zeros.

If x and y are not complex, then no complex transforms of x or y are taken, since a real transforms can simulate the complex transform above. Such a strategy is six times faster and requires less space than when using the complex transform.

Example

This example computes a periodic correlation between two distinct signals x and y . We have 100 equally spaced points on the interval $[0, 2\pi]$ and $f_1(x) = \sin(x)$. We define x and y as follows:

$$x_i = f_1\left(\frac{2\pi i}{n-1}\right) \quad i = 0, \dots, n-1$$

$$y_i = f_1\left(\frac{2\pi i}{n-1} + \frac{\pi}{2}\right) \quad i = 0, \dots, n-1$$

Note that the maximum value of z (the correlation of x with y) occurs at $i = 25$, which corresponds to the offset.

```
n = 100
t = 2*!DPI*FINDGEN(n) / (n-1)
x = SIN(t)
y = SIN(t+!dpi/2)
; Define the signals and compute the norms of the signals.
xnorm = IMSL_NORM(x)
ynorm = IMSL_NORM(y)
z = IMSL_CORR1D(x, y, /Periodic)/(xnorm*ynorm)
; Compute periodic correlation, and find the largest normalized
; element of the result.
max_z = (SORT(z))(N_ELEMENTS(z)-1)
PRINT, max_z, z(max_z)
      25  1.00
```

Version History

6.4	Introduced

IMSL_LAPLACE_INV

The IMSL_LAPLACE_INV function computes the inverse Laplace transform of a complex function.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_LAPLACE_INV(f, sigma0, t [, BIG_COEF_LOG=variable]
    [, BVALUE=parameter] [, COND_ERR=variable] [, DISC_ERR=variable]
    [, /DOUBLE] [, ERR_EST=variable] [, INDICATORS=variable] [, K=variable]
    [, MTOP=value] [, PSEUDO_ACC=value] [, R=variable] [, SIGMA=parameter]
    [, SMALL_COEF_LOG=variable] [, TRUNC_ERR=variable])
```

Return Value

One-dimensional array of length n whose i -th component contains the approximate value of the inverse Laplace transform at the point $t(i)$.

Arguments

f

Scalar string specifying the user-supplied function for which the inverse Laplace transform will be computed.

sigma0

An estimate for the maximum of the real parts of the singularities of f . If unknown, set $\text{sigma0} = 0.0$.

t

One-dimensional array of size n containing the points at which the inverse Laplace transform is desired.

Keywords

BIG_COEF_LOG

Named variable into which the logarithm of the largest coefficient in the decay function is stored. See “[Discussion](#)” on page 401 for details.

BVALUE

The second parameter of the Laguerre expansion. If BVALUE is less than $2.0 * (\text{Sigma} - \text{sigma0})$, it is reset to $2.5 * (\text{Sigma} - \text{sigma0})$. Default: $\text{BVALUE} = 2.5 * (\text{Sigma} - \text{sigma0})$

COND_ERR

Named variable into which the estimate of the pseudo condition error on the basis of minimal noise levels in the function values is stored.

DISC_ERR

Named variable into which the estimate of the pseudo discretization error is stored.

DOUBLE

If present and nonzero, double precision is used.

ERR_EST

Named variable into which an overall estimate of the pseudo error, $\text{DISC\_EST} + \text{TRUNC\_ERR} + \text{COND\_ERR}$ is stored. See “[Discussion](#)” on page 401 for details.

INDICATORS

Named variable into which an one-dimensional array of length n containing the overflow/underflow indicators for the computed approximate inverse Laplace transform is stored. [Table 9-1](#) shows, for the i -th point at which the transform is computed, what $\text{INDICATORS}(i)$ signifies.

Indicators(i)	Meaning
1	Normal termination.
2	The value of the inverse Laplace transform is too large to be representable. This component of the result is set to NaN.
3	The value of the inverse Laplace transform is found to be too small to be representable. This component of the result is set to 0.0.
4	The value of the inverse Laplace transform is estimated to be too large, even before the series expansion, to be representable. This component of the result is set to NaN.
5	The value of the inverse Laplace transform is estimated to be too small, even before the series expansion, to be representable. This component of the result is set to 0.0.

Table 9-1: Indicator Meanings

K

Named variable into which the coefficient of the decay function is stored. See “[Discussion](#)” on page 401 for details.

MTOP

An upper limit on the number of coefficients to be computed in the Laguerre expansion. The keyword MTOP must be a multiple of four. Default: MTOP = 1024

PSEUDO_ACC

The required absolute uniform pseudo accuracy for the coefficients and inverse Laplace transform values. Default: PSEUDO_ACC = $\text{SQRT}(\epsilon)$, where ϵ is machine epsilon

R

Named variable into which the base of the decay function is stored. See “[Discussion](#)” on page 401 for details.

SIGMA

The first parameter of the Laguerre expansion. If SIGMA is not greater than *sigma0*, it is reset to *sigma0*+ 0.7. Default: *Sigma* = *sigma0*+ 0.7

SMALL_COEF_LOG

Named variable into which the logarithm of the smallest nonzero coefficient in the decay function is stored. See “[Discussion](#)” on page 401 for details.

TRUNC_ERR

Named variable into which the estimate of the pseudo truncation error is stored.

Discussion

The IMSL_LAPLACE_INV function computes the inverse Laplace transform of a complex-valued function. Recall that if f is a function that vanishes on the negative real axis, then the Laplace transform of f is defined by:

$$L[f](s) = \int_0^{\infty} e^{-sx} f(x) dx$$

It is assumed that for some value of s the integrand is absolutely integrable.

The computation of the inverse Laplace transform is based on a modification of Weeks’ method (see Weeks (1966)) due to Garbow et al. (1988). This method is suitable when f has continuous derivatives of all orders on $[0, \infty)$. In particular, given a complex-valued function $F(s) = L[f](s)$, f can be expanded in a Laguerre series whose coefficients are determined by F . This is fully described in Garbow et al. (1988) and Lyness and Giunta (1986).

The algorithm attempts to return approximations $g(t)$ to $f(t)$ satisfying:

$$\left| \frac{g(t) - f(t)}{e^{\sigma t}} \right| < \varepsilon$$

where $\varepsilon = \text{PSEUDO\_ACC}$ and $\sigma = \text{Sigma} > \text{sigma0}$. The expression on the left is called the pseudo error. An estimate of the pseudo error is available in ERR_EST.

The first step in the method is to transform F to ϕ where:

$$\phi(z) = \frac{b}{1-z} F\left(\frac{b}{1-z} - \frac{b}{2} + \sigma\right)$$

Then, if f is smooth, it is known that ϕ is analytic in the unit disc of the complex plane and hence has a Taylor series expansion:

$$\phi(z) = \sum_{s=0}^{\infty} a_s z^s$$

which converges for all z whose absolute value is less than the radius of convergence R_c . This number is estimated in the output keyword R . Using the output keyword K , the smallest number K is estimated which satisfies:

$$|a_s| < \frac{K}{R^s}$$

for all $R < R_c$.

The coefficients of the Taylor series for ϕ can be used to expand f in a Laguerre series:

$$f(t) = \sum_{s=0}^{\infty} a_s e^{-bt/2} L_s(bt)$$

Examples

Example 1

This example computes the inverse Laplace transform of the function $(s-1)^{-2}$, and prints the computed approximation, true transform value, and difference at five points. The correct inverse transform is xe^x . From Abramowitz and Stegun (1964).

```
.RUN
FUNCTION fcn, x
; Return 1/(s - 1)**2
one = COMPLEX(1.0, 0.0)
f = one/((x - one)*(x - 1))
RETURN, f
END

.RUN
n = 5
; Initialize t and compute inverse.
t = FINDGEN(n) + 0.5
l_inverse = IMSL_LAPLACE_INV('fcn', 1.5, t)
; Compute true inverse, relative difference.
true_inverse = t*EXP(t)
relative_diff = ABS((l_inverse - true_inverse) / true_inverse)
PM, [[t(0:*)], [l_inverse(0:*)], [true_inverse(0:*)], $
[relative_diff(0:*)]], $
Title = ' t          f_inv          true          diff'
END
```

t	f_inv	true	diff
0.500000	0.824348	0.824361	1.48223e-05
1.50000	6.72247	6.72253	1.01432e-05
2.50000	30.4562	30.4562	2.50504e-07

3.50000	115.906	115.904	1.84310e-05
4.50000	405.053	405.077	5.90648e-05

Example 2

This example computes the inverse Laplace transform of $e^{-1/s}/s$, and prints the computed approximation, true transform value, and difference at five points. Additionally, the inverse is returned, and a required accuracy for the inverse transform values is specified. The correct inverse transform is:

$$J_0(2\sqrt{x})$$

```
.RUN
FUNCTION fcn, x
; Return (1/s)(exp(-1/s)
one = COMPLEX(1.0, 0.0)
s_inverse = one / x
f = s_inverse*EXP(-1*(s_inverse))
RETURN, f
END

.RUN
n = 5
; Initialize t and compute inverse.
t = FINDGEN(n) + 0.5
l_inverse = IMSL_LAPLACE_INV('fcn', 0.0, t, $
Pseudo_Acc = 1.0e-6, Indicator = indicator)
true_inverse = FLOAT(IMSL_BESSJ(0, 2.0*SQRT(t)))
relative_diff = ABS((l_inverse - true_inverse) / true_inverse)
FOR i = 0, 4 DO BEGIN
  IF (indicator(i) EQ 0) THEN BEGIN
    PM, t(i), l_inverse(i), true_inverse(i), $
      relative_diff(i), $
      Title = '          t          f_inv          true          diff'

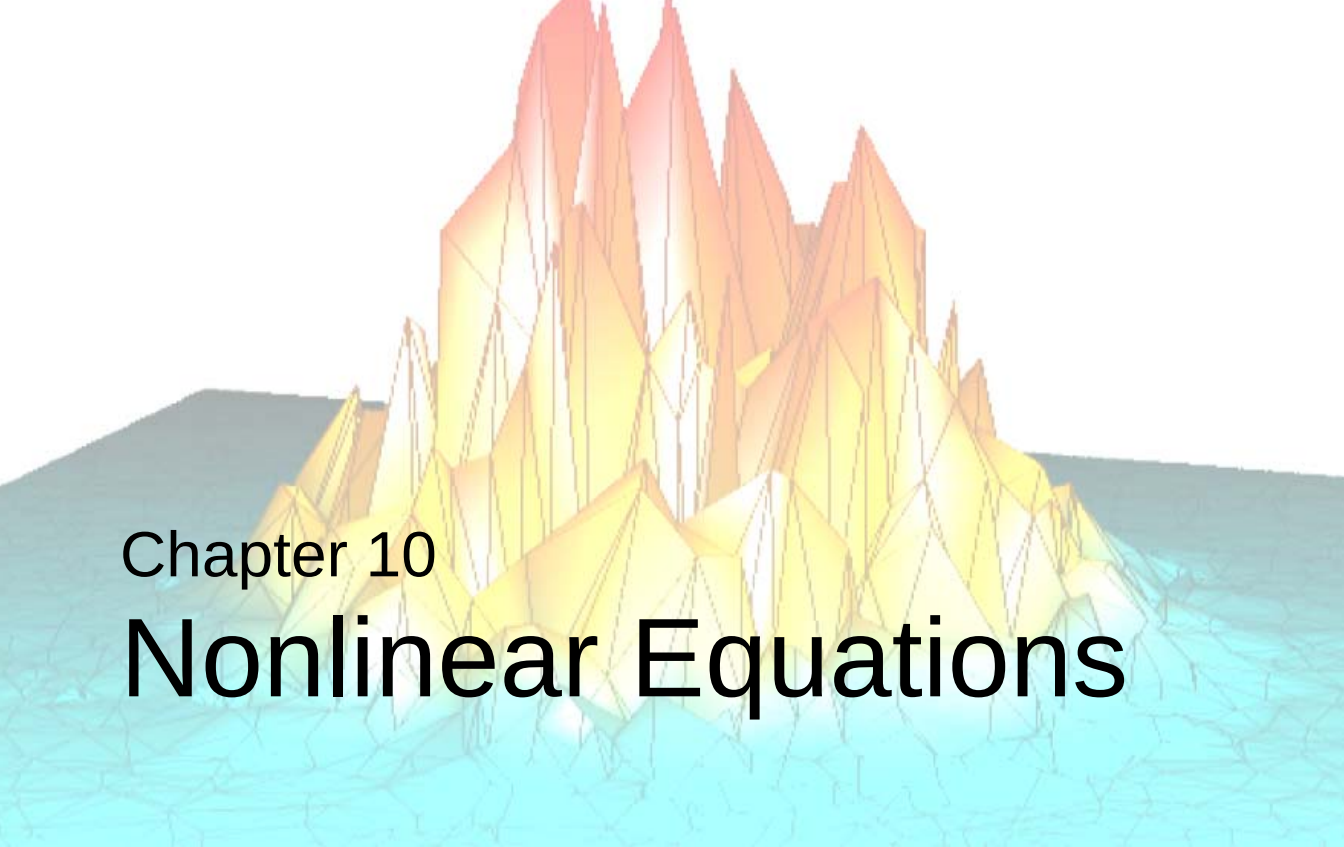
    ENDIF ELSE BEGIN
      PRINT, 'Overflow or underflow noted.'
    ENDELSE
  ENDFOR
END
```

t	f_inv	true	diff
0.50000	0.559134	0.559134	1.06602e-07
t	f_inv	true	diff
1.50000	-0.0229669	-0.0229670	4.21725e-06
t	f_inv	true	diff
2.50000	-0.310045	-0.310045	9.61226e-08
t	f_inv	true	diff
3.50000	-0.401115	-0.401115	2.22896e-07

t	f_inv	true	diff
4.50000	-0.370335	-0.370336	4.02369e-07

Version History

6.4	Introduced



Chapter 10

Nonlinear Equations

This section contains the following topics:

Overview: Nonlinear Equations	408	Nonlinear Equations Routines	409

Overview: Nonlinear Equations

This section introduces some of the mathematical concepts used with IDL Advanced Math and Stats.

Zeros of a Polynomial

A polynomial function of degree n can be expressed as follows:

$$p(z) = a_n z^n + a_{n-1} z^{n-1} + \dots + a_1 z + a_0$$

where $a_n \neq 0$. The [IMSL_ZEROPOLY](#) function finds zeros of a polynomial with real or complex coefficients using either the companion method or the Jenkins-Traub three-stage algorithm.

Zeros of a Function

The [IMSL_ZEROFCN](#) function uses Müller's method to find the real zeros of a real-valued function.

Root of System of Equations

A system of equations can be stated as follows:

$$f_i(x) = 0, \text{ for } i = 0, 1, \dots, n - 1$$

where $x \in R^n$, and $f_i : R^n \rightarrow R$.

The [IMSL_ZEROSYS](#) function uses a modified hybrid method due to M.J.D. Powell to find the zero of a system of nonlinear equations.

Nonlinear Equations Routines

Zeros of a Polynomial

[IMSL_ZEROPOLY](#)—Real or complex coefficients.

Zeros of a Function

[IMSL_ZEROFCN](#)—Real zeros of a function.

Root of a System of Equations

[IMSL_ZEROSYS](#)—Powell's hybrid method.

IMSL_ZEROPOLY

The IMSL_ZEROPOLY function finds the zeros of a polynomial with real or complex coefficients using the companion matrix method or, optionally, the Jenkins-Traub, three-stage algorithm.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_ZEROPOLY(coef[, /DOUBLE] [, COMPANION=value]  
[, JENKINS_TRAUB=value])
```

Return Value

The complex array of zeros of the polynomial.

Arguments

coef

Array containing coefficients of the polynomial in increasing order by degree. The polynomial is $coef(n) z^n + coef(n-1) z^{n-1} + \dots + coef(0)$.

Keywords

DOUBLE

If present and nonzero, double precision is used.

COMPANION

If present and nonzero, the companion matrix method is used. Default: companion matrix method

JENKINS_TRAUB

If present and nonzero, the Jenkins-Traub, three-stage algorithm is used.

Discussion

The IMSL_ZEROPOLY function computes the n zeros of the polynomial:

$$p(z) = a_n z^n + a_{n-1} z^{n-1} + \dots + a_1 z + a_0$$

where the coefficients a_i for $i = 0, 1, \dots, n$ are real and n is the degree of the polynomial.

The default method used by IMSL_ZEROPOLY is the companion matrix method. The companion matrix method is based on the fact that if C_a denotes the companion matrix associated with $p(z)$, then $\det(zI - C_a) = a(z)$, where I is an $n \times n$ identity matrix. Thus, $\det(z_0 I - C_a) = 0$ if, and only if, z_0 is a zero of $p(z)$. This implies that computing the eigenvalues of C_a will yield the zeros of $p(z)$. This method is thought to be more robust than the Jenkins-Traub algorithm in most cases, but the companion matrix method is not as computationally efficient. Thus, if speed is a concern, the Jenkins-Traub algorithm should be considered.

If the keyword JENKINS_TRAUB is set, then IMSL_ZEROPOLY function uses the Jenkins-Traub three-stage algorithm (Jenkins and Traub 1970, Jenkins 1975). The zeros are computed one-at-a-time for real zeros or two-at-a-time for a complex conjugate pair. As the zeros are found, the real zero or quadratic factor is removed by polynomial deflation.

Example

This example finds the zeros of the third-degree polynomial:

$$p(z) = z^3 - 3z^2 + 4z - 2$$

where z is a complex variable.

```
coef = [-2, 4, -3, 1]
; Set the coefficients.
zeros = IMSL_ZEROPOLY(coef)
; Compute the zeros.
PM, zeros, Title = $
'The complex zeros found are: '
; Print results.
The complex zeros found are:
(      1.00000,      0.00000)
(      1.00000,     -1.00000)
(      1.00000,      1.00000)
```

Errors

Warning Errors

`MATH_ZERO_COEFF`—First several coefficients of the polynomial are equal to zero. Several of the last roots are set to machine infinity to compensate for this problem.

`MATH_FEWER_ZEROS_FOUND`—Fewer than $(N\_ELEMENTS (coef) - 1)$ zeros were found. The root vector contains the value for machine infinity in the locations that do not contain zeros.

Version History

6.4	Introduced

IMSL_ZEROFCN

The IMSL_ZEROFCN function finds the real zeros of a real function using Müller's method.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_ZEROFCN(f[, /DOUBLE] [, ERR_ABS=value]
    [, ERR_REL=value] [, ETA=value] [, EPS=value] [, INFO=array]
    [, ITMAX=value] [, N_ROOTS=value] [, XGUESS=array] )
```

Return Value

An array containing the zeros x of the function.

Arguments

f

Scalar string specifying a user-supplied function for which the zeros are to be found. The f function accepts one scalar parameter from which the function is evaluated and returns a scalar of the same type.

Keywords

DOUBLE

If present and nonzero, double precision is used.

ERR_ABS

First stopping criterion. A zero, x_i , is accepted if $|f(x_i)| < \text{ERR\_ABS}$. Default: $\text{ERR\_ABS} = \text{SQRT}(\epsilon)$, where ϵ is the machine precision

ERR_REL

Second stopping criterion. A zero, x_i , is accepted if the relative change of two successive approximations to x_i is less than ERR_REL. Default: $\text{ERR\_REL} = \text{SQRT}(\epsilon)$, where ϵ is the machine precision

ETA

Spread criteria for multiple zeros. If the zero, x_i , has been computed and $|x_i - x_j| < \text{EPS}$, where x_j is a previously computed zero, then the computation is restarted with a guess equal to $x_i + \text{ETA}$. Default: $\text{ETA} = 0.01$

EPS

See ETA. Default: $\text{EPS} = \text{SQRT}(\epsilon)$, where ϵ is the machine precision.

INFO

Array of length N_ROOTS containing convergence information. The value $\text{INFO}(j - 1)$ is the number of iterations used in finding the j -th zero when convergence is achieved. If convergence is not obtained in ITMAX iterations, $\text{INFO}(j - 1)$ is greater than ITMAX.

ITMAX

Maximum number of iterations per zero. Default: $\text{ITMAX} = 100$.

N_ROOTS

Number of roots for IMSL_ZEROFCN to find. Default: $\text{N\_ROOTS} = 1$.

XGUESS

Array with N_ROOTS components containing the initial guesses for the zeros. Default: $\text{XGUESS} = 0$

Discussion

The IMSL_ZEROFCN function computes n real zeros of a real function f . Given a user-supplied function $f(x)$ and an n -vector of initial guesses $x_0, x_1, \dots, x_{n-1}$, the function uses Müller's method to locate n real zeros of f . The function has two convergence criteria. The first criterion requires that $|f(x_i^{(m)})|$ be less than ERR_ABS. The second criterion requires that the relative change of any two successive approximations to an x_i be less than ERR_REL. Here, $x_i^{(m)}$ is the m -th

approximation to x_i . Let ERR_ABS be denoted by ε_1 , and ERR_REL be denoted by ε_2 . The criteria can be stated mathematically as follows.

IMSL_ZEROFCN has two convergence criteria; “convergence” is the satisfaction of either criterion.

Criterion 1:

$$\left| f(x_i^{(m)}) \right| < \varepsilon_1$$

Criterion 2:

$$\left| \frac{x_i^{(m+1)} - x_i^{(m)}}{x_i^{(m)}} \right| < \varepsilon_2$$

“Convergence” is the satisfaction of either criterion.

Example

This example finds a real zero of the third-degree polynomial:

$$f(x) = x^3 - 3x^2 + 3x - 1$$

The results are shown in [Figure 10-1](#).

```
.RUN
; Define function f.
FUNCTION f, x
    return, x^3 - 3 * x^2 + 3 * x - 1
END

zero = IMSL_ZEROFCN('f')
; Compute the real zero(s).
x = 2 * FINDGEN(100)/99
PLOT, x, f(x)
; Plot results.
OPLOT, [zero], [f(zero)], Psym = 6
XYOUTS, .5, .5, 'Computed zero is at x = ' + $
    STRING(zero(0)), Charsize = 1.5
```

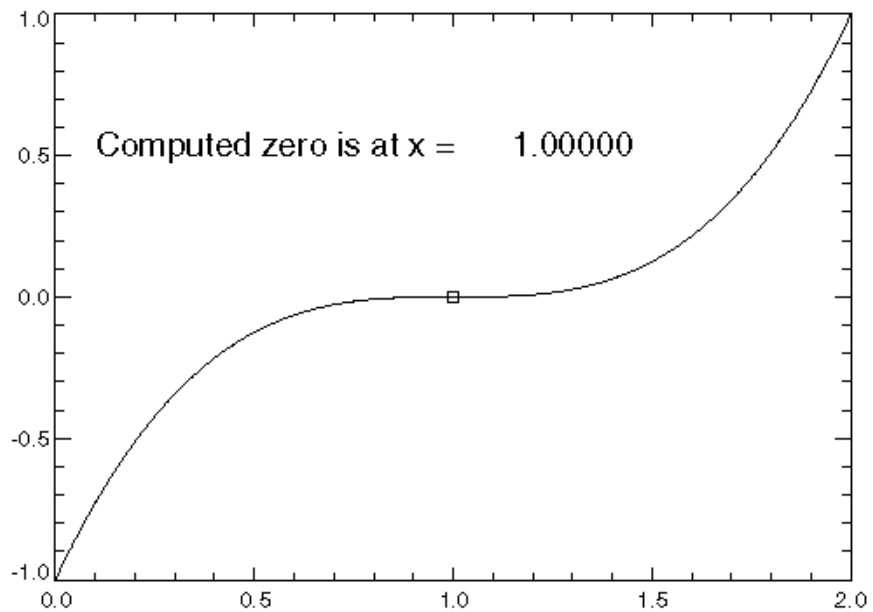


Figure 10-1: IMSL_ZEROFCN Function

Errors

Warning Errors

MATH_NO_CONVERGE_MAX_ITER—Function failed to converge within ITMAX iterations for at least one of the N_ROOTS roots.

Version History

6.4	Introduced

IMSL_ZEROSYS

The IMSL_ZEROSYS function solves a system of n nonlinear equations, $f_i(x) = 0$, using a modified Powell hybrid algorithm.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_ZEROSYS(f, n [, /DOUBLE] [, ERR_REL=value]
    [, FNORM=value] [, JACOBIAN=string] [, ITMAX=value] [, XGUESS=array])
```

Return Value

An array containing a solution of the system of equations.

Arguments

f

Scalar string specifying a user-supplied function to evaluate the system of equations to be solved. The f function accepts one parameter containing the point at which the functions are to be evaluated and returns the computed function values at the given point.

n

Number of equations to be solved and the number of unknowns.

Keywords

DOUBLE

If present and nonzero, double precision is used.

ERR_REL

Stopping criterion. The root is accepted if the relative error between two successive approximations to this root is less than ERR_REL. Default: $\text{ERR\_REL} = \text{SQRT}(\epsilon)$, where ϵ is the machine precision.

FNORM

Scalar with the value $f^2_0 + \dots + f^2_{n-1}$ at the point x .

JACOBIAN

Scalar string specifying a user-supplied function to evaluate the x n Jacobian. The function accepts as parameter the point at which the Jacobian is to be evaluated and returns a two-dimensional matrix defined by $\text{result}(i, j) = \partial f_i / \partial x_j$.

ITMAX

Maximum allowable number of iterations. Default: ITMAX = 200.

XGUESS

Array with N components containing the initial estimate of the root. Default: XGUESS = 0.

Discussion

The IMSL_ZEROSYS function is based on the MINPACK subroutine HYBRDJ, which uses a modification of the hybrid algorithm due to M.J.D. Powell. This algorithm is a variation of Newton's Method, which takes precautions to avoid undesirable large steps or increasing residuals. For further discussion, see Moré et al. (1980).

Example

The following 2×2 system of nonlinear equations is solved:

$$f(x) = x_0 + x_1 - 3$$

$$f(x) = x_0^2 + x_1^2 - 9$$

```
.RUN
; Define the system through the function f.
FUNCTION f, x
    RETURN, [x(0)+x(1)-3, x(0)^2+x(1)^2-9]
END
```

```
PM, IMSL_ZEROSYS('f', 2), $
Title = 'Solution of the system:', FORMAT = '(f10.5)'
; Compute the solution and output the results.
Solution of the system:
    0.00000
    3.00000
```

Errors

Warning Errors

MATH_TOO_MANY_FCN_EVALS—Number of function evaluations has exceeded ITMAX. A new initial guess can be tried.

MATH_NO_BETTER_POINT—Keyword ERR_REL is too small. No further improvement in the approximate solution is possible.

MATH_NO_PROGRESS—Iteration has not made good progress. A new initial guess can be tried.

Version History

6.4	Introduced



Chapter 11

Optimization

This section contains the following topics:

Overview: Optimization	422	Optimization Routines	424

Overview: Optimization

This section introduces some of the mathematical concepts used with IDL Advanced Math and Stats.

Unconstrained Minimization

The unconstrained minimization problem can be stated as follows:

$$\min_{\mathbf{x} \in \mathbf{R}^n} f(\mathbf{x})$$

where $f: \mathbf{R}^n \rightarrow \mathbf{R}$ is continuous and has derivatives of all orders required by the algorithms. The functions for unconstrained minimization are grouped into three categories: univariate functions, multivariate functions, and nonlinear least-squares functions.

For the univariate functions, it is assumed that the function is unimodal within the specified interval. For discussion on unimodality, see Brent (1973).

A quasi-Newton method is used for the multivariate [IMSL_FMINV](#) function. The default is to use a finite-difference approximation of the gradient of $f(\mathbf{x})$. Here, the gradient is defined to be the following vector:

$$\nabla f(\mathbf{x}) = \left[\frac{\partial f(\mathbf{x})}{\partial x_1}, \frac{\partial f(\mathbf{x})}{\partial x_2}, \dots, \frac{\partial f(\mathbf{x})}{\partial x_n} \right]$$

When the exact gradient can be easily provided, the *grad* argument should be used.

The nonlinear least-squares function uses a modified Levenberg-Marquardt algorithm. The most common application of the function is the nonlinear data-fitting problem where the user is trying to fit the data with a nonlinear model.

These functions are designed to find only a local minimum point. However, a function may have many local minima. Try different initial points and intervals to obtain a better local solution.

Double-precision arithmetic is recommended for the functions when the user provides only the function values.

Linearly Constrained Minimization

The linearly constrained minimization problem can be stated as follows:

$$\min_{\mathbf{x} \in \mathbf{R}^n} f(\mathbf{x})$$

subject to:

$$A_1 x = b_1$$

$$A_2 x \geq b_2$$

where $f: R^n \rightarrow R$, A_1 and A_2 are coefficient matrices and b_1 and b_2 are vectors. If $f(x)$ is linear, then the problem is a linear programming problem; if $f(x)$ is quadratic, the problem is a quadratic programming problem.

The [IMSL_LINPROG](#) function uses a revised simplex method to solve small- to medium-sized linear programming problems. No sparsity is assumed since the coefficients are stored in full matrix form.

The [IMSL_QUADPROG](#) function is designed to solve convex quadratic programming problems using a dual quadratic programming algorithm. If the given Hessian is not positive definite, then IMSL_QUADPROG modifies it to be positive definite. In this case, output should be interpreted with care because the problem has been changed slightly. Here, the Hessian of $f(x)$ is defined to be the $n \times n$ matrix as follows:

$$\nabla^2 f(x) = \left[\frac{\partial^2}{\partial x_i \partial x_j} f(x) \right]$$

Nonlinearly Constrained Minimization

The nonlinearly constrained minimization problem can be stated as follows:

$$\min_{x \in R^n} f(x)$$

subject to:

$$g_i(x) = 0 \quad \text{for } i = 1, 2, \dots, m_1$$

$$g_i(x) \geq 0 \quad \text{for } i = m_1 + 1, \dots, m$$

where $f: R^n \rightarrow R$ and $g_i: R^n \rightarrow R$ for $i = 1, 2, \dots, m$.

The routine [IMSL_CONSTRAINED_NLP](#) uses a sequential equality constrained quadratic programming method. A more complete discussion of this algorithm is in [“IMSL_CONSTRAINED_NLP”](#) on page 465.

Optimization Routines

Unconstrained Minimization

IMSL_FMIN—(Univariate Function) Using function and possibly first derivative values.

IMSL_FMINV—(Multivariate Function) Using quasi-Newton method.

IMSL_NLINLSQ—(Nonlinear Least Squares) Using Levenberg-Marquardt algorithm.

Linearly Constrained Minimization

IMSL_LINPROG—Dense linear programming.

IMSL_QUADPROG—Quadratic programming.

Nonlinearly Constrained Minimization

IMSL_MINCONGEN—Minimize a general objective function.

IMSL_CONSTRAINED_NLP—Using a sequential equality constrained quadratic programming method.

IMSL_FMIN

The IMSL_FMIN function finds the minimum point of a smooth function $f(x)$ of a single variable using function evaluations and, optionally, through both function evaluations and first derivative evaluations.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_FMIN(f, a, b [, grad] [, /DOUBLE] [, ERR_ABS=value]
  [, ERR_REL=value] [, FVALUE=value] [, GVALUE=value]
  [, MAX_EVALS=value] [, STEP=value] [, TOL_GRAD=value]
  [, XGUESS=value])
```

Return Value

The point at which a minimum value of f is found. If no value can be computed, then NaN (Not a Number) is returned.

Arguments

f

Scalar string specifying a user-supplied function to compute the value of the function to be minimized. Function f accepts the point at which the function is to be evaluated and returns the computed function value at this point.

a

Lower endpoint of the interval in which the minimum point of f is to be located.

b

Upper endpoint of the interval in which the minimum point of f is to be located.

grad

Scalar string specifying a user-supplied function to compute the first derivative of the function. The *grad* function accepts the point at which the derivative is to be evaluated and returns the computed derivative at this point.

Keywords

DOUBLE

If present and nonzero, double precision is used.

ERR_ABS

Required absolute accuracy in the final value of x . On a normal return, there are points on either side of x within a distance `ERR_ABS` at which f is no less than f at x . The keyword `ERR_ABS` cannot be used if the optional argument *grad* is supplied. Default: `ERR_ABS = 0.0001`.

ERR_REL

Required relative accuracy in the final value of x . This is the first stopping criterion. On a normal return, the solution x is in an interval that contains a local minimum and is less than or equal to $\max(1.0, |x|) * \text{ERR\_REL}$. When the given `ERR_REL` is less than zero, $\text{SQRT}(\epsilon)$ is used as `ERR_REL`, where ϵ is the machine precision. The keyword `ERR_REL` can only be used if the optional argument *grad* is supplied. Default: `ERR_REL = SQRT( $\epsilon$ )`.

FVALUE

Function value at point x . The keyword `FVALUE` can only be used if the optional argument *grad* is supplied.

GVALUE

Derivative value at point x . The keyword `GVALUE` can only be used if the optional argument *grad* is supplied.

MAX_EVALS

Maximum number of function evaluations allowed. Default: `MAX_EVALS = 1000`.

STEP

Order of magnitude estimate of the required change in x . The keyword STEP cannot be used if the optional argument *grad* is supplied. Default: STEP = 1.0

TOL_GRAD

Derivative tolerance used to decide if the current point is a local minimum. This is the second stopping criterion. Parameter x is returned as a solution when *grad* is less than or equal to TOL_GRAD. The keyword TOL_GRAD should be nonnegative; otherwise, zero is used. The keyword TOL_GRAD can only be used if the optional argument *grad* is supplied. Default: TOL_GRAD = SQRT(ϵ), where ϵ is the machine precision.

XGUESS

Initial guess of the minimum point of f . Default: XGUESS = $(a + b)/2$

Discussion

The IMSL_FMIN function uses a safeguarded, quadratic interpolation method to find a minimum point of a univariate function. Both the code and the underlying algorithm are based on the subroutine ZXLSF written by M.J.D. Powell at the University of Cambridge.

The IMSL_FMIN function finds the least value of a univariate function, f , which is specified by the function f . (Other required data are two points A and B that define an interval for finding a minimum point from an initial estimate of the solution, x_0 , where $x_0 = \text{XGUESS}$.) The algorithm begins the search by moving from x_0 to $x = x_0 + s$, where $s = \text{STEP}$ is an estimate of the required change in x and may be positive or negative. The first two function evaluations indicate the direction to the minimum point, and the search strides out along this direction until a bracket on a minimum point is found or until x reaches one of the endpoints a or b . During this stage, the step length increases by a factor of between 2 and 9 per function evaluation. The factor depends on the position of the minimum point that is predicted by quadratic interpolation of the three most recent function values.

When an interval containing a solution has been found, the three points are as follows:

$$x_1, x_2, x_3, \text{ with } x_1 < x_2 < x_3, \quad f(x_1) \geq f(x_2), \text{ and } f(x_2) \geq f(x_3)$$

Considered the following rules when choosing the new x from these three points:

- The estimate of the minimum point that is given by quadratic interpolation of the three function values
- A tolerance parameter η , which depends on the closeness of $|f|$ to a quadratic
- Whether x_2 is near the center of the range between x_1 and x_3 or is relatively close to an end of this range

In outline, the value of x is as near as possible to predicted minimum point, subject to being at least ϵ from x_2 and subject to being in the longer interval between x_1 and x_2 or x_2 and x_3 , when x_2 is close to x_1 or x_3 .

The algorithm is intended to provide fast convergence when f has a positive and continuous second derivative at the minimum and to avoid gross inefficiencies in pathological cases, such as the following:

$$f(x) = x + 1.001 |x|$$

The algorithm can automatically make ϵ large in the pathological cases. In this case, it is usual for a new value of x to be at the midpoint of the longer interval that is adjacent to the least calculated function value. The midpoint strategy is used frequently when changes to f are dominated by computer rounding errors, which happens if the user requests an accuracy that is less than the square root of the machine precision. In such cases, the subroutine claims to have achieved the required accuracy if it decides that there is a local minimum point within distance δ of x , where $\delta = \text{ERR\_ABS}$, even though the rounding errors in f may cause the existence of other local minimum points nearby. This difficulty is inevitable in minimization routines that use only function values, so high-precision arithmetic is recommended.

If the argument *grad* is supplied, then the IMSL_FMIN function uses a descent method with either the secant method or cubic interpolation to find a minimum point of a univariate function. It starts with an initial guess and two endpoints. If any of the three points is a local minimum point and has least function value, the function terminates with a solution; otherwise, the point with least function value is used as the starting point.

From the starting point, for example x_c , the function value $f_c = f(x_c)$, the derivative value $g_c = g(x_c)$, and a new point x_n , defined by $x_n = x_c - g_c$, are computed. The function $f_n = f(x_n)$ and the derivative $g_n = g(x_n)$ are then evaluated. If either $f_n \geq f_c$ or g_n has the opposite sign of g_c , then a minimum point exists between x_c and x_n , and an initial interval is obtained; otherwise, since x_c is kept as the point that has lowest function value, an interchange between x_n and x_c is performed. The secant method is then used to get a new point:

$$x_s = x_c - g_c \left(\frac{g_n - g_c}{x_n - x_c} \right)$$

Let $x_n < x_s$. Repeat this process until an interval containing a minimum is found or one of the following convergence criteria is satisfied:

Criterion 1: $|x_c - x_n| \leq \epsilon_c$

Criterion 2: $|g_c| \leq \epsilon_g$

where $\epsilon_c = \max \{1.0, |x_c|\} * \epsilon$, ϵ is a relative error tolerance and ϵ_g is a gradient tolerance.

When convergence is not achieved, a cubic interpolation is performed to obtain a new point. The function and derivative are then evaluated at that point; accordingly, a smaller interval that contains a minimum point is chosen. A safeguarded method is used to ensure that the interval be reduced by at least a fraction of the previous interval. Another cubic interpolation is then performed, and this function is repeated until one of the stopping criteria is met.

Examples

Example 1

This example finds a minimum point of $f(x) = e^x - 5x$. The results are shown in [Figure 11-1](#).

```
.RUN
; Define the function to be used.
FUNCTION f, x
    RETURN, EXP(x) - 5 * x
END

xmin = IMSL_FMIN('f', -100, 100)
; Call IMSL_FMIN to compute the minimum.
PM, xmin
; Print results.
1.60943
```

```

x = 10 * FINDGEN(100)/99 - 5
!P.Font = 0
PLOT, x, f(x), Title = '!8f(x) = e!Ex!N-5x!3', XTitle = 'x', $
    YTitle = 'f(x)'
; Plot results.
OPLOT, [xmin], [f(xmin)], Psym = 6
str = '(' + STRCOMPRESS(xmin) + ', ' + STRCOMPRESS(f(xmin)) + ' )'
OPLOT, [xmin], [f(xmin)], Psym = 6
XYOUTS, -5, 80, 'Minimum point:!C' + str, Charsize = 1.2

```

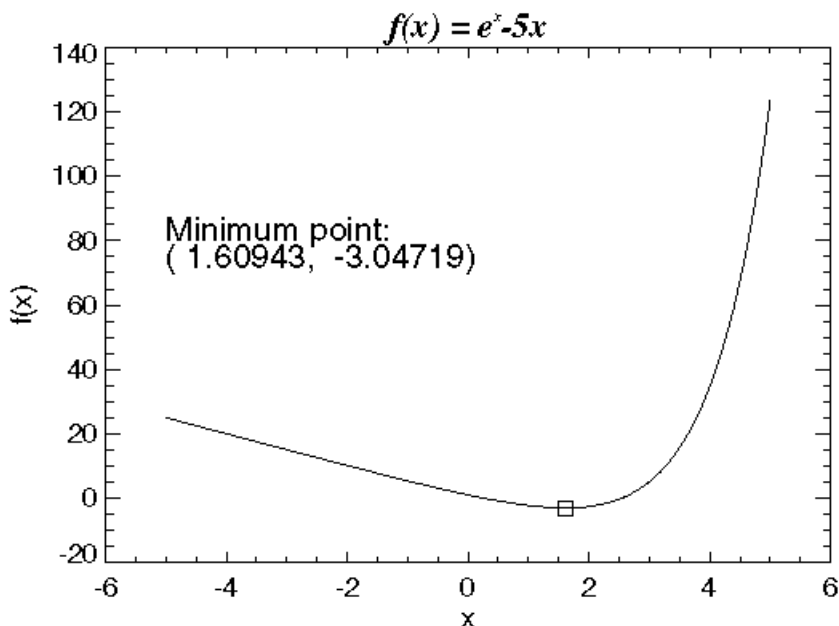


Figure 11-1: Minimum Point of a Smooth Function

Example 2

This example supplies the *grad* argument and finds a minimum point of $f(x) = x(x^3 - 1) + 10$ with an initial guess $x_0 = 3$. The results are shown in [Figure 11-2](#).

```

.RUN
FUNCTION f, x
    RETURN, x * (x^3 - 1) + 10
END

```

```
.RUN
FUNCTION grad, x
    RETURN, 4 * x^3 - 1
END

xmin = IMSL_FMIN('f', -10, 10, 'grad')
x = 4 * FINDGEN(100)/99 - 2
PLOT, x, f(x), Title = '!8f(x) = x(x!E3!N-1)+10!3', $
    XTitle = 'x', YTitle = 'f(x)'
OPLOT, [xmin], [f(xmin)], Psym = 6
str = '(' + STRCOMPRESS(xmin) + ', ' + STRCOMPRESS(f(xmin)) + ')'
XYOUTS, -1.5, 25, 'Minimum point:'+str, Charsize = 1.2
```

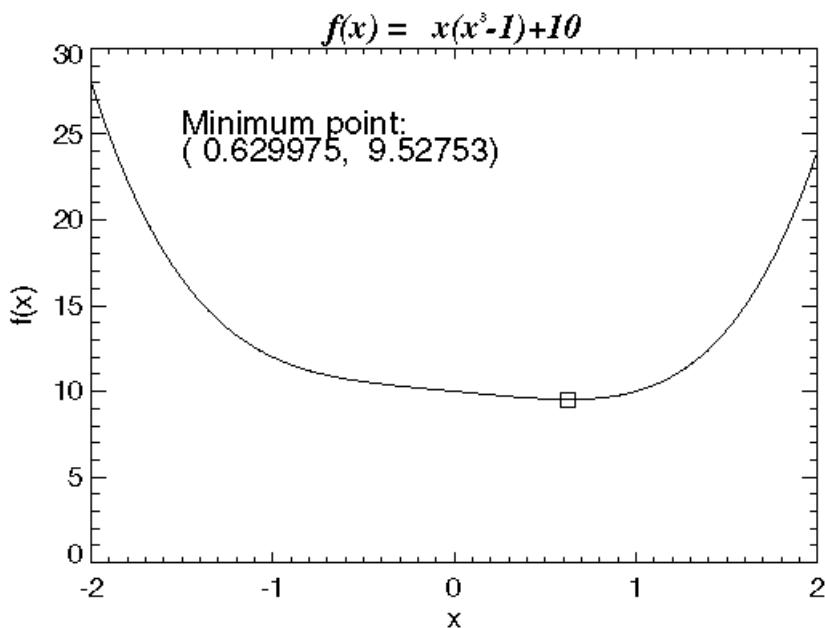


Figure 11-2: Minimum Point of a Smooth Function

Errors

Warning Errors

MATH_MIN_AT_LOWERBOUND—Final value of x is at the lower bound.

- MATH_MIN_AT_UPPERBOUND—Final value of x is at the upper bound.
- MATH_MIN_AT_BOUND—Final value of x is at a bound.
- MATH_NO_MORE_PROGRESS—Computer rounding errors prevent further refinement of x .
- MATH_TOO_MANY_FCN_EVAL—Maximum number of function evaluations exceeded.

Version History

6.4	Introduced

IMSL_FMINV

The IMSL_FMINV function minimizes a function $f(x)$ of n variables using a quasi-Newton method.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_FMINV(f, n [, /DOUBLE] [, GRAD=string] [, FSCALE=string]
[, FVALUE=variable] [, IHESS=parameter] [, ITMAX=value]
[, MAX_EVALS=value] [, MAX_GRAD=value] [, MAX_STEP=value]
[, N_DIGIT=value] [, TOL_GRAD=value] [, TOL_RFCN=value]
[, TOL_STEP=value] [, XGUESS=array] [, XSCALE=array])
```

Return Value

The minimum point x of the function. If no value can be computed, NaN is returned.

Arguments

f

Scalar string specifying a user-supplied function to evaluate the function to be minimized. Function f accepts the point at which the function is evaluated and returns the computed function value at the point.

n

Number of variables.

Keywords

DOUBLE

If present and nonzero, double precision is used.

GRAD

Scalar string specifying a user-supplied function to compute the gradient. The GRAD function accepts the point at which the gradient is evaluated and returns the computed gradient at the point.

FSCALE

Scalar containing the function scaling. The keyword FSCALE is used mainly in scaling the gradient. See the keyword TOL_GRAD for more detail. Default: FSCALE = 1.0.

FVALUE

Name of a variable into which the value of the function at the computed solution is stored.

IHESS

Hessian initialization parameter. If IHESS is zero, the Hessian is initialized to the identity matrix; otherwise, it is initialized to a diagonal matrix containing $\max(f(t), f_s) * s_i$ on the diagonal, where $t = XGUESS$, $f_s = FSCALE$, and $s = XSCALE$. Default: IHESS = 0.

ITMAX

Maximum number of iterations. Default: ITMAX = 100.

MAX_EVALS

Maximum number of function evaluations. Default: MAX_EVALS = 400.

MAX_GRAD

Maximum number of gradient evaluations. Default: MAX_GRAD = 400.

MAX_STEP

Maximum allowable step size. Default: MAX_STEP = $1000\max(\epsilon_1, \epsilon_2)$, where:

$$\epsilon_1 = \sqrt{\sum_{i=1}^n (s_i t_i)^2}$$

$$\epsilon_2 = \|s\|_2, s = XSCALE, \text{ and } t = XGUESS$$

N_DIGIT

Number of good digits in function. Default: machine dependent.

TOL_GRAD

Scaled gradient tolerance.

The i -th component of the scaled gradient at x is calculated as:

$$\frac{|g_i| \times \max(|x_i|, 1/s_i)}{\max(|f(x)|, f_s)}$$

where:

$$g = \nabla f(x)$$

$s = \text{XSCALE}$, and $f_s = \text{FSCALE}$. Default: $\text{TOL\_GRAD} = \epsilon^{1/2}$ ($\epsilon^{1/3}$ in double) where ϵ is the machine precision.

TOL_RFCN

Relative function tolerance. Default:

$\text{TOL\_RFCN} = \max(10^{-10}, \epsilon^{2/3}), \max(10^{-20}, \epsilon^{2/3})$ in double.

TOL_STEP

Scaled step tolerance.

The i -th component of the scaled step between two points x and y is computed as:

$$\frac{|x_i - y_i|}{\max(|x_i|, 1/s_i)}$$

where $s = \text{XSCALE}$. Default: $\text{TOL\_STEP} = \epsilon^{2/3}$

XGUESS

Array with n components containing an initial guess of the computed solution.

Default: $\text{XGUESS} (*) = 0$.

XSCALE

Array with n components containing the scaling vector for the variables. The keyword XSCALE is used mainly in scaling the gradient and the distance between

two points (see the keywords TOL_GRAD and TOL_STEP for more detail). Default: XSCALE (*) = 1.0.

Discussion

The IMSL_FMINV function uses a quasi-Newton method to find the minimum of a function $f(x)$ of n variables. The problem is stated below:

$$\min_{x \in \mathbb{R}^n} f(x)$$

Given a starting point x_c , the search direction is computed according to the formula:

$$d = -B^{-1}g_c$$

where B is a positive definite approximation of the Hessian and g_c is the gradient evaluated at x_c .

A line search is then used to find a new point:

$$x_n = x_c + \lambda d, \lambda > 0$$

such that:

$$f(x_n) \leq f(x_c) + \alpha g^T d$$

where $\alpha \in (0, 0.5)$.

Finally, the optimality condition:

$$\|g(x)\| \leq \epsilon$$

is checked, where ϵ is a gradient tolerance.

When optimality is not achieved, B is updated according to the BFGS formula:

$$B \leftarrow B - \frac{B s s^T B}{s^T B s} + \frac{y y^T}{y^T s}$$

where $s = x_n - x_c$ and $y = g_n - g_c$. Another search direction is then computed to begin the next iteration. For more details, see Dennis and Schnabel (1983, Appendix A).

In this implementation, the first stopping criterion for IMSL_FMINV occurs when the norm of the gradient is less than the given gradient tolerance TOL_GRAD. The second stopping criterion for IMSL_FMINV occurs when the scaled distance between the last two steps is less than the step tolerance TOL_STEP.

Since by default, a finite-difference method is used to estimate the gradient for some single-precision calculations, an inaccurate estimate of the gradient may cause the

algorithm to terminate at a noncritical point. In such cases, high-precision arithmetic is recommended or keyword GRAD is used to provide more accurate gradient evaluation.

Examples

Example 1

The function $f(x) = 100 (x_2 - x_1^2)^2 + (1 - x_1)^2$ is minimized.

```
.RUN
; Define the function.
FUNCTION f, x
    xn = x
    xn(0) = x(1) - x(0)^2
    xn(1) = 1 - x(0)
    RETURN, 100 * xn(0)^2 + xn(1)^2
END

xmin = IMSL_FMINV('f', 2)
; Call IMSL_FMINV to compute the minimum.
PM, xmin, Title = 'Solution:'
; Output the solution.
Solution:
    0.999986
    0.999971
PM, f(xmin), Title = 'Function value:'
Function value:
    2.09543e-10
```

Example 2

The function $f(x) = 100 (x_2 - x_1^2)^2 + (1 - x_1)^2$ is minimized with the initial guess $x = (-1.2, 1.0)$. In the following plot, the asterisk marks the minimum. The results are shown in [Figure 11-3](#).

```
.RUN
; Define the function.
FUNCTION f, x
    xn = x
    xn(0) = x(1) - x(0)^2
    xn(1) = 1 - x(0)
    RETURN, 100 * xn(0)^2 + xn(1)^2
END

.RUN
; Define the gradient function.
FUNCTION grad, x
```

```

g = x
g(0) = -400 * (x(1) - x(0)^2) * x(0) - 2 * (1 - x(0))
g(1) = 200 * (x(1) - x(0)^2)
RETURN, g
END

xmin = IMSL_FMINV('f', 2, grad = 'grad', $
  XGuess = [-1.2, 1.0], Tol_Grad = .0001)
; Call IMSL_FMINV with the gradient function, an initial guess,
; and a scaled gradient tolerance.
x = 4 * FINDGEN(100)/99 - 2
y = x
surf = FLTARR(100, 100)
FOR i = 0, 99 DO FOR j = 0, 99 do $
  surf(i, j) = f([x(i), y(j)])
  ; Evaluate function f on 100 x 100 grid for use in CONTOUR.
  str = '(' + STRCOMPRESS(xmin(0)) + ',' + $
  STRCOMPRESS(xmin(1)) + ',' + STRCOMPRESS(f(xmin)) + ')'
  !P.Charsize = 1.5
  CONTOUR, surf, x, y, Levels = [20*FINDGEN(6), $
    500 + FINDGEN(7)*500], /C_Annotation, $
    Title='!18Rosenbrock Function!C' + 'Minimum Point:!C' + $
    str, Position = [.1, .1, .8, .8]
; Call CONTOUR. Customize the contour plot, including the title
; of the plot.
OPLOT, [xmin(0)], [xmin(1)], Psym = 2, Symsize = 2
; Plot the solution as an asterisk.

```

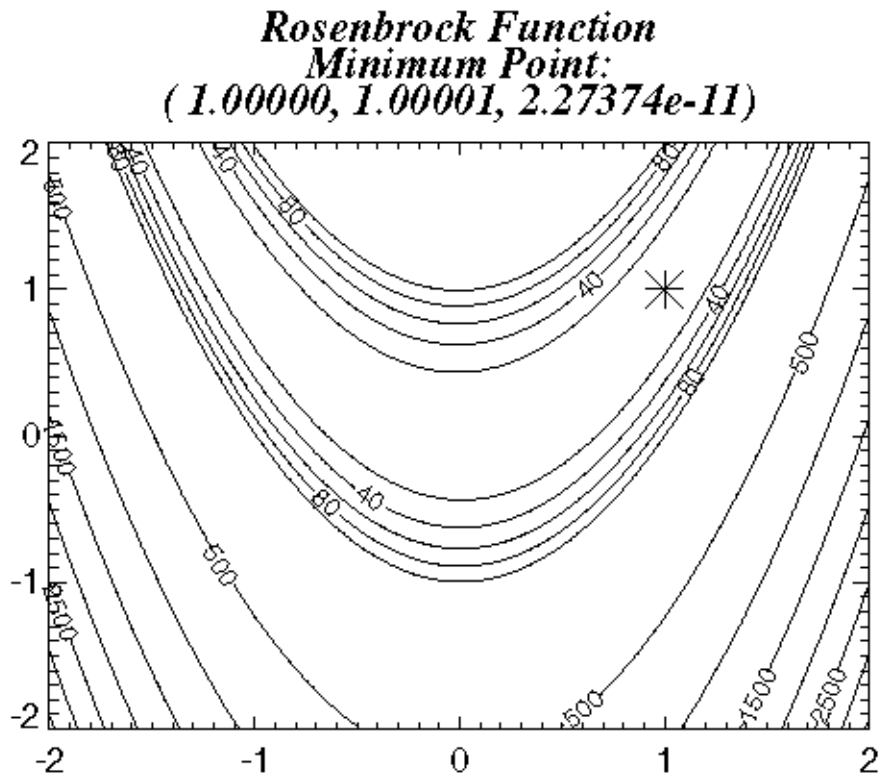


Figure 11-3: Rosenbrock Function Plot

Errors

Informational Errors

MATH_STEP_TOLERANCE—Scaled step tolerance satisfied. Current point may be an approximate local solution, but it is also possible that the algorithm is making very slow progress and is not near a solution or that **TOL_STEP** is too big.

Warning Errors

MATH_REL_FCN_TOLERANCE—Relative function convergence. Both the actual and predicted relative reductions in the function are less than or equal to the relative function convergence tolerance.

MATH_TOO_MANY_ITN—Maximum number of iterations exceeded.

MATH_TOO_MANY_FCN_EVAL—Maximum number of function evaluations exceeded.

MATH_TOO_MANY_GRAD_EVAL—Maximum number of gradient evaluations exceeded.

MATH_UNBOUNDED—Five consecutive steps have been taken with the maximum step length.

MATH_NO_FURTHER_PROGRESS—Last global step failed to locate a point lower than the current x value.

Fatal Errors

MATH_FALSE_CONVERGENCE—Iterates appear to converge to a noncritical point. It is possible that incorrect gradient information is used, or the function is discontinuous, or the other stopping tolerances are too tight.

Version History

6.4	Introduced

IMSL_NLINLSQ

The IMSL_NLINLSQ function solves a nonlinear least-squares problem using a modified Levenberg-Marquardt algorithm.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_NLINLSQ(f, m, n [, xlb] [, xub] [, /DOUBLE] [, FJAC=variable]
    [, FSCALE=array] [, FVEC=variable] [, INTERN_SCALE=variable]
    [, ITMAX=value] [, JACOBIAN=string] [, JTJ_INVERSE=variable]
    [, MAX_EVALS=value] [, MAX_JACOBIAN=value] [, MAX_STEP=value]
    [, N_DIGITS=value] [, RANK=value] [, TOL_AFCN=value]
    [, TOL_GRAD=value] [, TOL_RFCN=value] [, TOLERANCE=value]
    [, TRUST_REGION=value] [, XGUESS=array] [, XSCALE=array] )
```

Return Value

The solution *x* of the nonlinear least-squares problem. If no solution can be computed, NULL is returned.

Arguments

f

Scalar string specifying a user-supplied function to evaluate the function that defines the least-squares problem. Function *f* accepts the following two parameters and returns an array of length *m* containing the function values at *x*:

m—Number of functions.

x—Array length *n* containing the point at which the function is evaluated.

m

Number of functions.

n

Number of variables where $n \leq m$.

xlb****

One dimensional array with n components containing the lower bounds on the variables. The *xl**b*** and *xub* arguments must be used together.

xub

One dimensional array with n components containing the upper bounds on the variables. The *xl**b*** and *xub* arguments must be used together.

Keywords

DOUBLE

If present and nonzero, double precision is used.

FJAC

Name of the variable into which an array of size $m \times n$ containing the Jacobian at the approximate solution is stored.

FSCALE

Array with m components containing the diagonal scaling matrix for the functions. The i -th component of FSCALE is a positive scalar specifying the reciprocal magnitude of the i -th component function of the problem. Default: FSCALE (*) = 1.

FVEC

Name of the variable into which a real array of length m containing the residuals at the approximate solution is stored.

INTERN_SCALE

Internal variable scaling option. With this keyword, the values for XSCALE are set internally.

ITMAX

Maximum number of iterations. Default: ITMAX = 100.

JACOBIAN

Scalar string specifying a user-supplied function to compute the Jacobian. This function accepts two parameters and returns an $n \times m$ array containing the Jacobian at the point s input point. Note that each derivative $\partial f_i / \partial x_j$ should be returned in the (i, j) element of the returned matrix. The parameters of the function are as follows:

m —Number of equations.

x —Array of length n at which the point Jacobian is evaluated.

JTJ_INVERSE

Name of the variable into which an array of size $n \times n$ containing the inverse matrix of $J^T J$, where J is the final Jacobian, is stored. If $J^T J$ is singular, the inverse is a symmetric g_2 inverse of $J^T J$. (See “[IMSL_CHNNDSOL](#)” on page 108 for a discussion of generalized inverses and the definition of the g_2 inverse.)

MAX_EVALS

Maximum number of function evaluations. Default: $\text{MAX\_EVALS} = 400$.

MAX_JACOBIAN

Maximum number of Jacobian evaluations. Default: $\text{MAX\_JACOBIAN} = 400$.

MAX_STEP

Maximum allowable step size. Default: $\text{MAX\_STEP} = 1000 \max(\epsilon_1, \epsilon_2)$, where:

$$\epsilon_1 = \sqrt{\sum_{i=1}^n s_i t_i^2}$$

$$\epsilon_2 = \|s\|_2$$

$s = \text{XSCALE}$, and $t = \text{XGUESS}$

N_DIGITS

Number of good digits in the function. Default: machine dependent.

RANK

Name of the variable into which the rank of the Jacobian is stored.

TOL_AFCN

Absolute function tolerance. Default: $\text{TOL\_AFCN} = \max(10^{-20}, \epsilon^2)$, $[\max(10^{-40}, \epsilon^2)$ in double], where ϵ is the machine precision.

TOL_GRAD

Scaled gradient tolerance.

The i -th component of the scaled gradient at x is calculated as:

$$\frac{|g_i| \times \max(|x_i|, 1/s_i)}{\frac{1}{2}\|F(x)\|_2^2}$$

where $g = \nabla F(x)$, $s = XScale$, and:

$$\|F(x)\|_2^2 = \sum_{i=1}^m f_i(x)^2$$

Default: $\text{TOL\_GRAD} = \epsilon^{1/2}$ ($\epsilon^{1/3}$ in double), where ϵ is the machine precision.

TOL_RFCN

Relative function tolerance. Default: $\text{TOL\_RFCN} = \max(10^{-10}, \epsilon^{2/3})$, $[\max(10^{-40}, \epsilon^{2/3})$ in double], where ϵ is the machine precision.

TOLERANCE

Tolerance used in determining linear dependence for the computation of the inverse of $J^T J$. If the keyword JACOBIAN is specified, the default is $\text{TOLERANCE} = 100\epsilon$, where ϵ is the machine precision; otherwise, the default is $\text{SQRT}(\epsilon)$, where ϵ is the machine precision.

TRUST_REGION

Size of initial trust-region radius. Default: based on the initial scaled Cauchy step.

XGUESS

Array with n components containing an initial guess. Default: $\text{XGUESS} (*) = 0$.

Tol_Step—Scaled step tolerance.

The i -th component of the scaled step between two points x and y is computed as:

$$\frac{|x_i - y_y|}{\max(|x_i|, 1/s_i)}$$

where $s = \text{XSCALE}$.

Default: $\text{Tot\_Step} = \epsilon^{2/3}$, where ϵ is the machine precision

XSCALE

Array with n components containing the scaling vector for the variables. The keyword XSCALE is used mainly in scaling the gradient and the distance between two points (see the keywords TOL_GRAD and TOL_STEP for more detail). Default: $\text{XSCALE} (*) = 1$.

Discussion

The specific algorithm used in `IMSL_NLINLSQ` is dependent on whether the keywords XLB and XUB are supplied. If the keywords XLB and XUB are not supplied, then the `IMSL_NLINLSQ` function is based on the MINPACK routine LMDER by Moré et al. (1980).

The `IMSL_NLINLSQ` function, based on the MINPACK routine LMDER by Moré et al. (1980), uses a modified Levenberg-Marquardt method to solve nonlinear least-squares problems. The problem is stated as follows:

$$\min \frac{1}{2} F(x)^T F(x) = \frac{1}{2} \sum_{i=1}^m f_i(x)^2$$

where $m \geq n$, $F : R^n \rightarrow R^m$ and $f_i(x)$ is the i -th component function of $F(x)$. From a current point, the algorithm uses the trust region approach:

$$\min_{x \in R^n} \|F(x_c) + J(x_c)(x_n - x_c)\|_2$$

$$\text{subject to } \|x_n - x_c\|_2 \leq \delta_c$$

to get a new point x_n . Compute x_n as:

$$x_n = x_c - (J(x_c)^T J(x_c) + \mu_c I)^{-1} J(x_c)^T F(x_c)$$

where $\mu_c = 0$ if $\delta_c \geq \| (J(x_c)^T J(x_c))^{-1} J(x_c)^T F(x_c) \|_2$ and $\mu_c > 0$ otherwise.

The value μ_c is defined by the function. The vector and matrix $F(x_c)$ and $J(x_c)$ are the function values and the Jacobian evaluated at the current point x_c . This function is repeated until the stopping criteria are satisfied.

The first stopping criterion for IMSL_NLINLSQ occurs when the norm of the function is less than the absolute function tolerance, TOL_AFCN. The second stopping criterion occurs when the norm of the scaled gradient is less than the given gradient tolerance TOL_GRAD. The third stopping criterion for IMSL_NLINLSQ occurs when the scaled distance between the last two steps is less than the step tolerance TOL_STEP. For more details, see Levenberg (1944), Marquardt (1963), or Dennis and Schnabel (1983, Chapter 10).

If the keywords XLB and XUB are supplied, then the IMSL_NLINLSQ function uses a modified Levenberg-Marquardt method and an active set strategy to solve nonlinear least-squares problems subject to simple bounds on the variables. The problem is stated as follows:

$$\min \frac{1}{2} F(x)^T F(x) = \frac{1}{2} \sum_{i=1}^m f_i(x)^2$$

subject to $l \leq x \leq u$ where $m \geq n$, $F: \mathbf{R}^n \rightarrow \mathbf{R}^m$, and $f_i(x)$ is the i -th component function of $F(x)$. From a given starting point, an active set IA, which contains the indices of the variables at their bounds, is built. A variable is called a “free variable” if it is not in the active set. The routine then computes the search direction for the free variables according to the formula:

$$d = -(J^T J + \mu I)^{-1} J^T F$$

where μ is the Levenberg-Marquardt parameter, $F = F(x)$, and J is the Jacobian with respect to the free variables. The search direction for the variables in IA is set to zero. The trust region approach discussed by Dennis and Schnabel (1983) is used to find the new point. Finally, the optimality conditions are checked. The conditions are:

$$||g(x_i)|| \leq \epsilon, l_i < x_i < u_i$$

$$g(x_i) < 0, x_i = u_i$$

$$g(x_i) > 0, x_i = l_i$$

where ϵ is a gradient tolerance. This process is repeated until the optimality criterion is achieved.

The active set is changed only when a free variable hits its bounds during an iteration or the optimality condition is met for free variables but not for all variables in IA, the active set. In the latter case, a variable that violates the optimality condition will be dropped out of IA. For more detail on the Levenberg-Marquardt method, see Levenberg (1944) or Marquardt (1963). For more detail on the active set strategy, see Gill and Murray (1976).

Since a finite-difference method is used to estimate the Jacobian for some single-precision calculations, an inaccurate estimate of the Jacobian may cause the algorithm to terminate at a noncritical point. In such cases, high-precision arithmetic is recommended. Also, whenever the exact Jacobian can be easily provided, the keyword JACOBIAN should be used.

Example

In this example, the nonlinear data-fitting problem found in Dennis and Schnabel (1983, p. 225):

$$\min \frac{1}{2} \sum_{i=0}^3 f_i(x)^2 \quad \text{where } f_i(x) = e^{t_i x} - y_i$$

is solved with the data $t = [1, 2, 3]$ and $y = [2, 4, 3]$.

```
.RUN
; Define the function that defines the least-squares problem.
FUNCTION f, m, x
    y = [2, 4, 3]
    t = [1, 2, 3]
    RETURN, EXP(x(0) * t) - y
END

solution = IMSL_NLINLSQ('f', 3, 1)
; Call IMSL_NLINLSQ.
PM, solution, Title = 'The solution is:'

; Output the results.
The solution is:
    0.440066
PM, f(m, solution), Title = 'The function values are:'

The function values are:
    -0.447191
    -1.58878
    0.744159
```

Errors

Informational Errors

MATH_STEP_TOLERANCE—Scaled step tolerance satisfied. The current point may be an approximate local solution, but it is also possible that the algorithm is making very slow progress and is not near a solution or that TOL_STEP is too big.

Warning Errors

- MATH_LITTLE_FCN_CHANGE—Both the actual and predicted relative reductions in the function are less than or equal to the relative function tolerance.
- MATH_TOO_MANY_ITN—Maximum number of iterations exceeded.
- MATH_TOO_MANY_FCN_EVAL—Maximum number of function evaluations exceeded.
- MATH_TOO_MANY_JACOBIAN_EVAL—Maximum number of Jacobian evaluations exceeded.
- MATH_UNBOUNDED—Five consecutive steps have been taken with the maximum step length.

Fatal Errors

- MATH_FALSE_CONVERGE—Iterates appear to be converging to a noncritical point.

Version History

6.4	Introduced

IMSL_LINPROG

The IMSL_LINPROG function solves a linear programming problem using the revised simplex algorithm.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_LINPROG(a, b, c [, BU=array] [, /DOUBLE] [, DUAL=variable]
    [, IRTYPE=array] [, ITMAX=value] [, OBJ=variable] [, XLB=array]
    [, XUB=array])
```

Return Value

The solution x of the linear programming problem.

Arguments

a

Two-dimensional matrix containing the coefficients of the constraints. The coefficient for the i -th constraint is contained in $A(i, *)$.

b

One-dimensional matrix containing the right-hand side of the constraints. If there are limits on both sides of the constraints, b contains the lower limit of the constraints.

c

One-dimensional array containing the coefficients of the objective function.

Keywords

BU

Array with N_ELEMENTS(*b*) elements containing the upper limit of the constraints that have both the lower and the upper bounds. If no such constraint exists, BU is not needed.

DOUBLE

If present and nonzero, double precision is used.

DUAL

Name of the variable into which the array with N_ELEMENTS(*c*) elements, containing the dual solution, is stored.

IRTYPE

Array with N_ELEMENTS(*b*) elements indicating the types of general constraints in the matrix *A*. Let $r_i = A_{i0}x_0 + \dots + A_{in-1}x_{n-1}$. The value of IRTYPE (*i*) is described in [Table 11-1](#).

Irtype (i)	Constraints
0	$r_i = b_i$
1	$r_i \leq bu$
2	$r_i \geq b_i$
3	$b_i \leq r_i \leq bu$

Table 11-1: Constraint Types

Default: IRTYPE (*) = 0

ITMAX

Maximum number of iterations. Default: ITMAX = 10,000

OBJ

Name of the variable into which the optimal value of the objective function is stored.

XLB

Array with N_ELEMENTS(c) elements containing the lower bound on the variables. If there is no lower bound on a variable, 10^{30} should be set as the lower bound. Default: XLB (*) = 0

XUB

Array with N_ELEMENTS(c) elements containing the upper bound on the variables. If there is no upper bound on a variable, -10^{30} should be set as the upper bound. Default: XUB (*) = *infinity*

Discussion

The IMSL_LINPROG function uses a revised simplex method to solve linear programming problems; i.e., problems of the form:

$$\min_{x \in \mathbf{R}^n} \mathbf{c}^T \mathbf{x}$$

subject to:

$$\mathbf{b}_l \leq \mathbf{A}_x \leq \mathbf{b}_u$$

$$\mathbf{x}_l \leq \mathbf{x} \leq \mathbf{x}_u$$

where $\mathbf{c}$ is the objective coefficient vector, $\mathbf{A}$ is the coefficient matrix, and the vectors $\mathbf{b}_l$, $\mathbf{b}_u$, $\mathbf{x}_l$, and $\mathbf{x}_u$ are the lower and upper bounds on the constraints and the variables.

For a complete discussion of the revised simplex method, see Murtagh (1981) or Murty (1983). This problem can be solved more efficiently.

Example

In this example, the linear programming problem in the standard form:

$$\min f(x) = -x_0 - 3x_1$$

subject to:

is solved.

```
RM, a, 4, 6
; Define the coefficients of the constraints.
row 0: 1 1 1 0 0 0
row 1: 1 1 0 -1 0 0
row 2: 1 0 0 0 1 0
row 3: 0 1 0 0 0 1
```

$$\begin{aligned}
 x_0 + x_1 + x_2 &= 1.5 \\
 x_0 + x_1 - x_3 &= 0.5 \\
 x_0 + x_4 &= 1.0 \\
 x_1 + x_5 &= 1.0 \\
 x_i &\geq 0, \text{ for } i = 0, \dots, 5
 \end{aligned}$$

```

RM, b, 4, 1
; Define the right-hand side of the constraints.
row 0: 1.5
row 1: .5
row 2: 1
row 3: 1
RM, c, 6, 1
; Define the coefficients of the objective function.
row 0: -1
row 1: -3
row 2: 0
row 3: 0
row 4: 0
row 5: 0
PM, IMSL_LINPROG(a, b, c), Title = 'Solution'
; Call IMSL_LINPROG and print the solution.
Solution
0.500000
1.00000
0.00000
1.00000
0.500000
0.00000

```

Errors

Warning Errors

MATH_PROB_UNBOUNDED—Problem is unbounded.

MATH_TOO_MANY_ITN—Maximum number of iterations exceeded.

MATH_PROB_INFEASIBLE—Problem is infeasible.

Fatal Errors

MATH_NUMERIC_DIFFICULTY—Numerical difficulty occurred. If *float* is currently being used, using *double* may help.

MATH_BOUNDS_INCONSISTENT—Bounds are inconsistent.

Version History

6.4	Introduced

IMSL_QUADPROG

The IMSL_QUADPROG function solves a quadratic programming (QP) problem subject to linear equality or inequality constraints.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_QUADPROG(a, b, g, h [, DIAG=variable] [, /DOUBLE]  
[, DUAL=variable] [, MEQ=value] [, OBJ=variable])
```

Return Value

The solution x of the QP problem.

Arguments

a

Two-dimensional matrix containing the linear constraints.

b

One-dimensional matrix of the right-hand sides of the linear constraints.

g

One-dimensional array of the coefficients of the linear term of the objective function.

h

Two-dimensional array of size $N\_ELEMENTS(g) \times N\_ELEMENTS(g)$ containing the Hessian matrix of the objective function. It must be symmetric positive definite. If h is not positive definite, the algorithm attempts to solve the QP problem with h replaced by $h + Diag * I$, such that $h + Diag * I$ is positive definite.

Keywords

DIAG

Name of the variable into which the scalar, equal to the multiple of the identity matrix added to h to give a positive definite matrix, is stored.

DOUBLE

If present and nonzero, double precision is used.

DUAL

Name of the variable into which an array with $N\_ELEMENTS(g)$ elements, containing the Lagrange multiplier estimates, is stored.

MEQ

Number of linear equality constraints. If MEQ is used, then the equality constraints are located at $a(i, *)$ for $i = 0, \dots, Meq - 1$.

Default: $MEQ = N\_ELEMENTS(a(*, 0))$ n ; i.e., all constraints are equality constraints.

OBJ

Name of variable into which optimal object function found is stored.

Discussion

The IMSL_QUADPROG function is based on M.J.D. Powell's implementation of the Goldfarb and Idnani dual quadratic programming (QP) algorithm for convex QP problems subject to general linear equality/inequality constraints (Goldfarb and Idnani 1983). That is, problems of the form:

$$\min_{x \in \mathbf{R}^n} g^T x + \frac{1}{2} x^T H x$$

subject to:

$$A_1 x = b_1$$

$$A_2 x \geq b_2$$

given the vectors b_0 , b_1 , and g , and the matrices H , A_0 , and A_1 . Matrix H is required to be positive definite. In this case, a unique x solves the problem, or the constraints are

inconsistent. If H is not positive definite, a positive definite perturbation of H is used in place of H . For more details, see Powell (1983, 1985).

If a perturbation of H , $H + \alpha I$, is used in the QP problem, $H + \alpha I$ also should be used in the definition of the Lagrange multipliers.

Example

In this example, the QP problem:

$$\min f(x) = -x_0^2 + x_1^2 + x_2^2 + x_3^2 + x_4^2 - 2x_1x_2 - 2x_3x_4 - 2x_0$$

subject to:

$$x_0 + x_1 + x_2 + x_3 + x_4 = 5$$

$$x_2 - 2x_3 - 2x_4 = -3$$

is solved.

```
RM, a, 2, 5
; Define the coefficient matrix A.
row 0: 1 1 1 1 1
row 1: 0 0 1 -2 -2
h = [[2, 0, 0, 0, 0], [0, 2, -2, 0, 0], $
      [0, -2, 2, 0, 0], [0, 0, 0, 2, -2], $
      [0, 0, 0, -2, 2]]
; Define the Hessian matrix of the objective function. Notice
; that since h is symmetric, the array concatenation operators
; "[ ]" are used to define it.
b = [5, -3]
; Define b.
g = [-2, 0, 0, 0, 0]
; Define g.
x = IMSL_QUADPROG(a, b, g, h)
; Call IMSL_QUADPROG.
PM, x
; Output solution.
Solution:
      1.00000
      1.00000
      1.00000
      1.00000
      1.00000
```

Errors

Warning Errors

`MATH_NO_MORE_PROGRESS`—Due to the effect of computer rounding error, a change in the variables fails to improve the objective function value. Usually, the solution is close to optimum.

Fatal Errors

`MATH_SYSTEM_INCONSISTENT`—System of equations is inconsistent. There is no solution.

Version History

6.4	Introduced

IMSL_MINCONGEN

The IMSL_MINCONGEN function minimizes a general objective function subject to linear equality/inequality constraints.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_MINCONGEN(f, a, b, xl, xub [, ACTIVE_CONST=variable]
[, /DOUBLE] [, GRAD=string] [, LAGRANGE_MULT=variable]
[, MAX_FCN=value] [, MEQ=value] [, NUM_ACTIVE=variable]
[, OBJ=variable] [, TOLERANCE=value] [, XGUESS=array])
```

Return Value

One-dimensional array of length *nvar* containing the computed solution.

Arguments

a

Two-dimensional array of size *ncon* by *nvar* containing the equality constraint gradients in the first MEQ rows followed by the inequality constraint gradients, where *ncon* is the number of linear constraints (excluding simple bounds) and *nvar* is the number of variables. See the keyword MEQ for setting the number of equality constraints.

b

One-dimensional array of size *ncon* containing the right-hand sides of the linear constraints. Specifically, the constraints on the variables x_i , $i = 0, nvar - 1$, are $a_{k,0}x_0 + \dots + a_{k,nvar-1}x_{nvar-1} = b_k$, $k = 0, \dots, Meq - 1$ and $a_{k,0}x_0 + \dots + a_{k,nvar-1}x_{nvar-1} \leq b_k$, $k = Meq, \dots, ncon - 1$. Note that the data that define the equality constraints come before the data of the inequalities.

f

Scalar string specifying a user-supplied function to evaluate the function to be minimized. Function f accepts a one-dimensional array of length $n = N\_ELEMENTS(x)$ containing the point at which the function is evaluated. The return value of this function is the function value at x .

xlbl

One-dimensional array of length $nvar$ containing the lower bounds on the variables; choose a very large negative value if a component should be unbounded below or set $xlbl(i) = xub(i)$ to freeze the i -th variable. Specifically, these simple bounds are $xlbl(i) \leq x_i$, for $i = 0, \dots, nvar-1$.

xub

One-dimensional array of length $nvar$ containing the upper bounds on the variables; choose a very large positive value if a component should be unbounded above. Specifically, these simple bounds are $x_i \leq xub(i)$, for $i = 0, nvar - 1$.

Keywords

ACTIVE_CONST

Named variable into which an one-dimensional array of length `NUM_ACTIVE` containing the indices of the final active constraints is stored.

DOUBLE

If present and nonzero, double precision is used.

GRAD

Scalar string specifying the name of the user-supplied function to compute the gradient at the point x . The `GRAD` function accepts a one-dimensional array of length $nvar$. The return value of this function is a one-dimensional array of length $nvar$ containing the values of the gradient of the objective function.

LAGRANGE_MULT

Named variable into which an one-dimensional array of length `NUM_ACTIVE` containing the Lagrange multiplier estimates of the final active constraints is stored.

MAX_FCN

Maximum number of function evaluations. Default: MAX_FCN = 400

MEQ

Number of linear equality constraints. Default: MEQ = 0

NUM_ACTIVE

Named variable into which the final number of active constraints is stored.

OBJ

Named variable into which the value of the objective function is stored.

TOLERANCE

The nonnegative tolerance on the first order conditions at the calculated solution.
Default: TOLERANCE = SQRT(ϵ), where ϵ is machine epsilon.

XGUESS

One-dimensional array with nvar components containing an initial guess. Default: XGUESS = 0

Discussion

The IMSL_MINCONGEN function is based on M.J.D. Powell's TOLMIN, which solves linearly constrained optimization problems, i.e., problems of the form:

$$\min f(x)$$

subject to:

$$A_1 x = b_1$$

$$A_2 x \leq b_2$$

$$x_l \leq x \leq x_u$$

given the vectors b_1 , b_2 , x_l , and x_u and the matrices A_1 and A_2 .

The algorithm starts by checking the equality constraints for inconsistency and redundancy. If the equality constraints are consistent, the method will revise x^0 , the initial guess, to satisfy:

$$A_1 x = b_1$$

Next, x^0 is adjusted to satisfy the simple bounds and inequality constraints. This is done by solving a sequence of quadratic programming subproblems to minimize the sum of the constraint or bound violations.

Now, for each iteration with a feasible x^k , let J_k be the set of indices of inequality constraints that have small residuals. Here, the simple bounds are treated as inequality constraints. Let I_k be the set of indices of active constraints. The following quadratic programming problem:

$$\min f(x^k) + d^T \nabla f(x^k) + \frac{1}{2} d^T B^k d$$

subject to:

$$a_j d = 0, \quad j \in I_k$$

$$a_j d \leq 0, \quad j \in J_k$$

is solved to get (d^k, λ^k) where a_j is a row vector representing either a constraint in A_1 or A_2 or a bound constraint on x . In the latter case, the $a_j = e_i$ for the bound constraint $x_i \leq (x_u)_i$ and $a_j = -e_i$ for the constraint $-x_i \leq (x_l)_i$. Here, e_i is a vector with 1 as the i -th component, and zeros elsewhere. Variables λ^k are the Lagrange multipliers, and B^k is a positive definite approximation to the second derivative $\nabla^2 f(x^k)$.

After the search direction d^k is obtained, a line search is performed to locate a better point. The new point $x^{k+1} = x^k + \alpha^k d^k$ has to satisfy the conditions:

$$f(x^k + \alpha^k d^k) \leq f(x^k) + 0.1 \alpha^k (d^k)^T \nabla f(x^k)$$

and:

$$(d^k)^T \nabla f(x^k + \alpha^k d^k) \leq 0.7 (d^k)^T \nabla f(x^k)$$

The main idea in forming the set J_k is that, if any of the equality constraints restricts the step-length α^k , then its index is not in J_k . Therefore, small steps are likely to be avoided.

Finally, the second derivative approximation B^k , is updated by the BFGS formula, if the condition:

$$(d^k)^T \nabla f(x^k + \alpha^k d^k) - \nabla f(x^k) > 0$$

holds. Let $x^k \leftarrow x^{k+1}$, and start another iteration.

The iteration repeats until the stopping criterion:

$$\| \nabla f(x^k) - A^k \lambda^K \|_2 \leq \tau$$

is satisfied. Here τ is the supplied tolerance. For more details, see Powell (1988, 1989).

Since a finite difference method is used to approximate the gradient for some single precision calculations, an inaccurate estimate of the gradient may cause the algorithm to terminate at a noncritical point. In such cases, high precision arithmetic is recommended. Also, if the gradient can be easily provided, the input keyword GRAD should be used.

Examples

Example 1

In this example, the problem:

$$\begin{aligned} \min f(x) &= x_1^2 + x_2^2 + x_3^2 + x_3^2 + x_3^2 - 2x_2x_3 - 2x_4x_5 - 2x_1 \\ &\text{subject to } x_1 + x_2 + x_3 + x_4 + x_5 = 5 \\ &\quad x_3 - 2x_4 - 2x_5 = -3 \\ &\quad 0 \leq x \leq 10 \end{aligned}$$

```
.RUN
FUNCTION fcn, x
    f = x(0)*x(0) + x(1)*x(1) + x(2)*x(2) + x(3)*x(3) + $
        x(4)*x(4) - 2.0*x(1)*x(2) - 2.0*x(3) * x(4) - $
        2.0*x(0)
    RETURN, f
END

meq = 2
a = TRANSPOSE([[1.0, 1.0, 1.0, 1.0, 1.0], $
    [0.0, 0.0, 1.0, -2.0, -2.0]])
b = [5.0, -3.0]
xlb = FLTARR(5)
xlb(*) = 0.0
xub = FLTARR(5)
xub(*) = 10.0
; Set !QUIET to suppress note errors
!QUIET = 1
x = IMSL_MINCONGEN('fcn', a, b, xlb, xub, Meq = meq)
PM, x, Title = 'Solution'

Solution
    1.00000
    1.00000
    1.00000
    1.00000
```

1.00000

Example 2

In this example, the problem from Schittkowski (1987):

$$\begin{aligned} \min f(x) &= -x_0x_1x_2 \\ \text{subject to } &-x_0 - 2x_1 - 2x_2 \leq 0 \\ &x_0 + 2x_1 + 2x_2 \leq 72 \\ &0 \leq x_0 \leq 20 \\ &0 \leq x_1 \leq 11 \\ &0 \leq x_2 \leq 42 \end{aligned}$$

is solved with an initial guess of $x_0 = 10$, $x_1 = 10$ and $x_2 = 10$.

```
.RUN
FUNCTION fcn, x
    f = -x(0)*x(1)*x(2)
    RETURN, f
END

.RUN
FUNCTION gradient, x
    g = FLTARR(3)
    g(0) = -x(1)*x(2)
    g(1) = -x(0)*x(2)
    g(2) = -x(0)*x(1)
    RETURN, g
END

meq = 0
a = TRANSPOSE([[-1.0, -2.0, -2.0], [1.0, 2.0, 2.0]])
b = [0.0, 72.0]
xlb = FLTARR(3)
xlb(*) = 0.0
xub = [20.0, 11.0, 15.0]
xguess = FLTARR(3)
xguess(*) = 10.0
; Set !QUIET to suppress note errors
!QUIET = 1
x = IMSL_MINCONGEN('fcn', a, b, xlb, xub, Meq = meq, $
    Grad = 'gradient', Xguess = xguess, Obj = obj)
PM, x, Title = 'Solution'

Solution
20.0000
```

```
11.0000
15.0000
```

```
PRINT, 'Objective value =', obj
```

```
Objective value = -3300.00
```

Version History

6.4	Introduced

IMSL_CONSTRAINED_NLP

The IMSL_CONSTRAINED_NLP function solves a general nonlinear programming problem using a sequential equality constrained quadratic programming method.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_CONSTRAINED_NLP (f, m, n [, /DOUBLE] [, DEL0=value]
    [, DELMIN=string] [, DIFFTYPE=value] [, EPSDIF=value] [, EPSFCN=value]
    [, GRAD=value] [, IBTYPE=string] [, ITMAX=value] [, MEQ=value]
    [, OBJ=value] [, SCFMAX=string] [, SMALLW=string] [, TAU0=value]
    [, TAUBND=value] [, XGUESS=array] [, XLB=variable] [, XSCALE=vector]
    [, XUB=variable])
```

Return Value

The solution of the nonlinear programming problem.

Arguments

f

Scalar string specifying a user-supplied procedure to evaluate the objective function and constraints at a given point. The input parameters are:

- *x*—One dimensional array at which the objective function or a constraint is evaluated.
- *iact*—Integer indicating whether evaluation of the objective function is requested or evaluation of a constraint is requested. If *iact* is zero, then an objective function evaluation is requested. If *iact* is nonzero then the value if *iact* indicates the index of the constraint to evaluate.
- *result*—If *iact* is zero, then *Result* is the computed objective function at the point *x*. If *iact* is nonzero, then *Result* is the requested constraint value at the point *x*.

- *ierr*—Integer variable. On input *ierr* is set to 0. If an error or other undesirable condition occurs during evaluation, then *ierr* should be set to 1. Setting *ierr* to 1 will result in the step size being reduced and the step being tried again. (If *ierr* is set to 1 for XGUESS, then an error is issued.)

m

Total number of constraints.

n

Number of variables.

Keywords

DOUBLE

If present and nonzero, double precision is used.

DEL0

In the initial phase of minimization, a constraint is considered binding if:

$$\frac{g_i(x)}{\max(1, \|\nabla g_i(x)\|)} \leq \text{Del0} \quad i = M_e + 1, \dots, M$$

Good values are between .01 and 1.0. If DEL0 is too small, then identification of the correct set of binding constraints may be delayed. Conversely, if DEL0 is too large, then the method will often escape to the full regularized SQP method. This method uses individual slack variables for any active constraint, which is quite costly. For well-scaled problems $DEL0 = 1.0$ is reasonable.

Default: $DEL0 = .5 * \text{Tau0}$.

DELMIN

Scalar which defines allowable constraint violations of the final accepted result.

Constraints are satisfied if $|g_i(x)|$ is less than or equal to DELMIN, and $g_i(x)$ is greater than or equal to $(-\text{Delmin})$ respectively. Default: $DELMIN = \min(\text{Del0}/10, \max(\text{epsdif}, \min(\text{del0}/10, \max(1.E-6 * \text{del0}, \text{smallw})))$

DIFFTYPE

Type of numerical differentiation to be used. Default: DIFFTYPE = 1

- 1—Use a forward difference quotient with discretization stepsize $0.1(\text{epsfcn}^{1/2})$ component-wise relative.
- 2—Use the symmetric difference quotient with discretization stepsize $0.1(\text{epsfcn}^{1/3})$ component-wise relative.
- 3—Use the sixth order approximation computing a Richardson extrapolation of three symmetric difference quotient values. This uses a discretization stepsize $0.01(\text{epsfcn}^{1/7})$.

This keyword is not valid if the keyword GRAD is supplied.

EPSDIF

Relative precision in gradients. Default: $\text{EPSDIF} = \text{eps}$ where eps is the machine precision. This keyword is not valid if the keyword GRAD is supplied.

EPSFCN

Relative precision of the function evaluation routine. Default: $\text{EPSFCN} = \text{eps}$ where eps is the machine precision. This keyword is not valid if the keyword GRAD is supplied.

GRAD

Scalar string specifying a user-supplied procedure to evaluate the gradients at a given point. The procedure specified by GRAD has the following parameters:

- x —One dimensional array at which the gradient of the objective function or gradient of a constraint is evaluated.
- $iact$ —Integer indicating whether evaluation of the gradient of the objective function is requested or evaluation of gradient of a constraint is requested. If $iact$ is zero, then an objective function evaluation is requested. If $iact$ is nonzero then the value if $iact$ indicates the index of the constraint to evaluate.
- $result$ —If $iact$ is zero, then $Result$ is the computed gradient of the objective function at the point x . If $iact$ is nonzero, then $Result$ is the gradient of the requested constraint value at the point x .

IBTYPE

Scalar indicating the types of bounds on variables.

- 0—User supplies all the bounds.
- 1—All variables are non-negative.

- 2—All variables are nonpositive.
- 3—User supplies only the bounds on first variable; all other variables have the same bounds.
- Default: no bounds are enforced

ITMAX

Maximum number of iterations allowed. Default: ITMAX = 200

MEQ

Number of equality constraints. Default: MEQ = m

OBJ

Name of a variable into which a scalar containing the value of the objective function at the computed solution is stored.

SCFMAX

Scalar containing the bound for the internal automatic scaling of the objective function. Default: SCFMAX = 1.0e4

SMALLW

Scalar containing the error allowed in the multipliers. For example, a negative multiplier of an inequality constraint is accepted (as zero) if its absolute value is less than SMALLW. Default: SMALLW = $\exp(2 \cdot \log(\text{eps}/3))$ where eps is the machine precision.

TAU0

A universal bound describing how much the unscaled penalty-term may deviate from zero.

IMSL_CONSTRAINED_NLP assumes that within the region described by:

$$\sum_{i=1}^{M_e} |g_i(x)| - \sum_{i=M_e+1}^M \min(0, g_i(x)) \leq \text{Tau0}$$

all functions may be evaluated safely. The initial guess, however, may violate these requirements. In that case, an initial feasibility improvement phase is run by IMSL_CONSTRAINED_NLP until such a point is found. A small TAU0 diminishes the efficiency of IMSL_CONSTRAINED_NLP, because the iterates then will follow

the boundary of the feasible set closely. Conversely, a large TAU0 may degrade the reliability of the code. Default TAU0 = 1.0

TAUBND

Amount by which bounds may be violated during numerical differentiation. Bounds are violated by *TAUBND* (at most) only if a variable is on a bound and finite differences are taken for gradient evaluations. This keyword is not valid if the keyword GRAD is supplied. Default: TAUBND = 1.0.

XGUESS

Array with n components containing an initial guess of the computed solution. Default: XGUESS = X , with the smallest value of $\|X\|_2$ that satisfies the bounds.

XLB

Named variable, containing a one-dimensional array with n components, containing the lower bounds on the variables. (Input, if IBTYPE = 0; Output, if IBTYPE = 1 or 2; Input/Output, if IBTYPE = 3). If there is no lower bound on a variable, the corresponding XLB value should be set to negative machine infinity. Default: no lower bounds are enforced on the variables

XSCALE

Vector of length n setting the internal scaling of the variables. The initial value given and the objective function and gradient evaluations however are always in the original unscaled variables. The first internal variable is obtained by dividing values $x(I)$ by $XSCALE(I)$. This keyword is not valid if the keyword GRAD is supplied.

In the absence of other information, set all entries to 1.0. Default: XSCALE(*) = 1.0.

XUB

Named variable, containing a one-dimensional array with n components, containing the upper bounds on the variables. (Input, if IBTYPE = 0; Output, if IBTYPE = 1 or 2; Input/Output, if IBTYPE = 3). If there is no upper bound on a variable, the corresponding XUB value should be set to positive machine infinity. Default: no upper bounds are enforced on variables.

Description

The routine IMSL_CONSTRAINED_NLP provides an interface to a licensed version of subroutine DONLP2, a code developed by Peter Spellucci (1998). It uses a

sequential equality constrained quadratic programming method with an active set technique, and an alternative usage of a fully regularized mixed constrained subproblem in case of nonregular constraints (for example, linear dependent gradients in the “working sets”). It uses a slightly modified version of the Pantoja-Mayne update for the Hessian of the Lagrangian, variable dual scaling and an improved Armijjo-type stepsize algorithm. Bounds on the variables are treated in a gradient-projection like fashion. Details may be found in the following two papers:

- P. Spellucci: An SQP method for general nonlinear programs using only equality constrained subproblems. *Math. Prog.* 82, (1998), 413-448.
- P. Spellucci: A new technique for inconsistent problems in the SQP method. *Math. Meth. of Oper. Res.* 47, (1998), 355-500. (published by Physica Verlag, Heidelberg, Germany).

The problem is stated as follows:

$$\min_{x \in \mathbb{R}^n} f(x)$$

subject to:

$$\begin{aligned} g_j(x) &= 0, \text{ for } j = 1, \dots, m_e \\ g_j(x) &\geq 0, \text{ for } j = m_e + 1, \dots, m \\ x_l &\leq x \leq x_u \end{aligned}$$

Although default values are provided for input keywords, it may be necessary to adjust these values for some problems. Through the use of keywords, IMSL_CONSTRAINED_NLP allows for several parameters of the algorithm to be adjusted to account for specific characteristics of problems. The *DONLP2 Users Guide* provides detailed descriptions of these parameters as well as strategies for maximizing the performance of the algorithm. The *DONLP2 Users Guide* is available in the “manuals” subdirectory of the main product installation directory. In addition, the following are guidelines to consider when using IMSL_CONSTRAINED_NLP.

- A good initial starting point is very problem-specific and should be provided by the calling program whenever possible. For more details, see the keyword XGUESS.
- Gradient approximation methods can have an effect on the success of IMSL_CONSTRAINED_NLP. Selecting a higher order approximation method may be necessary for some problems. For more details, see the keyword DIFFTYPE.
- If a two-sided constraint:

$$l_i \leq g_i(x) \leq u_i$$

is transformed into two constraints:

$$g_{2i}(x) \geq 0 \quad \text{and} \quad g_{2i+1}(x) \geq 0$$

then choose:

$$\text{DEL0} < \frac{1}{2}(u_i - l_i) / \max\{1, \|\nabla g_i(x)\|\}$$

or at least try to provide an estimate for that value. This will increase the efficiency of the algorithm. For more details, see the keyword DEL0.

- The parameter *ierr* provided in the interface to the user supplied function *f* can be very useful in cases when evaluation is requested at a point that is not possible or reasonable. For example, if evaluation at the requested point would result in a floating point exception, then setting *ierr* to 1 and returning without performing the evaluation will avoid the exception. IMSL_CONSTRAINED_NLP will then reduce the stepsize and try the step again. Note, if *ierr* is set to 1 for the initial guess, then an error is issued.

Example

The problem:

$$\min F(x) = (x_1 - 2)^2 + (x_2 - 1)^2$$

subject to:

$$g_1(x) = x_1 - 2x_2 + 1 = 0$$

$$g_2(x) = -x_1^2/4 - x_2^2 + 1 \geq 0$$

is solved first with finite difference gradients, then with analytic gradients.

```

PRO Nlp_grad, x, iact, result
  CASE iact OF
    0:result = [ 2 * (x(0) - 2.), 2 * (x(1)-1.)]
    1:result = [1., -2. ]
    2:result = [-0.5*x(0), -2.0*x(1)]
  ENDCASE
  RETURN
END

```

```

PRO Nlp_fcn, x, iact, result, ierr
  tmp1 = x(0)-2.
  tmp2 = x(1) - 1.
  CASE iact OF
    0:result = tmp1^2 + tmp2^2
    1:result = x(0) -2.*x(1) + 1.
  ENDCASE
  RETURN
END

```

```

        2:result = -(x(0)^2)/4. - x(1)^2 + 1.
    ENDCASE
    ierr = 0
END

; Ex #1, Finite difference gradients
ans1 = IMSL_CONSTRAINED_NLP('nlp_fcn', 2, 2, MEQ = 1)
PM, ans1, title='X with finite difference gradient'

; Ex #2, Analytic gradients
ans2 = IMSL_CONSTRAINED_NLP('nlp_fcn', 2, 2, MEQ = 1, $
    GRAD = 'nlp_grad')
PM, ans2, title='X with Analytic gradient'
```

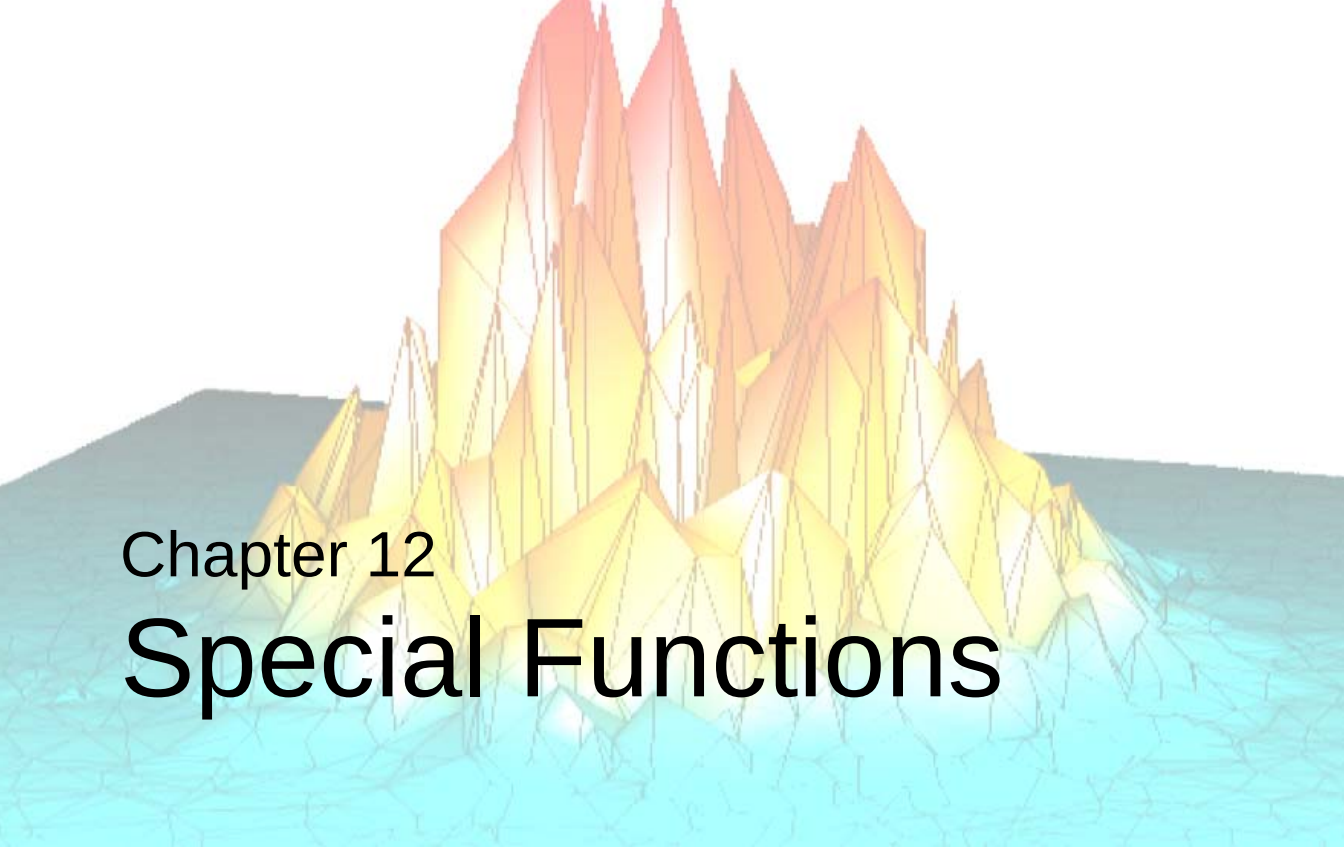
Output

```

X with finite difference gradient
    0.822877
    0.911439
X with Analytic gradient
    0.822877
    0.911438
```

Version History

6.4	Introduced



Chapter 12

Special Functions

This section contains the following topics:

Overview: Special Functions	474	Special Functions Routines	475

Overview: Special Functions

This chapter describes special functions included in IDL Advanced Math and Stats. See “[Special Functions Routines](#)” on page 475 for a list of the included routines.

Special Functions Routines

Error Functions

[IMSL_ERF](#)—Error function.

[IMSL_ERFC](#)—Complementary error function.

[IMSL_BETA](#)—Beta function.

[IMSL_LNBETA](#)—Logarithmic beta function.

[IMSL_BETAI](#)—Incomplete beta function.

Gamma Functions

[IMSL_LNGAMMA](#)—Logarithmic gamma function.

[IMSL_GAMMA_ADV](#)—Real gamma function.

[IMSL_GAMMAI](#)—Incomplete gamma function.

Bessel Functions with Real Order and Complex Argument

[IMSL_BESSI](#)—Modified Bessel function of the first kind.

[IMSL_BESSJ](#)—Bessel function of the first kind.

[IMSL_BESSK](#)—Modified Bessel function of the second kind.

[IMSL_BESSY](#)—Bessel function of the second kind.

[IMSL_BESSI_EXP](#)—Bessel function $e^{-|x|}I_0(x)$, Bessel function $e^{-|x|}I_1(x)$.

[IMSL_BESSK_EXP](#)—Bessel function $e^xK_0(x)$, Bessel function $e^xK_1(x)$.

Elliptic Integrals

[IMSL_ELK](#)—Complete elliptic integral of the first kind.

[IMSL_ELE](#)—Complete elliptic integral of the second kind.

[IMSL_ELRF](#)—Carlson's elliptic integral of the first kind.

[IMSL_ELRD](#)—Carlson's elliptic integral of the second kind.

[IMSL_ELRJ](#)—Carlson's elliptic integral of the third kind.

[IMSL_ELRC](#)—Special case of Carlson's elliptic integral.

Fresnel Integrals

[IMSL_FRESNEL_COSINE](#)—Cosine Fresnel integral.

[IMSL_FRESNEL_SINE](#)—Sine Fresnel integral.

Airy Functions

[IMSL_AIRY_AI](#)—Airy function, and derivative of the Airy function.

[IMSL_AIRY_BI](#)—Airy function of the second kind, and derivative of the Airy function of the second kind.

Kelvin Functions

[IMSL_KELVIN_BER0](#)—Kelvin function *ber* of the first kind, order 0, and derivative of the Kelvin function *ber*.

[IMSL_KELVIN_BEI0](#)—Kelvin function *bei* of the first kind, order 0, and derivative of the Kelvin function *bei*.

[IMSL_KELVIN_KER0](#)—Kelvin function *ker* of the second kind, order 0, and derivative of the Kelvin function *ker*.

[IMSL_KELVIN_KEI0](#)—Kelvin function *kei* of the second kind, order 0 and derivative of the Kelvin function *kei*.

IMSL_ERF

The IMSL_ERF function evaluates the real error function $\text{erf}(x)$. Using a keyword, the inverse error function $\text{erf}^{-1}(x)$ can be evaluated.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_ERF(*x* [, /DOUBLE] [, /INVERSE])

Return Value

The value of the error function $\text{erf}(x)$.

Arguments

x

Expression for which the error function is to be evaluated.

Keywords

DOUBLE

If present and nonzero, double precision is used.

INVERSE

Evaluates the real inverse error function $\text{erf}^{-1}(x)$. The inverse error function is defined only for $-1 < x < 1$.

Discussion

The error function $\text{erf}(x)$ is defined below:

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

All values of x are legal. The inverse error function $y = \text{erf}^{-1}(x)$ is such that $x = \text{erf}(y)$.

Examples

Example 1

Plot the error function over $[-3, 3]$. The results are shown in [Figure 12-1](#).

```
x = 6 * FINDGEN(100)/99 - 3  
PLOT, x, IMSL_ERF(x), XTitle = 'x', YTitle = 'erf(x)'
```

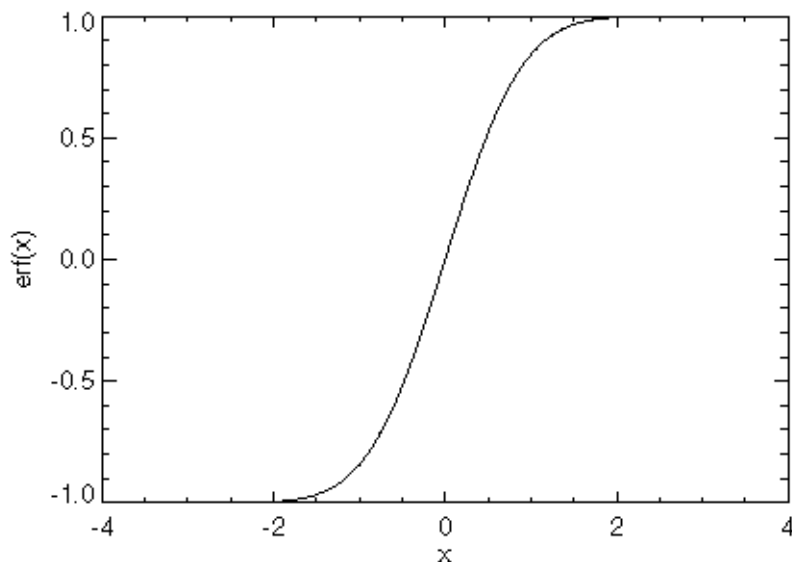


Figure 12-1: Plot of $\text{erf}(x)$

Example 2

Plot the inverse of the error function over $(-1, 1)$. The results are shown in [Figure 12-2](#).

```
x = 2 * FINDGEN(100)/99 - 1
PLOT, x, IMSL_ERF(x(1:98), /Inverse), XTitle = 'x', $
      YTitle = 'erf-1(x)'
```

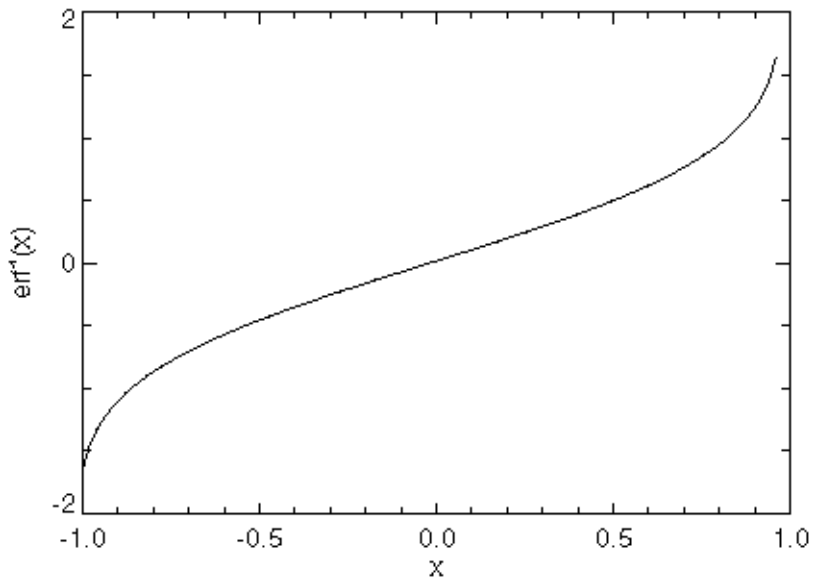


Figure 12-2: Plot of $\text{erf}^{-1}(x)$

Version History

6.4	Introduced

IMSL_ERFC

The IMSL_ERFC function evaluates the real complementary error function $\text{erfc}(x)$. Using a keyword, the inverse complementary error function $\text{erfc}^{-1}(x)$ can be evaluated.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_ERFC(*x* [, /DOUBLE] [, /INVERSE])

Return Value

The value of the complementary error function $\text{erfc}(x)$.

Arguments

x

Expression for which the complementary error function is to be evaluated.

Keywords

DOUBLE

If present and nonzero, double precision is used.

INVERSE

Evaluates the inverse complementary error function $\text{erfc}^{-1}(x)$. The parameter must be in the range $0 < x < 2$.

Discussion

The complementary error function $\text{erfc}(x)$ is defined as:

$$\text{erfc}(x) = \frac{2}{\sqrt{\pi}} \int_x^{\infty} e^{-t^2} dt$$

where parameter x must not be so large that the result underflows. Approximately, x should be less than:

$$[-\ln(\sqrt{\pi} s)]^{1/2}$$

where s is the smallest representable floating-point number.

The inverse complementary error function $y = \text{erfc}^{-1}(x)$ is such that $x = \text{erfc}(y)$.

Examples

Example 1

Plot the complementary error function over $[-3, 3]$. The results are shown in [Figure 12-3](#).

```
x = FINDGEN(100)/99
PLOT, 6 * x - 3, IMSL_ERFC(6 * x - 3), XTitle = 'x', $
      YTitle = 'erfc(x)'
```

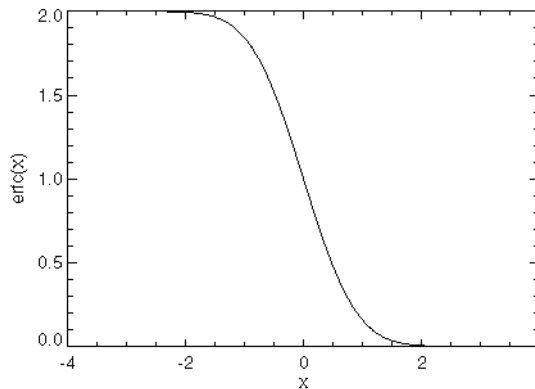


Figure 12-3: Plot of $\text{erf}(x)$

Example 2

Plot the inverse of the complementary error function over (0, 2). The results are shown in [Figure 12-4](#).

```
x = FINDGEN(100)/99
PLOT, 2 * x(1:98), IMSL_ERFC(2 * x(1:98), /Inverse), $
      XTitle = 'x', YTitle = 'erfc!E-1!N(x)'
```

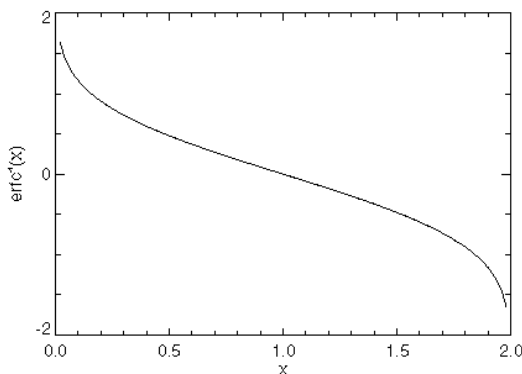


Figure 12-4: Plot of $\text{erfc}^{-1}(x)$

Errors

Alert Errors

MATH_LARGE_ARG_UNDERFLOW—Parameter x must not be so large that the result underflows. Very approximately, x should be less than:

$$2 - \sqrt{\epsilon / (4\pi)}$$

where ϵ is the machine precision.

Warning Errors

MATH_LARGE_ARG_WARN—Parameter $|x|$ should be less than

$$1 / (\sqrt{\epsilon})$$

where ϵ is the machine precision, to prevent the answer from being less accurate than half precision.

Fatal Errors

MATH_ERF_ALGORITHM—Algorithm failed to converge.

MATH_SMALL_ARG_OVERFLOW—Computation of:

$$e^{x^2} \operatorname{erfc}(x)$$

must not overflow.

MATH_REAL_OUT_OF_RANGE—Function is defined only for $0 < x < 2$.

Version History

6.4	Introduced

IMSL_BETA

The IMSL_BETA function evaluates the real beta function $\beta(x, y)$.]

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_BETA(*x*, *y* [, /DOUBLE])

Return Value

The value of the beta function $\beta(x, y)$. If no result can be computed, then NaN (Not a Number) is returned.

Arguments

x

First beta parameter. It must be positive.

y

Second beta parameter. It must be positive.

Keywords

DOUBLE

If present and nonzero, double precision is used.

Discussion

The beta function, $\beta(x, y)$, is defined as:

$$\beta(x, y) = \frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)} = \int_0^1 t^{x-1} (1-t)^{y-1} dt$$

requiring that $x > 0$ and $y > 0$. It underflows for large parameters.

Example

Plot the beta function over $[\epsilon, 1/4 + \epsilon] \times [\epsilon, 1/4 + \epsilon]$ for $\epsilon = 0.01$. The results are shown in [Figure 12-5](#).

```
x = 1e-2 + .25 * FINDGEN(25)/24
y = x
b = FLTARR(25, 25)
FOR i = 0, 24 DO b(i, *) = IMSL_BETA(x(i), y)
; Compute values of the beta function.
SURFACE, b, x, y, XTitle = 'X', YTitle = 'Y', Az = 320, ZAxis = 2
; Plot the computed values as a surface and rotate the plot.
```

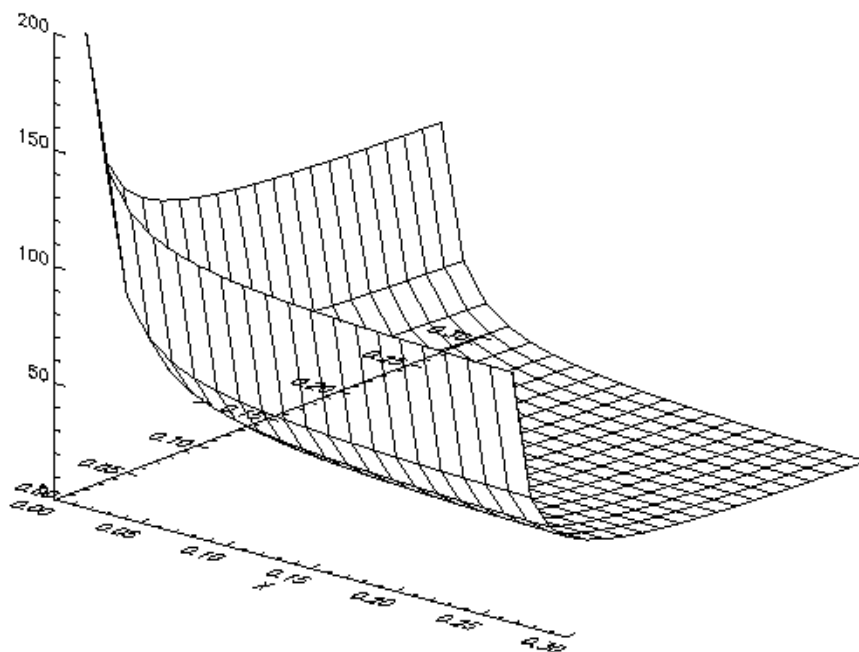


Figure 12-5: Real Beta Function Plot

Errors

Alert Errors

MATH_BETA_UNDERFLOW—Parameters must not be so large that the result underflows.

Fatal Errors

MATH_ZERO_ARG_OVERFLOW—One of the parameters is so close to zero that the result overflows.

Version History

6.4	Introduced

IMSL_LNBETA

The IMSL_LNBETA function evaluates the logarithm of the real beta function $\ln \beta(x, y)$.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_LNBETA(*x*, *y* [, /DOUBLE])

Return Value

The value of the logarithm of the beta function $\beta(x, y)$.

Arguments

x

First argument of the beta function. It must be positive.

y

Second argument of the beta function. It must be positive.

Keywords

DOUBLE

If present and nonzero, double precision is used.

Discussion

The beta function, $\beta(x, y)$, is defined as:

$$\beta(x, y) = \frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)} = \int_0^1 t^{x-1} (1-t)^{y-1} dt$$

and IMSL_LNBETA returns $\ln \beta(x, y)$. The logarithm of the beta function requires that $x > 0$ and $y > 0$. It can overflow for very large parameters.

Example

Evaluate the log of the beta function $\ln \beta(0.5, 0.2)$.

```
PM, IMSL_LNBETA(.5, .2)
      1.83556
```

Errors

Warning Errors

MATH_X_IS_TOO_CLOSE_TO_NEG_1—Result is accurate to less than one precision because the expression $-x / (x + y)$ is too close to -1 .

Version History

6.4	Introduced

IMSL_BETAI

The IMSL_BETAI function evaluates the real incomplete beta function.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_BETAI(*x*, *a*, *b* [, /DOUBLE])

Return Value

The value of the incomplete beta function.

Arguments

x

Upper limit of integration.

a

First beta distribution parameter.

b

Second beta distribution parameter.

Keywords

DOUBLE

If present and nonzero, double precision is used.

Discussion

The incomplete beta function is defined as:

$$I_x(a, b) = \frac{\beta_x(a, b)}{\beta(a, b)} = \frac{1}{\beta(a, b)} \int_0^x t^{a-1} (1-t)^{b-1} dt$$

requiring that $0 \leq x \leq 1$, $a > 0$, and $b > 0$. It underflows for sufficiently small x and large a . This underflow is not reported as an error. Instead, the value zero is returned.

Example

In this example, $I_{0.61}(2.2, 3.7)$ is computed and printed.

```
PM, IMSL_BETAI(.61, 2.2, 3.7)
0.882172
```

Version History

6.4	Introduced

IMSL_LNGAMMA

The IMSL_LNGAMMA function evaluates the logarithm of the absolute value of the gamma function $\log|\Gamma(x)|$.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_LNGAMMA(*x* [, /DOUBLE])

Return Value

The value of the logarithm of gamma function $\log|\Gamma(x)|$.

Arguments

x

Expression for which the logarithm of the absolute value of the gamma function is to be evaluated.

Keywords

DOUBLE

If present and nonzero, double precision is used.

Discussion

The logarithm of the absolute value of the gamma function $\log|\Gamma(x)|$ is computed.

Example

In this example, $\log|\Gamma(3.5)|$ is computed and printed.

```
PM,  IMSL_LNGAMMA(3.5)  
1.20097
```

Errors

Warning Errors

`MATH_NEAR_NEG_INT_WARN`—Result is accurate to less than one-half precision because x is too close to a negative integer.

Fatal Errors

`MATH_NEGATIVE_INTEGER`—Parameter for the function cannot be a negative integer.

`MATH_NEAR_NEG_INT_FATAL`—Parameter for the function is too close to a negative integer.

`MATH_LARGE_ABS_ARG_OVERFLOW`—Parameter $|x|$ must not be so large that the result overflows.

Version History

6.4	Introduced

IMSL_GAMMA_ADV

The IMSL_GAMMA_ADV function evaluates the real gamma function.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_GAMMA_ADV(*x* [, /DOUBLE])

Return Value

The value of the gamma function $\Gamma(x)$.

Arguments

x

Point at which the gamma function is to be evaluated.

Keywords

DOUBLE

If present and nonzero, double precision is used.

Discussion

The gamma function, $\Gamma(x)$, is defined to be:

$$\Gamma(x) = \int_0^{\infty} t^{x-1} e^{-t} dt$$

For $x < 0$, the above definition is extended by analytic continuation.

The gamma function is not defined for integers less than or equal to zero. It underflows for $x \ll 0$ and overflows for large x . It also overflows for values near negative integers.

Example

In this example, $\Gamma(1.5)$ is computed and printed.

```
x = 1.5
ans = IMSL_GAMMA_ADV(x)
PRINT, 'Gamma(', x, ') =', ans
Gamma( 1.50000) = 0.886227
```

Errors

Alert Errors

STAT_SMALL_ARG_UNDERFLOW—The parameter x must be large enough that $\Gamma(x)$ does not underflow. The underflow limit occurs first for parameters close to large negative half integers. Even though other parameters away from these half integers may yield machine-representable values of $\Gamma(x)$, such parameters are considered illegal.

Warning Errors

STAT_NEARR_NEG_INT_WARN—The result is accurate to less than one-half precision because x is too close to a negative integer.

Fatal Errors

STAT_ZERO_ARG_OVERFLOW—The parameter for the gamma function is too close to zero.

STAT_NEAR_NEG_INT_FATAL—The parameter for the function is too close to a negative integer.

STAT_LARGE_ARG_OVERFLOW—The function overflows because x is too large.

STAT_CANNOT_FIND_XMIN—The algorithm used to find $x_{\min}$ failed. This error should never occur.

STAT_CANNOT_FIND_XMAX—The algorithm used to find $x_{\max}$ failed. This error should never occur.

Version History

6.4	Introduced

IMSL_GAMMAI

The IMSL_GAMMAI function evaluates the incomplete gamma function $\gamma(a, x)$.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_GAMMAI(*a*, *x* [, /DOUBLE])

Return Value

The value of the incomplete gamma function $\gamma(a, x)$.

Arguments

a

Integrand exponent parameter. It must be positive.

x

Upper limit of integration. It must be nonnegative.

Keywords

DOUBLE

If present and nonzero, double precision is used.

Discussion

The incomplete gamma function, $\gamma(a, x)$, is defined as follows:

$$\gamma(a, x) = \int_0^x t^{a-1} e^{-t} dt$$

The incomplete gamma function is defined only for $a > 0$. Although $\gamma(a, x)$ is well-defined for $x > -\text{infinity}$, this algorithm does not calculate $\gamma(a, x)$ for negative x . For large a and sufficiently large x , $\gamma(a, x)$ may overflow. Gamma function $\gamma(a, x)$ is bounded by $\Gamma(a)$, and users may find this bound a useful guide in determining legal values for a .

Example

Plot the incomplete gamma function over $[0.1, 1.1] \times [0, 4]$. The results are shown in Figure 12-6.

```
x = 4. * FINDGEN(25)/24
a = 1e-1 + FINDGEN(25)/24
b = FLTARR(25, 25)
FOR i = 0, 24 DO b(i, *) = IMSL_GAMMAI(a(i), x)
!P.Charsize = 2.5
SURFACE, b, a, x, XTitle = 'a', YTitle = 'x'
```

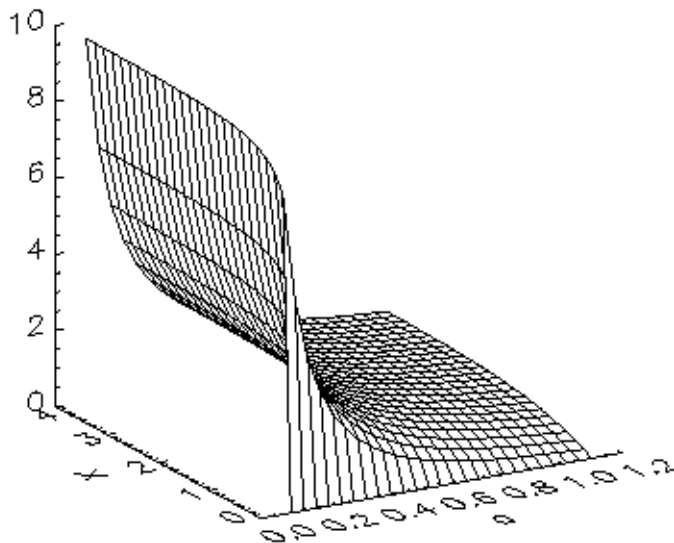


Figure 12-6: Incomplete Gamma Function Plot

Errors

Fatal Errors

MATH_NO_CONV_200_TS_TERMS—Function did not converge in 200 terms of Taylor series.

MATH_NO_CONV_200_CF_TERMS—Function did not converge in 200 terms of the continued fraction.

Version History

6.4	Introduced

IMSL_BESSI

The IMSL_BESSI function evaluates a modified Bessel function of the first kind with real order and real or complex parameters.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_BESSI(*order*, *z* [, /DOUBLE] [, SEQUENCE=*value*])

Return Value

The desired value of the modified Bessel function.

Arguments

order

Real parameter specifying the desired order. The argument *order* must be greater than $-1/2$.

z

Real or complex parameter for which the Bessel function is to be evaluated.

Keywords

DOUBLE

If present and nonzero, double precision is used.

SEQUENCE

If present and nonzero, a one-dimensional array of length *n* containing the values of the Bessel function through the series is returned by IMSL_BESSI, where $n = N\_ELEMENTS(SEQUENCE)$. The *i*-th element of this array is the Bessel function of order (*order* + *i*) at *z* for $i = 0, \dots (n - 1)$.

Discussion

The `IMSL_BESSI` function evaluates a modified Bessel function of the first kind with real order and real or complex parameters. The data type of the returned value is always complex.

The Bessel function, $I_\nu(z)$, is defined as follows:

$$I_\nu(z) = e^{-\nu\pi i/2} J_\nu(ze^{\pi i/2}) \quad \text{for } -\pi < \arg z \leq \frac{\pi}{2}$$

For large parameters, z , Temme's (1975) algorithm is used to find $I_\nu(z)$. The $I_\nu(z)$ values are recurred upward (if stable). This involves evaluating a continued fraction. If this evaluation fails to converge, the answer may not be accurate. For moderate and small parameters, Miller's method is used.

Example

In this example, $J_{0.3+\nu-1}(1.2+0.5i)$, $\nu = 1, \dots, 4$ is computed and printed first by calling `IMSL_BESSI` four times in a row, then by using the keyword `SEQUENCE`.

```
z = COMPLEX(1.2, .5)
FOR i = 0, 3 DO PM, IMSL_BESSI(i + .3, z)
  ( 1.16339, 0.396301)
  ( 0.447264, 0.332142)
  ( 0.0821799, 0.127165)
  ( 0.00577678, 0.0286277)
PM, IMSL_BESSI(.3, z, Sequence = 4), Title = 'With SEQUENCE:'
With SEQUENCE:
  ( 1.16339, 0.396301)
  ( 0.447264, 0.332142)
  ( 0.0821799, 0.127165)
  ( 0.00577678, 0.0286277)
```

Version History

6.4	Introduced

IMSL_BESSJ

The IMSL_BESSJ function evaluates a Bessel function of the first kind with real order and real or complex parameters.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_BESSJ(*order*, *z* [, /DOUBLE] [, SEQUENCE=*value*])

Return Value

The desired value of the Bessel function.

Arguments

order

Real parameter specifying the desired order. The argument *order* must be greater than $-1/2$.

z

Real or complex parameter for which the Bessel function is to be evaluated.

Keywords

DOUBLE

If present and nonzero, double precision is used.

SEQUENCE

If present and nonzero, a one-dimensional array of length *n* containing the values of the Bessel function through the series is returned by IMSL_BESSJ, where $n = \text{NELEMENTS}(\text{SEQUENCE})$. The *i*-th element of this array is the Bessel function of order (*order* + *i*) at *z* for $i = 0, \dots (n - 1)$.

Discussion

The IMSL_BESSJ function evaluates a Bessel function of the first kind with real order and real or complex parameters. The data type of the returned value is always complex.

The Bessel function, $J_\nu(z)$, is defined as follows:

$$J_\nu(z) = \frac{1}{\pi} \int_0^\pi \cos(z \sin \theta - \nu \theta) d\theta - \frac{\sin(\gamma\pi)}{\pi} \int_0^\infty e^{z \sinh t - \nu t} dt$$

for:

$$|\arg z| < \frac{\pi}{2}$$

This function is based on the code BESSCC of Barnett (1981) and Thompson and Barnett (1987). This code computes $J_\nu(z)$ from the modified Bessel function $I_\nu(z)$, using the following relation with:

$$\rho = e^{i\pi/2}$$

$$J_\nu(z) = \begin{cases} \rho I_\nu(z/\rho) & \text{for } -\pi/2 < \arg z \leq \pi \\ \rho^3 I_\nu(\rho^3 z) & \text{for } -\pi < \arg z \leq \pi/2 \end{cases}$$

Example

In this example, $J_{0.3 + \nu-1}(1.2 + 0.5i)$, $\nu = 1, \dots, 4$ is computed and printed.

```
z = COMPLEX(1.2, .5)
FOR i = 0, 3 DO PM, IMSL_BESSJ(i + .3, z)
  ( 0.773756, -0.106925)
  ( 0.400001, 0.158598)
  ( 0.0867063, 0.0920276)
  ( 0.00844932, 0.0239868)
PM, IMSL_BESSJ(.3, z, Sequence = 4), Title = 'With SEQUENCE:'
With SEQUENCE:
  ( 0.773756, -0.106925)
  ( 0.400001, 0.158598)
  ( 0.0867063, 0.0920276)
  ( 0.00844932, 0.0239868)
```

Version History

6.4	Introduced

IMSL_BESSK

The IMSL_BESSK function evaluates a modified Bessel function of the second kind with real order and real or complex parameters.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_BESSK(*order*, *z* [, /DOUBLE] [, SEQUENCE=*value*])

Return Value

The desired value of the modified Bessel function.

Arguments

order

Real parameter specifying the desired order. The argument *order* must be greater than $-1/2$.

z

Real or complex parameter for which the Bessel function is to be evaluated.

Keywords

DOUBLE

If present and nonzero, double precision is used.

SEQUENCE

If present and nonzero, a one-dimensional array of length *n* containing the values of the Bessel function through the series is returned by IMSL_BESSK, where $n = \text{NELEMENTS}(\text{SEQUENCE})$. The *i*-th element of this array is the Bessel function of order (*order* + *i*) at *z* for $i = 0, \dots (n - 1)$.

Discussion

The `IMSL_BESSK` function evaluates a modified Bessel function of the second kind with real order and real or complex parameters. The data type of the returned value is always complex.

The Bessel function, $K_\nu(z)$, is defined as follows:

$$K_\nu(z) = \frac{\pi}{2} e^{v\pi i/2} [iJ_\nu(iz) - Y_\nu(iz)] \quad \text{for } -\pi < \arg z \leq \frac{\pi}{2}$$

This function is based on the code `BESSCC` of Thompson (1981) and Thompson and Barnett (1987). For moderate or large parameters, z , Temme's (1975) algorithm is used to find $K_\nu(z)$. This involves evaluating a continued fraction. If this evaluation fails to converge, the answer may not be accurate. For small z , a Neumann series is used to compute $K_\nu(z)$. Upward recurrence of the $K_\nu(z)$ is always stable.

Example

In this example, $K_{0.3+v-1}(1.2 + 0.5i)$, $v = 1, \dots, 4$ is computed and printed.

```
z = COMPLEX(1.2, .5)
FOR i = 0, 3 DO PM, IMSL_BESSK(i + .3, z)
  ( 0.245546, -0.199599)
  ( 0.335637, -0.362005)
  ( 0.586718, -1.12610)
  ( 0.719457, -4.83864)
PM, IMSL_BESSK(.3, z, Sequence = 4), Title = 'With SEQUENCE:'
With SEQUENCE:
  ( 0.245546, -0.199599)
  ( 0.335637, -0.362005)
  ( 0.586718, -1.12610)
  ( 0.719456, -4.83864)
```

Version History

6.4	Introduced

IMSL_BESSY

The IMSL_BESSY function evaluates a Bessel function of the second kind with real order and real or complex parameters.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_BESSY(*order*, *z* [, /DOUBLE] [, SEQUENCE=*value*])

Return Value

The desired value of the modified Bessel function.

Arguments

order

Real parameter specifying the desired order. The argument *order* must be greater than $-1/2$.

z

Real or complex parameter for which the Bessel function is to be evaluated.

Keywords

DOUBLE

If present and nonzero, double precision is used.

SEQUENCE

If present and nonzero, a one-dimensional array of length *n* containing the values of the Bessel function through the series is returned by IMSL_BESSY, where *n* = NELEMENTS(SEQUENCE). The *i*-th element of this array is the Bessel function of order (*order* + *i*) at *z* for *i* = 0, ... (*n* - 1).

Discussion

The IMSL_BESSY function evaluates a Bessel function of the second kind with real order and real or complex parameters. The data type of the returned value is always complex.

The Bessel function, $Y_\nu(z)$, is defined as follows:

$$Y_\nu(z) = \frac{1}{\pi} \int_0^\pi \sin(z \sin \theta - \nu \theta) d\theta - \frac{\sin(\gamma\pi)}{\pi} \int [e^{\nu t} + e^{-\nu t} \cos(\nu t)] e^{z \sinh t} dt$$

for $|\arg z| < \frac{\pi}{2}$

This function is based on the code BESSCC of Thompson (1981) and Thompson and Barnett (1987). This code computes $Y_\nu(z)$ from the modified Bessel functions $I_\nu(z)$ and $K_\nu(z)$, using the following relation:

$$Y_\nu(z) = e^{(\nu+1)\pi i/2} I_\nu(z) - \frac{2}{\pi} e^{-\nu\pi i/2} K_\nu(z) \quad \text{for } -\pi < \arg z \leq \frac{\pi}{2}$$

Example

In this example, $Y_{0.3+\nu-1}(1.2+0.5i)$, $\nu = 1, \dots, 4$ is computed and printed.

```
z = COMPLEX(1.2, .5)
FOR i = 0, 3 DO PM, IMSL_BESSY(i + .3, z)
  ( -0.0131453, 0.379593)
  ( -0.715533, 0.338082)
  ( -1.04777, 0.794969)
  ( -1.62487, 3.68447)
PM, IMSL_BESSY(.3, z, Sequence = 4), Title = 'With SEQUENCE:'
With SEQUENCE:
  ( -0.0131453, 0.379593)
  ( -0.715533, 0.338082)
  ( -1.04777, 0.794969)
  ( -1.62487, 3.68447)
```

Version History

6.4	Introduced

IMSL_BESSI_EXP

The IMSL_BESSI_EXP function evaluates the exponentially scaled modified Bessel function of the first kind of orders zero and one.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_BESSI_EXP(*order*, *x* [, /DOUBLE])

Return Value

The value of the exponentially scaled modified Bessel function of the first kind of order zero or one evaluated at *x*.

Arguments

order

Order of the function. The order must be either zero or one.

x

Argument for which the function value is desired.

Keywords

DOUBLE

If present and nonzero, double precision is used.

Discussion

If the argument *order* is zero, the Bessel function is $I_0(x)$ is defined to be:

$$I_0(x) = \frac{1}{\pi} \int_0^\pi \cos(x \cos \theta) d\theta$$

If *order* is one, the function $I_1(x)$ is defined to be:

$$I_1(x) = \frac{1}{\pi} \int_0^\pi e^{x \cos \theta} \cos \theta d\theta$$

If *order* is one then IMSL_BESSI_EXP underflows if $|x|/2$ underflows.

Example

The expression $e^{-4.5} I_0(4.5)$ is computed directly by calling IMSL_BESSI_EXP and indirectly by calling IMSL_BESSI. The absolute difference is printed. For large x , the internal scaling provided by IMSL_BESSI_EXP avoids overflow that may occur in IMSL_BESSI.

Output

```
ans = IMSL_BESSI_EXP(0, 4.5)
error = ABS(ans - EXP(-4.5)*IMSL_BESSI(0, 4.5))
PRINT, ans
      0.194198
PRINT, 'Error =', error
Error = 4.4703484e-08
```

Version History

6.4	Introduced

IMSL_BESSK_EXP

The IMSL_BESSK_EXP function evaluates the exponentially scaled modified Bessel function of the third kind of orders zero and one.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_BESSK_EXP(*order*, *x* [, /DOUBLE])

Return Value

The value of the exponentially scaled Bessel function $e^x K_0(x)$ or $e^x K_1(x)$

Arguments

order

Order of the function. The order must be either zero or one.

x

Argument for which the function value is desired.

Keywords

DOUBLE

If present and nonzero, double precision is used.

Discussion

If the argument *order* is zero, the Bessel function $K_0(x)$ is defined to be:

$$K_0(x) = \int_0^{\infty} \cos(x \sin t) dt$$

If *order* is one, the value of the Bessel function $K_1(x)$:

$$K_1(x) = \frac{1}{\pi} \int_0^\pi e^{x \cos \theta} \cos \theta d\theta$$

The argument x must be greater than zero for the result to be defined.

Example

The expression:

$$\sqrt{e} K_0(0.5)$$

is computed directly by calling `IMSL_BESSK_EXP`, and indirectly by calling [IMSL_BESSK](#). The absolute difference is printed. For large x , the internal scaling provided by `IMSL_BESSK_EXP` avoids underflow that may occur in `IMSL_BESSK`.

```
ans = IMSL_BESSK_EXP(0, 0.5)
error = ABS(ans - (EXP(0.5))*IMSL_BESSK(0, 0.5))
PRINT, ans
      1.52411
PRINT, 'Error =', error
Error = 1.1920929e-07
```

Errors

Fatal Errors

MATH_SMALL_ARG_OVERFLOW—The argument x must be large enough ($x > \max(1/b, s)$ where s is the smallest representable positive number and b is the largest representable number) that $K_1(x)$ does not overflow.

Version History

6.4	Introduced

IMSL_ELK

The IMSL_ELK function evaluates the complete elliptic integral of the kind $K(x)$.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_ELK(*x* [, /DOUBLE])

Return Value

The complete elliptic integral $K(x)$.

Arguments

x

Argument for which the function value is desired.

Keywords

DOUBLE

If present and nonzero, double precision is used.

Discussion

The complete elliptic integral of the first kind is defined to be:

$$K(x) = \int_0^{\pi/2} \frac{d\theta}{[1 - x \sin^2 \theta]^{1/2}} \quad \text{for } 0 \leq x < 1$$

The argument x must satisfy $0 \leq x < 1$; otherwise, IMSL_ELK returns the largest representable floating-point number.

The function $K(x)$ is computed using [IMSL_ELRF](#) and the relation $K(x) = R_F(0, 1 - x, 1)$.

Example

The integral $K(0)$ is evaluated.

```
PRINT, IMSL_ELK(0.0)
      1.57080
```

Version History

6.4	Introduced

IMSL_ELE

The IMSL_ELE function evaluates the complete elliptic integral of the second kind $E(x)$.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_ELE(*x* [, /DOUBLE])

Return Value

The complete elliptic integral $E(x)$.

Arguments

x

Argument for which the function value is desired.

Keywords

DOUBLE

If present and nonzero, double precision is used.

Discussion

The complete elliptic integral of the second kind is defined to be:

$$E(x) = \int_0^{\pi/2} [1 - x \sin^2 \theta]^{1/2} d\theta \quad \text{for } 0 \leq x < 1$$

The argument x must satisfy $0 \leq x < 1$; otherwise, IMSL_ELE returns the largest representable floating-point number.

The function $E(x)$ is computed using the routine [IMSL_ELRF](#) and [IMSL_ELRD](#). The computation is done using the relation:

$$E(x) = R_F(0.5 - x) - \frac{x}{3} R_D(0.5 - x)$$

Example

The integral $E(0.33)$ is evaluated.

```
PRINT, IMSL_ELE(0.33)
1.43183
```

Version History

6.4	Introduced

IMSL_ELRF

The IMSL_ELRF function evaluates Carlson's elliptic integral of the first kind $R_F(x, y, z)$.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_ELRF(*x*, *y*, *z* [, /DOUBLE])

Return Value

The complete elliptic integral $R_F(x, y, z)$.

Arguments

x

First argument for which the function value is desired. It must be nonnegative.

y

Second argument for which the function value is desired. It must be nonnegative.

z

Third argument for which the function value is desired. It must be nonnegative.

Keywords

DOUBLE

If present and nonzero, double precision is used.

Discussion

Carlson's elliptic integral of the second kind is defined to be:

$$R_F(x, y, z) = \frac{1}{2} \int_0^\infty \frac{dt}{[(t+x)(t+y)(t+z)]^{1/2}}$$

The arguments must be nonnegative and less than or equal to $b/5$. In addition, $x + y$, $x + z$, and $y + z$ must be greater than or equal to $5s$. Should any of these conditions fail, `IMSL_ELRF` is set to b . Here, b is the largest and is the smallest representable number.

The `IMSL_ELRF` function is based on the code by Carlson and Notis (1981) and the work of Carlson (1979).

Example

The integral $R_F(0, 1, 2)$ is computed.

```
PRINT, IMSL_ELRF(0.0, 1.0, 2.0)
      1.31103
```

Version History

6.4	Introduced

IMSL_ELRD

The IMSL_ELRD function evaluates Carlson's elliptic integral of the second kind $R_D(x, y, z)$.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_ELRD(*x*, *y*, *z* [, /DOUBLE])

Return Value

The complete elliptic integral $R_D(x, y, z)$

Arguments

x

First argument for which the function value is desired. It must be nonnegative.

y

Second argument for which the function value is desired. It must be nonnegative.

z

Third argument for which the function value is desired. It must be positive.

Keywords

DOUBLE

If present and nonzero, double precision is used.

Discussion

Carlson's elliptic integral of the second kind is defined to be:

$$R_D(x, y, z) = \frac{3}{2} \int_0^\infty \frac{dt}{[(t+x)(t+y)(t+z)^3]^{1/2}}$$

Arguments must be nonnegative and less than or equal to $0.69(-\ln \epsilon)^{1/9} s^{-2/3}$ where ϵ is the machine precision, s is the smallest representable positive number. Furthermore, $x + y$ and z must be greater than $\max\{3s^{2/3}, 3/b^{2/3}\}$, where b is the largest floating point number. If any of these conditions is false, then `IMSL_ELRD` returns b .

The `IMSL_ELRD` function is based on the code by Carlson and Notis (1981) and the work of Carlson (1979).

Example

The integral $R_D(0, 2, 1)$ is computed.

```
PRINT, IMSL_ELRD(0.0, 2.0, 1.0)
1.79721
```

Version History

6.4	Introduced

IMSL_ELRJ

The IMSL_ELRJ function evaluates Carlson's elliptic integral of the third kind $R_J(x, y, z, \rho)$.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_ELRJ(*x*, *y*, *z*, *rho* [, /DOUBLE])

Return Value

The complete elliptic integral $R_J(x, y, z, \rho)$.

Arguments

rho

Fourth argument for which the function value is desired. It must be positive.

x

First argument for which the function value is desired. It must be nonnegative.

y

Second argument for which the function value is desired. It must be nonnegative.

z

Third argument for which the function value is desired. It must be positive.

Keywords

DOUBLE

If present and nonzero, double precision is used.

Discussion

Carlson's elliptic integral of the third kind is defined to be:

$$R_J(x, y, z, \rho) = \frac{3}{2} \int_0^\infty \frac{dt}{[(t+x)(t+y)(t+z)(t+\rho)^3]^{1/2}}$$

The arguments must be nonnegative. In addition, $x+y$, $x+z$, $y+z$ and ρ must be greater than or equal to $(5s)^{1/3}$ and less than or equal to $0.3(b/5)^{1/3}$, where s is the smallest representable floating-point number. Should any of these conditions fail IMSL_ELRJ is set to b , the largest floating-point number.

The IMSL_ELRJ function is based on the code by Carlson and Notis (1981) and the work of Carlson (1979).

Example

The integral $R_J(2, 3, 4, 5)$ is computed.

```
PRINT, IMSL_ELRJ(2.0, 3.0, 4.0, 5.0)
0.142976
```

Version History

6.4	Introduced

IMSL_ELRC

The IMSL_ELRC function evaluates an elementary integral from which inverse circular functions, logarithms and inverse hyperbolic functions can be computed.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_ELRC(*x*, *y* [, /DOUBLE])

Return Value

The elliptic integral $R_C(x, y)$.

Arguments

x

First argument for which the function value is desired. It must be nonnegative and must satisfy the conditions given below.

y

Second argument for which the function value is desired. It must be nonnegative and must satisfy the conditions given below.

Keywords

DOUBLE

If present and nonzero, double precision is used.

Discussion

Carlson's elliptic integral of the third kind is defined to be:

$$R_C(x,y) = \frac{1}{2} \int_0^\infty \frac{dt}{[(t+x)(t+y)^2]^{1/2}}$$

The argument x must be nonnegative, y must be positive, and $x + y$ must be less than or equal to $b/5$ and greater than or equal to $5s$. If any of these conditions are false, the `IMSL_ELRC` is set to b . Here, b is the largest and s is the smallest representable floating-point number.

The `IMSL_ELRC` function is based on the code by Carlson and Notis (1981) and the work of Carlson (1979).

Example

The integral $R_C(2.25, 2)$ is computed.

```
PRINT, IMSL_ELRC(2.25, 2.0)
0.693147
```

Version History

6.4	Introduced

IMSL_FRESNEL_COSINE

The IMSL_FRESNEL_COSINE function evaluates the cosine Fresnel integral.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_FRESNEL_COSINE(*x* [, /DOUBLE])

Return Value

The value of the cosine Fresnel integral evaluated at *x*.

Arguments

x

Argument for which the function value is desired.

Keywords

DOUBLE

If present and nonzero, double precision is used.

Discussion

The cosine Fresnel integral is defined to be

$$C(x) = \int_0^x \cos\left(\frac{\pi}{2}t^2\right) dt$$

Example

The Fresnel integral $C(1.75)$ is evaluated.

```
PRINT, IMSL_FRESNEL_COSINE(1.75)  
0.321935
```

Version History

6.4	Introduced

IMSL_FRESNEL_SINE

The IMSL_FRESNEL_SINE function evaluates the sine Fresnel integral.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_FRESNEL_SINE(*x* [, /DOUBLE])

Return Value

The value of the sine Fresnel integral evaluated at *x*.

Arguments

x

Argument for which the function value is desired.

Keywords

DOUBLE

If present and nonzero, double precision is used.

Discussion

The sine Fresnel integral is defined to be:

$$S(x) = \int_0^x \sin\left(\frac{\pi}{2}t^2\right) dt$$

Example

The Fresnel integral $S(1.75)$ is evaluated.

```
PRINT, IMSL_FRESNEL_SINE(1.75)
```


0.499385

Version History

6.4	Introduced

IMSL_AIRY_AI

The IMSL_AIRY_AI function evaluates the Airy function.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_AIRY_AI(*x* [, **DERIVATIVE**=*value*] [, /**DOUBLE**])

Return Value

The value of the Airy function evaluated at *x*, $Ai(x)$.

Arguments

x

Argument for which the function value is desired.

Keywords

DERIVATIVE

If present and nonzero, then the derivative of the Airy function is computed.

DOUBLE

If present and nonzero, double precision is used.

Discussion

The airy function $\text{Ai}(x)$ is defined to be:

$$\text{Ai}(x) = \frac{1}{\pi} \int_0^{\infty} \cos\left(xt + \frac{1}{3}t^3\right) dt = \sqrt{\frac{x}{3\pi^2}} K_{1/3}\left(\frac{2}{3}x^{3/2}\right)$$

The Bessel function $K_\nu(x)$ is defined in “[IMSL_BESSK](#)” on page 502.

If $x < -1.31\epsilon^{-2/3}$, then the answer will have no precision. If $x < -1.31\epsilon^{-1/3}$, the answer will be less accurate than half precision. Here ϵ is the machine precision.

x should be less than $x_{\max}$ so the answer does not underflow. Very approximately, $x_{\max} = \{-1.5\ln s\}^{2/3}$, where s = the smallest representable positive number.

If the keyword DERIVATIVE is set, then the airy function $\text{Ai}'(x)$ is defined to be the derivative of the Airy function, $\text{Ai}(x)$ (see the “[IMSL_AIRY_AI](#)” on page 526). If $x < -1.31\epsilon^{-2/3}$, then the answer will have no precision. If $x < -1.31\epsilon^{-1/3}$, the answer will be less accurate than half precision. Here ϵ is the machine precision. x should be less than $x_{\max}$ so the answer does not underflow. Very approximately, $x_{\max} = \{-1.51\ln s\}$, where s is the smallest representable positive number.

Example

In this example, $\text{Ai}(-4.9)$ and $\text{Ai}'(-4.9)$ are evaluated.

```
PRINT, IMSL_AIRY_AI(-4.9)
      0.374536
PRINT, IMSL_AIRY_AI(-4.9, /Derivative)
      0.146958
```

Version History

6.4	Introduced

IMSL_AIRY_BI

The IMSL_AIRY_BI function evaluates the Airy function of the second kind.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_AIRY_BI(*x* [, **DERIVATIVE**=*value*] [, /**DOUBLE**])

Return Value

The value of the Airy function evaluated at *x*, Bi(*x*).

Arguments

x

Argument for which the function value is desired.

Keywords

DERIVATIVE

If present and nonzero, then the derivative of the Airy function of the second kind is computed.

DOUBLE

If present and nonzero, double precision is used.

Discussion

The airy function $\text{Bi}(x)$ is defined to be:

$$\text{Bi}(x) = \frac{1}{\pi} \int_0^{\infty} \exp\left(xt - \frac{1}{3}t^3\right) dt = \frac{1}{\pi} \int_0^{\infty} \sin\left(xt + \frac{1}{3}t^3\right) dt$$

It can also be expressed in terms of modified Bessel functions of the first kind, $I_\nu(x)$, and Bessel functions of the first kind $J_\nu(x)$ (see “[IMSL_BESSI](#)” on page 498, and “[IMSL_BESSJ](#)” on page 500):

$$\text{Bi}(x) = \sqrt{\frac{x}{3}} \left[I_{-1/3}\left(\frac{2}{3}x^{3/2}\right) + I_{1/3}\left(\frac{2}{3}x^{3/2}\right) \right] \quad \text{for } x > 0$$

and:

$$\text{Bi}(x) = \sqrt{\frac{-x}{3}} \left[J_{-1/3}\left(\frac{2}{3}|x|^{3/2}\right) - J_{1/3}\left(\frac{2}{3}|x|^{3/2}\right) \right] \quad \text{for } x < 0$$

Here ϵ is the machine precision. If $x < -1.31\epsilon^{-2/3}$, then the answer will have no precision. If $x < -1.31\epsilon^{-1/3}$, the answer will be less accurate than half precision. In addition, x should not be so large that $\exp[(2/3)x^{3/2}]$ overflows.

If the keyword `DERIVATIVE` is set, the airy function $\text{Bi}'(x)$ is defined to be the derivative of the Airy function of the second kind, $\text{Bi}(x)$ (see “[IMSL_AIRY_BI](#)” on page 528). If $x < -1.31\epsilon^{-2/3}$, then the answer will have no precision. If $x < -1.31\epsilon^{-1/3}$, the answer will be less accurate than half precision. Here ϵ is the machine precision. In addition, x should not be so large that $\exp[(2/3)x^{3/2}]$ overflows.

Example

In this example, $\text{Bi}(-4.9)$ and $\text{Bi}'(-4.9)$ are evaluated.

```
PRINT, IMSL_AIRY_BI(-4.9)
      -0.0577468
PRINT, IMSL_AIRY_BI(-4.9, /Derivative)
      0.827219
```

Version History

6.4	Introduced

IMSL_KELVIN_BER0

The IMSL_KELVIN_BER0 function evaluates the Kelvin function of the first kind, ber, of order zero.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_KELVIN_BER0(*x* [, **DERIVATIVE**=*value*] [, **/DOUBLE**])

Return Value

The value of the Kelvin function of the first kind, ber, of order zero evaluated at *x*.

Arguments

x

Argument for which the function value is desired.

Keywords

DERIVATIVE

If present and nonzero, then the derivative of the Kelvin function of the first kind, ber, of order zero evaluated at *x* is computed.

DOUBLE

If present and nonzero, double precision is used.

Discussion

The Kelvin function $\text{ber}_0(x)$ is defined to be $\Re J_0(xe^{3\pi i/4})$. The Bessel function $J_0(x)$ is defined:

$$J_0(x) = \frac{1}{\pi} \int_0^\pi \cos(x \sin \theta) d\theta$$

If the keyword DERIVATIVE is set, the function $\text{ber}_0'(x)$ is defined to be:

$$\frac{d}{dx} \text{ber}_0(x)$$

If $|x| > 119$, NaN is returned.

The IMSL_KELVIN_BER0 function is based on the work of Burgoyne (1963).

Example

In this example, $\text{ber}_0(0.4)$ and $\text{ber}_0'(0.6)$ are evaluated.

```
PRINT, IMSL_KELVIN_BER0(0.4)
      0.999600
PRINT, IMSL_KELVIN_BER0(0.6, /DERIVATIVE)
     -0.0134985
```

Version History

6.4	Introduced

IMSL_KELVIN_BEI0

The IMSL_KELVIN_BEI0 function evaluates the Kelvin function of the first kind, bei , of order zero.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

$\text{Result} = \text{IMSL\_KELVIN\_BEI0}(x [, \text{DERIVATIVE}=\text{value}] [, /\text{DOUBLE}])$

Return Value

The value of the Kelvin function of the first kind, bei , of order zero evaluated at x .

Arguments

x

Argument for which the function value is desired.

Keywords

DERIVATIVE

If present and nonzero, then the derivative of the Kelvin function of the first kind, bei , of order zero evaluated at x is computed.

DOUBLE

If present and nonzero, double precision is used.

Discussion

The Kelvin function $\text{bie}_0(x)$ is defined to be $\Im J_0(xe^{3\pi i/4})$. The Bessel function $J_0(x)$ is defined:

In IMSL_KELVIN_BEI0, x must be less than 119.

$$J_0(x) = \frac{1}{\pi} \int_0^\pi \cos(x \sin \theta) d\theta$$

If the keyword DERIVATIVE is set, the function $\text{bei}_0'(x)$ is defined to be:

$$\frac{d}{dx} \text{bei}_0(x)$$

If the keyword DERIVATIVE is set and $|x| > 119$, NaN is returned.

The IMSL_KELVIN_BEI0 function is based on the work of Burgoyne (1963).

Example

In this example, $\text{bei}_0(0.4)$ and $\text{bei}_0'(0.6)$ are evaluated.

```
PRINT, IMSL_KELVIN_BEI0(0.4)
      0.0399982
PRINT, IMSL_KELVIN_BEI0(0.6, /DERIVATIVE)
      0.299798
```

Version History

6.4	Introduced

IMSL_KELVIN_KERO

The KELVIN_KERO function evaluates the Kelvin function of the second kind, $\ker$, of order zero.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_KELVIN_KERO(*x* [, DERIVATIVE=*value*] [, /DOUBLE])

Return Value

The value of the Kelvin function of the second kind, $\ker$, of order zero evaluated at x .

Arguments

x

Argument for which the function value is desired.

Keywords

DERIVATIVE

If present and nonzero, then the derivative of the Kelvin function of the second kind, $\ker$, of order zero evaluated at x is computed.

DOUBLE

If present and nonzero, double precision is used.

Discussion

The modified Kelvin function $\ker_0(x)$ is defined to be $\Re K_0(xe^{pi/4})$. The Bessel function $K_0(x)$ is defined:

If the keyword DERIVATIVE is set, the function $\ker'_0(x)$ is defined to be:

$$K_0(x) = \int_0^\pi \cos(x \sin t) dt$$

$$\frac{d}{dx} \text{ker}_0(x)$$

If $x < 0$, NaN (Not a Number) is returned. If $x \geq 119$, then zero is returned.

The IMSL_KELVIN_KER0 function is based on the work of Burgoyne (1963).

Example

In this example, $\text{ker}_0(0.4)$ and $\text{ker}_0'(0.6)$ are evaluated.

```
PRINT, IMSL_KELVIN_KER0 (0.4)
      1.06262
PRINT, IMSL_KELVIN_KER0 (0.6, /DERIVATIVE)
     -1.45654
```

Version History

6.4	Introduced

IMSL_KELVIN_KEI0

The IMSL_KELVIN_KEI0 function evaluates the Kelvin function of the second kind, kei, of order zero.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_KELVIN_KEI0(*x* [, **DERIVATIVE**=*value*] [, **/DOUBLE**])

Return Value

The value of the Kelvin function of the second kind, kei, of order zero evaluated at *x*.

Arguments

x

Argument for which the function value is desired.

Keywords

DERIVATIVE

If present and nonzero, then the derivative of the Kelvin function of the second kind, kei, of order zero evaluated at *x* is computed.

DOUBLE

If present and nonzero, double precision is used.

Discussion

The modified Kelvin function $\text{kei}_0(x)$ is defined to be $\Im K_0(xe^{3\pi i/4})$. The Bessel function $K_0(x)$ is defined as:

$$K_0(x) = \int_0^\infty \cos(x \sin t) dt$$

If the keyword DERIVATIVE is set, the function $\text{kei}_0'(x)$ is defined to be:

$$\frac{d}{dx} \text{kei}_0(x)$$

The IMSL_KELVIN_KEI0 function is based on the work of Burgoyne (1963).

If $x < 0$, NaN (Not a Number) is returned. If $x \geq 119$, zero is returned.

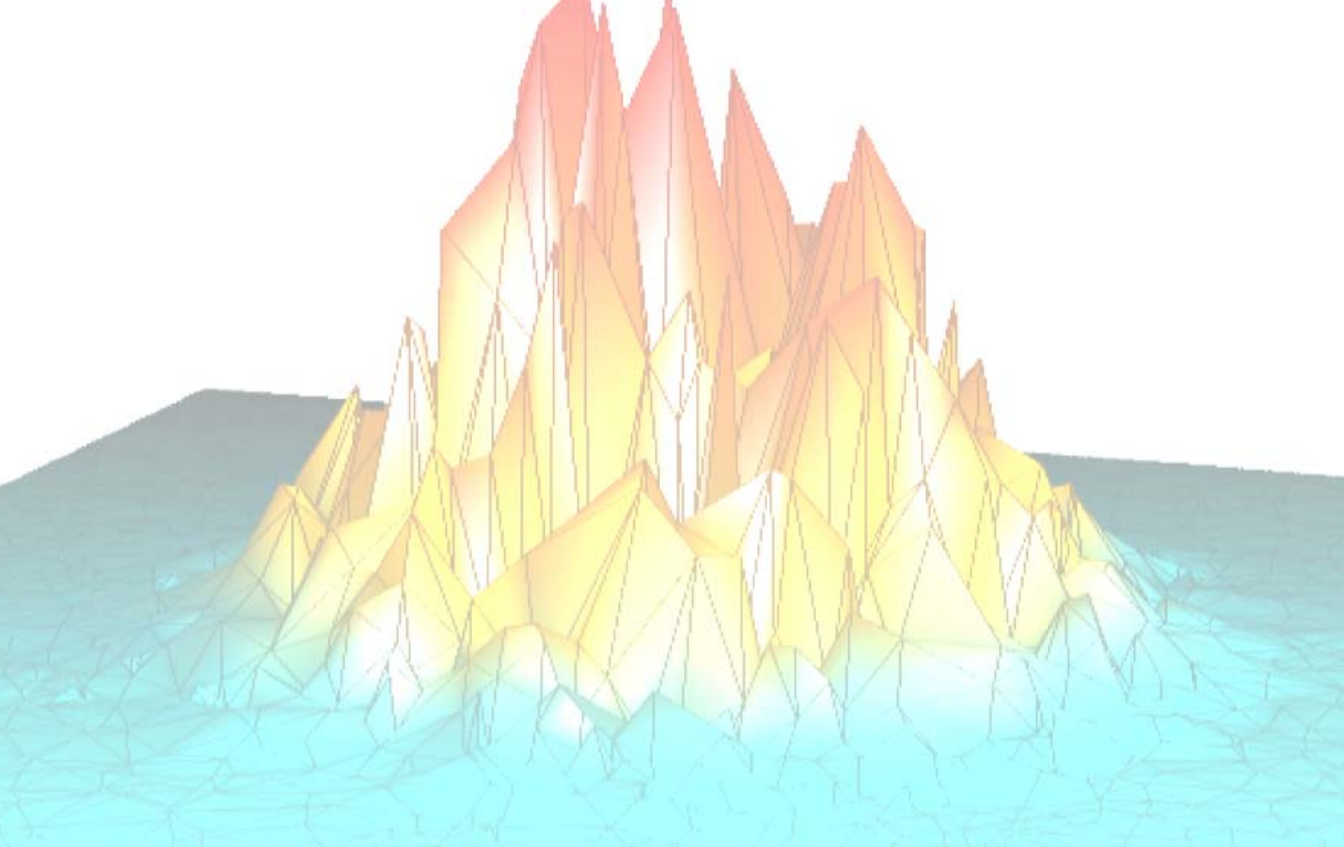
Example

In this example, $\text{kei}_0(0.4)$ and $\text{kei}_0'(0.6)$ are evaluated.

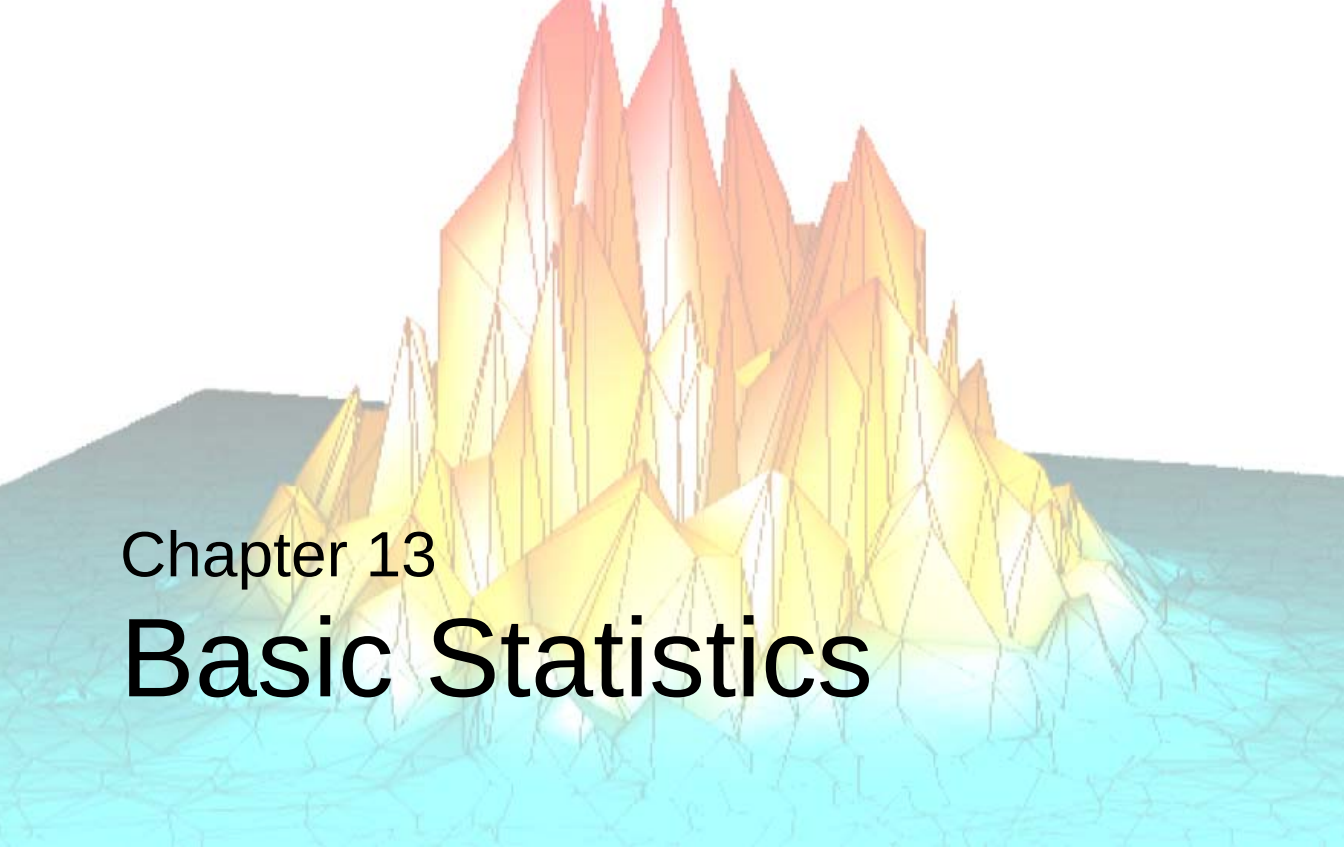
```
PRINT, IMSL_KELVIN_KEI0(0.4)
      -0.703800
PRINT, IMSL_KELVIN_KEI0(0.6, /DERIVATIVE)
      0.348164
```

Version History

6.4	Introduced



Part II: Statistics Routines



Chapter 13

Basic Statistics

This section contains the following topics:

Overview: Basic Statistics	542	Basic Statistics Routines	543

Overview: Basic Statistics

The functions for computations of basic statistics generally have relatively simple input parameters. The data are input in either a one- or two-dimensional array. As usual, when a two-dimensional array is used, the rows contain observations and the columns represent variables. Most of the functions in this chapter allow for missing values. Missing value codes can be set using [IMSL_MACHINE](#).

Several functions in this chapter perform statistical tests. These functions generally return a “ p -value” for the test, often as the return value for the C function. The p -value is between 0 and 1 and is the probability of observing data that would yield a test statistic as extreme or more extreme under the assumption of the null hypothesis. Hence, a small p -value is evidence for the rejection of the null hypothesis.

Basic Statistics Routines

Simple Summary Statistics

[IMSL_SIMPLESTAT](#)—Univariate summary statistics.

[IMSL_NORM1SAMP](#)—Mean and variance inference for a single normal population.

[IMSL_NORM2SAMP](#)—Inferences for two normal populations.

Tabulate, Sort, and Rank

[IMSL_FREQTABLE](#)—Tallies observations into a one-way frequency table.

[IMSL_SORTDATA](#)—Sorts data with options to tally cases into a multiway frequency table.

[IMSL_RANKS](#)—Ranks, normal scores, or exponential scores.

IMSL_SIMPLESTAT

The IMSL_SIMPLESTAT function computes basic univariate statistics.

Note _____
This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_SIMPLESTAT(*x*)

Return Value

A two-dimensional matrix containing some simple statistics for each variable *x*. If *Median* and *Median_And_Scale* are not used as keywords, then element (*i*, *j*) of the returned matrix contains the *i*-th statistic of the *j*-th variable. Refer to [Table 13-1](#) for a list of results.

<i>i</i>	Statistic Returned in Element (<i>i</i> , *)
0	mean
1	variance
2	standard deviation
3	coefficient of skewness
4	coefficient of excess (kurtosis)
5	minimum value
6	maximum value
7	range
8	coefficient of variation (when defined) If the coefficient of variation is not defined, zero is returned.

Table 13-1: IMSL_SIMPLESTAT Results

<i>i</i>	Statistic Returned in Element (<i>i</i> , *)
9	number of observations (the counts)
10	lower confidence limit for the mean (assuming normality) The default is a 95-percent confidence interval.
11	upper confidence limit for the mean (assuming normality)
12	lower confidence limit for the variance (assuming normality) The default is a 95-percent confidence interval.
13	upper confidence limit for the variance (assuming normality)

Table 13-1: *IMSL_SIMPLESTAT Results (Continued)*

Arguments

x

Data matrix. The data value for the *i*-th observation of the *j*-th variable should be in the matrix element (*i*, *j*).

Keywords

CONF_MEANS

Scalar specifying the confidence level for a two-sided interval estimate of the means (assuming normality) in percent. The CONF_MEANS keyword must be between 0.0 and 100.0 and is often 90.0, 95.0, or 99.0. For a one-sided confidence interval with confidence level *c*, set CONF_MEANS = 100.0 – 2.0(100.0 – *c*) (at least 50 percent). Default: 95-percent confidence interval is computed

CONF_VARIANCES

Confidence level for a two-sided interval estimate of the variances (assuming normality) in percent. The confidence intervals are symmetric in probability (rather than in length). For one-sided confidence interval with confidence level *c*, set CONF_MEANS = 100.0 – 2.0(100.0 – *c*) (at least 50 percent). Default: 95-percent confidence interval is computed.

DOUBLE

If present and nonzero, double precision is used.

ELEMENTWISE

If present and nonzero, all nonmissing data for any variable is used in computing the statistics for that variable. Default action: if an observation (row of x) contains a missing value, the observation is excluded from computations for all variables. In either case, if weights and/or frequencies are specified and the value of the weight and/or frequency is missing, the observation is excluded from computations for all variables.

FREQUENCIES

One-dimensional array containing the frequency for each observation. Default: each observation has a frequency of 1

MEDIAN_ONLY

If present and nonzero, medians are computed and stored in elements (14, *) of the returned matrix of simple statistics. The `MEDIAN_ONLY` and `MEDIAN_AND_SCALE` keywords cannot be used together.

MEDIAN_AND_SCALE

If present and nonzero, specified, the medians, the medians of the absolute deviations from the medians, and a simple robust estimate of scale are computed and stored in elements (14, *), (15, *), and (16, *) of the returned matrix of simple statistics. The `MEDIAN_ONLY` and `MEDIAN_AND_SCALE` keywords cannot be used together.

WEIGHTS

One-dimensional array containing the weight for each observation. Default: each observation has a weight of 1.

Discussion

The `IMSL_SIMPLESTAT` function computes the sample mean, variance, minimum, maximum, and other basic statistics for the data in x . It also computes confidence intervals for the mean and variance (under the hypothesis that the sample is from a normal population).

Frequencies, f_i 's, are interpreted as multiple occurrences of the other values in the observations. In other words, a row of x with a frequency variable having a value of 2 has the same effect as two rows with frequencies of 1. The total of the frequencies is used in computing all the statistics based on moments (mean, variance, skewness, and kurtosis). Weights, w_i 's, are not viewed as replication factors. The sum of the weights

is used only in computing the mean (the weighted mean is used in computing the central moments). Both weights and frequencies can be zero, but neither can be negative. In general, a zero frequency means that the row is to be eliminated from the analysis; no further processing or error checking is done on the row. A weight of zero results in the row being counted, and updates are made of the statistics.

The definitions of some of the statistics are given below in terms of a single variable x of which the i -th datum is x_i .

Mean

$$\bar{x}_w = \frac{\sum f_i w_i x_i}{\sum f_i w_i}$$

Variance

$$s_w^2 = \frac{\sum f_i w_i (x_i - \bar{x}_w)^2}{n - 1}$$

Skewness

$$\frac{\sum f_i w_i (x_i - \bar{x}_w)^3 / n}{\left[\sum f_i w_i (x_i - \bar{x}_w)^2 / n \right]^{3/2}}$$

Excess or Kurtosis

$$\frac{\sum f_i w_i (x_i - \bar{x}_w)^4 / n}{\left[\sum f_i w_i (x_i - \bar{x}_w)^2 / n \right]^2} - 3$$

Minimum

$$x_{\min} = \min(x_i)$$

Maximum

$$x_{\max} = \max(x_i)$$

Range

$$x_{\max} - x_{\min}$$

Coefficient of Variation

$$\frac{s_w}{\bar{x}_w} \quad \text{for } \bar{x} \neq 0$$

Median

$$\text{median}\{x_i\} = \begin{cases} \text{middle } x_i \text{ after sorting if } n \text{ is odd} \\ \text{average of middle two } x_i\text{'s if } n \text{ is even} \end{cases}$$

Median Absolute Deviation

$$\text{MAD} = \text{median}\{ |x_i - \text{median}\{x_j\}| \}$$

Simple Robust Estimate of Scale

$$\text{MAD} / \Phi^{-1}(3/4)$$

$$\text{where } \Phi^{-1}(3/4) \approx 0.6745$$

is the inverse of the standard normal distribution function evaluated at 3/4. This standardizes MAD in order to make the scale estimate consistent at the normal distribution for estimating the standard deviation (Huber 1981, pp. 107–108).

Example

This example uses data from Draper and Smith (1981). There are five variables and 13 observations.

```
x = IMSL_STATDATA(5)
stats = IMSL_SIMPLESTAT(x)
; Call IMSL_SIMPLESTAT.
labels = ['means', 'variances', 'std. dev', $
         'skewness', 'kurtosis', 'minima', $
         'maxima', 'ranges', 'C.V.', 'counts', $
         'lower mean', 'upper mean', 'lower var', 'upper var']
; Define the character strings that will be used as labels for the
; rows of the output.
FOR i = 0, 13 DO PM, labels(i), stats(i, *), $
    FORMAT = '(a10, 5f9.3)'
; Output the results.
```

means	7.462	48.154	11.769	30.000	95.423
variances	34.603	242.141	41.026	280.167	226.314
std. dev	5.882	15.561	6.405	16.738	15.044
skewness	0.688	-0.047	0.611	0.330	-0.195
kurtosis	0.075	-1.323	-1.079	-1.014	-1.342

minima	1.000	26.000	4.000	6.000	72.500
maxima	21.000	71.000	23.000	60.000	115.900
ranges	20.000	45.000	19.000	54.000	43.400
C.V.	0.788	0.323	0.544	0.558	0.158
counts	13.000	13.000	13.000	13.000	13.000
lower mean	3.907	38.750	7.899	19.885	86.332
upper mean	11.016	57.557	15.640	40.115	104.514
lower var	17.793	124.512	21.096	144.065	116.373
upper var	94.289	659.817	111.792	763.434	616.688

Version History

6.4	Introduced

IMSL_NORM1SAMP

The IMSL_NORM1SAMP function computes statistics for mean and variance inferences using a sample from a normal population.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_NORM1SAMP(x [, CHI_SQ_NULL_HYP=value]
    [, CI_MEAN=variable] [, CI_VAR=variable] [, CHI_SQ_TEST=variable]
    [, CONF_MEAN=value] [, CONF_VAR=value] [, /DOUBLE]
    [, STDEV=variable] [, T_NULL_HYP=value] [, T_TEST=variable])
```

Return Value

The mean of the sample.

Arguments

x

One-dimensional array containing the observed values.

Keywords

CHI_SQ_NULL_HYP

Null hypothesis value for the chi-squared test for the variance. Default:
CHI_SQ_NULL_HYP = 1.0

CI_MEAN

Named variable into which the two-element array containing the lower confidence limit for the mean, and the upper confidence limit for the mean is stored.

CI_VAR

Named variable into which the two-element array containing lower and upper confidence limits for the variance is stored.

CHI_SQ_TEST

Named variable into which the three-element array containing statistics associated with the chi-squared test is stored. The first element contains the degrees of freedom associated with the chi-squared test for variances, the second element contains the test statistic, and the third element contains the probability of a larger chi-squared value. The chi-squared test is a test of the hypothesis $\sigma^2 = \sigma_0^2$, where σ_0^2 is the null hypothesis value as described in CHI_SQ_NULL_HYP.

CONF_MEAN

Confidence level (in percent) for two-sided interval estimate of the mean. The keyword CONF_MEAN must be between 0.0 and 100.0 and is often 90.0, 95.0, or 99.0. For a one-sided confidence interval with confidence level c (at least 50 percent), set CONF_MEAN = $100.0 - 2.0 \times (100.0 - c)$. Default: 95-percent confidence interval is computed.

CONF_VAR

Confidence level (in percent) for two-sided interval estimate of the variances. Keyword CONF_VAR must be between 0.0 and 100.0 and is often 90.0, 95.0, or 99.0. For a one-sided confidence interval with confidence level c (at least 50 percent), set CONF_VAR = $100.0 - 2.0 \times (100.0 - c)$. Default: 95-percent confidence interval is computed.

DOUBLE

If present and nonzero, double precision is used.

STDEV

Variable into which the standard deviation of the sample is stored.

T_NULL_HYP

Null hypothesis value for t test for the mean. Default: T_NULL_HYP = 0.0

T_TEST

Named variable into which the three-element array containing statistics associated with the t test is stored. The first element contains the degrees of freedom associated with the t test for the mean, the second element contains the test statistic, and the third element contains the probability of a larger t in absolute value. The t test is a test of the hypothesis $\mu = \mu_0$, where μ_0 is the null hypothesis value as described in T_NULL_HYP.

Discussion

Statistics for mean and variance inferences using a sample from a normal population are computed, including confidence intervals and tests for both mean and variance. The definitions of mean and variance are given below. The summation in each case is over the set of valid observations, based on the presence of missing values in the data.

Mean, return value

$$\bar{x} = \frac{\sum x_i}{n}$$

Standard deviation

$$s = \sqrt{\frac{\sum (x_i - \bar{x})^2}{n - 1}}$$

The t statistic for the two-sided test concerning the population mean is given by:

$$t = \frac{\bar{x} - \mu_0}{s/\sqrt{n}}$$

where s and $\bar{x}$

are given above. This quantity has a T distribution with $n - 1$ degrees of freedom.

The chi-squared statistic for the two-sided test concerning the population variance is given by:

$$\chi^2 = \frac{(n - 1)s^2}{\sigma_0^2}$$

where s is given above. This quantity has a χ^2 distribution with $n - 1$ degrees of freedom.

Examples

Example 1

This example uses data from Devore (1982, p. 335), which is based on data published in the *Journal of Materials*. There are 15 observations; the mean is the only output.

```
x = [26.7, 25.8, 24.0, 24.9, 26.4, $
25.9, 24.4, 21.7, 24.1, 25.9, $
27.3, 26.9, 27.3, 24.8, 23.6]
PRINT, 'Sample Mean = ', IMSL_NORM1SAMP(x)
Sample Mean = 25.3133
```

Example 2

This example uses the same data as the initial example. The hypothesis $H_0: \mu = 20.0$ is tested. The extremely large t value and the correspondingly small p -value provide strong evidence to reject the null hypothesis. First, a procedure to print the results is defined.

```
.RUN
PRO print_results, mean, stdev, $
    ci_mean, t_test
    PM, mean, Title = 'Sample Mean:'
    PM, stdev, Title = 'Sample Standard Deviation:'
    PM, '(', ci_mean(0), ci_mean(1), ')', $
        Title = '95% CI for the mean:'
    PM, ' '
    PM, ' df = ', t_test(0), Title = 't-test statistics:'
    PM, ' t = ', t_test(1)
    PM, ' p-value = ', t_test(2)
END

x = [26.7, 25.8, 24.0, 24.9, 26.4, 25.9, 24.4, $
21.7, 24.1, 25.9, 27.3, 26.9, 27.3, 24.8, 23.6]
mean = IMSL_NORM1SAMP(x, Stdev = stdev, Ci_Mean = ci_mean, $
    T_Null_Hyp = 40.0, T_Test = t_test)
print_results, mean, stdev, ci_mean, t_test

Sample Mean:
    25.3133
Sample Standard Deviation:
    1.57882
95% CI for the mean:
    (    24.4390    26.1877)

t-test statistics:
    df      =    14.0000
```

t = -36.0277
p-value = 0.00000

Version History

6.4	Introduced

IMSL_NORM2SAMP

The IMSL_NORM2SAMP function computes statistics for mean and variance inferences using samples from two independently normal populations.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_NORM2SAMP(x1, x2 [, CI_DIFF_EQ_VAR=variable]
    [, CI_DIFF_NE_VAR=variable] [, CONF_MEAN=value] [, CONF_VAR=value]
    [, CI_COMM_VAR=variable] [, CI_RATIO_VAR=variable]
    [, CHI_SQ_NULL_HYP=value] [, CHI_SQ_TEST=variable] [, /DOUBLE]
    [, F_TEST=variable] [, MEAN_X1=value] [, MEAN_X2=value]
    [, POOLED_VAR=variable] [, STDEV_X1=variable] [, STDEV_X2=variable]
    [, T_TEST_EQ_VAR=variable] [, T_TEST_NE_VAR=variable]
    [, T_TEST_NULL_HYP=value])
```

Return Value

Difference in means of the mean of the second sample from the first sample.

Arguments

x1

One-dimensional array containing the first sample.

x2

One-dimensional array containing the second sample.

Keywords

CI_DIFF_EQ_VAR

Named variable into which the two-element array containing the lower confidence limit and the upper limit for the mean of the first population minus the mean of the second, assuming equal variances is stored.

CI_DIFF_NE_VAR

Named variable into which the two-element array containing the lower confidence limit and the upper limit for the mean of the first population minus the mean of the second, assuming unequal variances, is stored.

CONF_MEAN

Confidence level for two-sided interval estimate of the mean of x_1 minus the mean of x_2 , in percent. The keyword CONF_MEAN must be between 0.0 and 100.0 and is often 90.0, 95.0, or 99.0. For a one-sided confidence interval with confidence level c (at least 50 percent), set $\text{CONF\_MEAN} = 100.0 - 2.0 \times (100.0 - c)$. Default: CONF_MEAN = 95.0

CONF_VAR

Confidence level for inference on variances. Under the assumption of equal variances, the pooled variance is used to obtain a two-sided CONF_VAR percent confidence interval for the common variance if CI_COMM_VAR is specified. Without making the assumption of equal variances, the ratio of the variances is of interest. A two-sided CONF_VAR percent confidence interval for the ratio of the variance of the first sample to that of the second sample is computed and is returned if CI_RATIO_VAR is specified. The confidence intervals are symmetric in probability. Default: CONF_VAR = 95.0

CI_COMM_VAR

Named variable into which the two-element array containing the lower confidence limit and the upper confidence limit for the common (or pooled) variance is stored.

CI_RATIO_VAR

Named variable into which the two-element array containing the approximate lower confidence limit and the approximate upper confidence limit for the ratio of the variance of the first population to the second is stored.

CHI_SQ_NULL_HYP

Null hypothesis value for the chi-squared test. Default: `CHI_SQ_NULL_HYP` = 1.0

CHI_SQ_TEST

Named variable into which the three-element array containing statistics associated with the chi-squared test for $\sigma^2 = \sigma_0^2$, where σ^2 is the common (or pooled) variance and σ_0^2 is the null hypothesis value, is stored. (See the description for `CHI_SQ_NULL_HYP`.) The first element contains the degrees of freedom, the second element contains the chi-squared value, and the third element contains the probability of a larger chi-squared value, p -value. This test assumes equal variances.

DOUBLE

If present and nonzero, double precision is used.

F_TEST

Named variable into which the four-element array containing statistics associated with the F test for equality of variances is stored. The first element contains the degrees of freedom for the numerator, the second element contains the degrees of freedom for the denominator, the third element contains the F test value, and the fourth element contains the probability of a larger F value, p -value, assuming the null hypothesis ($H_0: \sigma_1^2 = \sigma_2^2$) is true.

MEAN_X1

Means of the first sample.

MEAN_X2

Means of the second sample.

POOLED_VAR

Named variable into which the pooled variance for the two samples is stored.

STDEV_X1

Named variable into which the standard deviation of the first sample is stored.

STDEV_X2

Named variable into which the standard deviation of the second sample is stored.

T_TEST_EQ_VAR

Variable into which the three-element array containing statistics associated with a t test for $\mu_1 - \mu_2 = d$, where d is the null hypothesis value, is stored. (See the description of T_TEST_NULL_HYP.) The first element contains degrees of freedom, second element contains the t value, and third element contains the probability of a larger t in absolute value, assuming the null hypothesis is true. This test assumes equal variances.

T_TEST_NE_VAR

Named variable into which the three-element array containing statistics associated with a t test for $\mu_1 - \mu_2 = d$, where d is the null hypothesis value, is stored. (See the description for T_TEST_NULL_HYP.) The first element contains the degrees of freedom for Satterthwaite's approximation, the second element contains the t value, and the third element contains the probability of a larger t in absolute value, assuming the null hypothesis is true. This test does not assume equal variances.

T_TEST_NULL_HYP

Null hypothesis value for the t test. Default: T_TEST_NULL_HYP = 0.0

Discussion

The IMSL_NORM2SAMP function computes statistics for making inferences about the means and variances of two normal populations, using independent samples in $x1$ and $x2$. For inferences concerning parameters of a single normal population, see [“IMSL_NORM1SAMP”](#) on page 550.

Let μ_1 and σ_1^2 be the mean and variance of the first population, and let μ_2 and σ_2^2 be the corresponding quantities of the second population. The function contains test statistics and confidence intervals for difference in means, equality of variances, and the pooled variance.

The means and variances for the two samples are as follows:

$$\left[\bar{x}_1 = \left(\sum x_{1i} / n_1 \right), \quad \bar{x}_2 = \left(\sum x_{2i} \right) / n_2 \right]$$

and:

$$s_1^2 = \sum \frac{(x_{1i} - \bar{x}_1)^2}{(n_1 - 1)}, s_2^2 = \sum \frac{(x_{2i} - \bar{x}_2)^2}{(n_2 - 1)}$$

Inferences about the Means

The test that the difference in means equals a certain value, for example, μ_0 , depends on whether or not the variances of the two populations can be considered equal. If the variances are equal and `T_TEST_NULL_HYP` equals zero, the test is the two-sample t test, which is equivalent to an analysis-of-variance test. The pooled variance for the difference-in-means test is as follows:

$$s^2 = \frac{(n_1 - 1)s_1 + (n_2 - 1)s_2}{n_1 + n_2 - 2}$$

The t statistic is as follows:

$$t = \frac{\bar{x}_1 - \bar{x}_2 - d}{s\sqrt{(1/n_1) + (1/n_2)}}$$

Also, the confidence interval for the difference in means can be obtained by specifying `CI_DIFF_EQ_VAR`.

If the population variances are not equal, the ordinary t statistic does not have a t distribution and several approximate tests for the equality of means have been proposed. (For example, see Anderson and Bancroft 1952, and Kendall and Stuart 1979.) One of the earliest tests devised for this situation is the Fisher-Behrens test, based on Fisher's concept of fiducial probability. A procedure used if `T_TEST_NE_VAR` and/or `CI_DIFF_NE_VAR` are specified is the Satterthwaite's procedure, as suggested by H.F. Smith and modified by F.E. Satterthwaite (Anderson and Bancroft 1952, p. 83).

The test statistic is:

$$t' = (\bar{x}_1 - \bar{x}_2 - d)/s_d$$

where:

$$s_d = \sqrt{(s_1^2/n_1) + (s_2^2/n_2)}$$

Under the null hypothesis of $\mu_1 - \mu_2 = d$, this quantity has an approximate t distribution with degrees of freedom given by the following equation:

$$df = \frac{s_d^4}{\frac{(s_1^2/n_1)^2}{n_1 - 1} + \frac{(s_2^2/n_2)^2}{n_2 - 1}}$$

Inferences about the Variances

The F statistic for testing the equality of variances is given by:

$$F = s_{\max}^2 / s_{\min}^2,$$

where $s_{\max}^2$ is the maximum of s_1^2 and s_2^2 . If the variances are equal, this quantity has an F distribution with $n_1 - 1$ and $n_2 - 1$ degrees of freedom, where n_1 is the sample size corresponding to $s_{\max}^2$.

Generally, it is not recommended that the results of the F test be used to decide whether to use the regular t test or the modified t' on a single set of data. The modified t' (Satterthwaite's procedure) is the more conservative approach to use if there is doubt about the equality of the variances.

Examples

Example 1

This example, taken from Conover and Iman (1983, p. 294), involves scores on arithmetic tests of two grade-school classes. The question is whether a group taught by an experimental method has a higher mean score. Only the difference in means is output. The data are shown in [Table 13-2](#).

Scores for Standard Group	Scores for Experimental Group
72	111
75	118

Table 13-2: Class Scores

Scores for Standard Group	Scores for Experimental Group
77	128
80	138
104	140
110	150
125	163
	164
	169

Table 13-2: Class Scores

```

x1 = [72, 75, 77, 80, 104, 110, 125]
x2 = [111, 118, 128, 138, 140, 150, 163, 164, 169]
PRINT, 'difference of means = ', IMSL_NORM2SAMP(x1, x2)
difference of means =      -50.4762

```

Example 2

The same data is used for this example as for the initial example. Here, the results of the t test are output. The variances of the two populations are assumed to be equal. It is seen from the output that there is strong reason to believe that the two means are different (t value of -4.804). Since the lower 97.5-percent confidence limit does not include zero, the null hypothesis is that $\mu_1 \leq \mu_2$ would be rejected at the 0.05 significance level. (The closeness of the values of the sample variances provides some qualitative substantiation of the assumption of equal variances.) First, define a procedure to print the results.

```

PRO print_results, diff, sp, ci, t
  PM, diff, Title = 'Difference of Means: '
  PM, sp, Title = 'Pooled Variance: '
  PM, 'CI for Difference of Means is (' , ci(0), ', ', ci(1), ') '
  PM, ' '
  PM, 't-test for Equal Variances:'
  PM, t(0), Title = 'Degrees of Freedom:'
  PM, t(1), Title = 't statistic: '
  PM, t(2), Title = 'P-Value:'
END
x1 = [72, 75, 77, 80, 104, 110, 125]
x2 = [111, 118, 128, 138, 140, 150, 163, 164, 169]
diff = IMSL_NORM2SAMP(x1, x2, Pooled_Var = sp, $

```

```

    Ci_Diff_Eq_Var = ci, T_Test_Eq_Var = t)
print_results, diff, sp, ci, t
Difference of Means:
    -50.4762
Pooled Variance:
    434.633
CI for Difference of Means is
    (    -73.0100,    -27.9424)
t-test for Equal Variances:
Degrees of Freedom:
    14.0000
t statistic:
    -4.80436
P-Value:
    0.000280258

```

Version History

6.4	Introduced

IMSL_FREQTABLE

The IMSL_FREQTABLE function tallies observations into a one-way or two-way frequency table.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_FREQTABLE(x, nxbins[, y, nybins] [, CUTPOINTS=array]  
[, CUTPOINTS=array] [, CLASS_MARKS=array] [, /DOUBLE]  
[, LOWER_BOUND=value] [, UPPER_BOUND=value])
```

Return Value

One-dimensional or two-dimensional array containing the counts.

Arguments

nxbins

Number of intervals (bins) for *x*.

nybins

(Optional) Number of intervals (bins) for *y*.

x

One-dimensional array containing the observations for the first variable.

y

(Optional) One-dimensional array containing the observations for the second variable.

Keywords

CUTPOINTS

(Use this keyword if two positional arguments are used) Specifies a one-dimensional array of length *nxbins* containing the cutpoints to use. This option allows unequal intervals. The initial interval is closed on the right and contains the initial cutpoint as its right endpoint. The last interval is open on the left and includes all values greater than the last cutpoint. The remaining $nxbins - 2$ intervals are open on the left and closed on the right. The argument *nxbins* must be greater than 3 for this option. If CUTPOINTS is used, no other keywords should be specified.

CUTPOINTS

(Use this keyword if four positional arguments are used.) Specifies a one-dimensional array of cutpoints (boundaries). CUTPOINTS must be a one-dimensional array of length $(nxbins - 1) + (nybins - 1)$ containing the cutpoints for *x* in the first $(nxbins - 1)$ elements followed by the cutpoints for *y* in the final $(nybins - 1)$ elements.

CLASS_MARKS

If two positional arguments are used, this keyword specifies a one-dimensional array containing equally spaced class marks in ascending order. The class marks are the midpoints of each of the *nxbins*, and each interval is taken to have length $(CLASS\_MARKS(1) - CLASS\_MARKS(0))$. The argument *nxbins* must be greater than or equal to 2 for this option. If CLASS_MARKS is used, then no other keywords should be specified.

If four positional arguments are used, this keyword specifies a one-dimensional array containing equally spaced class marks in ascending order. The class marks are the midpoints of each interval. The keyword CLASS_MARKS must be a one-dimensional array of length $(nxbins + nybins)$ containing the class marks for *x* in the first *nxbins* elements followed by the class marks for *y* in the final *nybins* elements.

DOUBLE

If present and nonzero, double precision is used.

LOWER_BOUND

If two positional arguments are used, use this keyword and the UPPER_BOUND keyword together to specify two semi-infinite intervals that are used as the initial and last interval. The LOWER_BOUND and UPPER_BOUND keywords must be used together. The initial interval is closed on the right and includes LOWER_BOUND as

its right endpoint. The last interval is open on the left and includes all values greater than `UPPER_BOUND`. The remaining $nxbins - 2$ intervals are of length $(UPPER\_BOUND - LOWER\_BOUND)/(nxbins - 2)$ and are open on the left and closed on the right. The argument *nxbins* must be greater than or equal to 3 for this option.

If four positional arguments are used, use this keyword with the `UPPER_BOUND` keyword to specify intervals of equal lengths. The `LOWER_BOUND` and `UPPER_BOUND` keywords must be used together. See “[Discussion](#)” below for details.

UPPER_BOUND

If two positional arguments are used, use this keyword along with the `LOWER_BOUND` keyword to specify two semi-infinite intervals that are used as the initial and last interval. The `UPPER_BOUND` and `LOWER_BOUND` keywords must be used together. The initial interval is closed on the right and includes `LOWER_BOUND` as its right endpoint. The last interval is open on the left and includes all values greater than `UPPER_BOUND`. The remaining $nxbins - 2$ intervals are of length $(UPPER\_BOUND - LOWER\_BOUND)/(nxbins - 2)$ and are open on the left and closed on the right. `LOWER_BOUND` must also be specified with this keyword. The argument *nxbins* must be greater than or equal to 3 for this option.

If four positional arguments are used, use this keyword with the `LOWER_BOUND` keyword to specify intervals of equal lengths. The `UPPER_BOUND` and `LOWER_BOUND` keywords must be used together. See “[Discussion](#)” below for details.

Discussion

If Two Positional Arguments Are Used

The default action of `IMSL_FREQTABLE` is to group data into *nxbins* categories of size $(\max(x) - \min(x))/nxbins$. The initial interval is closed on the left and open on the right. The remaining intervals are open on the left and closed on the right. Using keywords, the types of intervals used may be changed.

If `UPPER_BOUND` and `LOWER_BOUND` are specified, two semi-infinite intervals are used as the initial and last interval. The initial interval is closed on the right and includes `LOWER_BOUND` as its right endpoint. The last interval is open on the left and includes all values greater than `UPPER_BOUND`. The remaining $nxbins - 2$ intervals are of length $(UPPER\_BOUND - LOWER\_BOUND)/(nxbins - 2)$ and are open on the left and closed on the right. The argument *nxbins* must be greater than or equal to 3 for this option.

If the keyword `CLASS_MARKS` is used, equally spaced class marks in ascending order must be provided in an array of length `nxbins`. The class marks are the midpoints of each of the `nxbins`, and each interval is taken to have the following length:

$$(\text{CLASS\_MARKS}(1) - \text{CLASS\_MARKS}(0))$$

The argument `nxbins` must be greater than or equal to 2 for this option.

If the keyword `CUTPOINTS` is used, cutpoints (bounders) must be provided in an array of length `nxbins`. This option allows unequal intervals. The initial interval is closed on the right and contains the initial cutpoint as its right endpoint. The last interval is open on the left and includes all values greater than the last cutpoint. The remaining `nxbins - 2` intervals are open on the left and closed on the right. The argument `nxbins` must be greater than 3 for this option.

If Four Positional Arguments Are Used

By default, `nxbins` intervals of equal length are used. Let `xmin` and `xmax` be the minimum and maximum values in `x`, respectively, with similar meanings for `ymin` and `ymax`. Then, `table(0, 0)` is the tally of observations with the `x` value less than or equal to `xmin + (xmax - xmin)/nxbins`, and the `y` value less than or equal to `ymin + (ymax - ymin)/ny`.

If `UPPER_BOUND` and `LOWER_BOUND` are specified, intervals of equal lengths are used just as in the default case, except the upper and lower bounds are taken as supplied keywords `xmin = LOWER_BOUND(0)`, `xmax = UPPER_BOUND(0)`, `ymin = LOWER_BOUND(1)`, and `ymax = UPPER_BOUND(1)`, instead of the actual minima and maxima in the data. Therefore, the first and last intervals for both variables are semi-infinite in length.

If `CUTPOINTS` is specified, cutpoints (boundaries) must be provided. The keyword `CUTPOINTS` must be a one-dimensional array of length $(nxbins - 1) + (nybins - 1)$ containing the cutpoints for `x` in the first $(nxbins - 1)$ elements followed by the cutpoints for `y` in the final $(nybins - 1)$ elements.

If `CLASS_MARKS` is specified, equally spaced class marks in ascending order must be provided. The class marks are the midpoints of each interval. The keyword `CLASS_MARKS` must be a one-dimensional array of length $(nxbins + nybins)$ containing the class marks for `x` in the first `nxbins` elements followed by the class marks for `y` in the final `nybins` elements.

Examples

Example 1: One-way Frequency Table

The data for this example is from Hinkley (1977) and Velleman and Hoaglin (1981). Data includes measurements (in inches) of precipitation in Minneapolis/St. Paul during the month of March for 30 consecutive years.

```
x = [0.77, 1.74, 0.81, 1.20, 1.95, 1.20, 0.47, 1.43, 3.37, 2.20, $
      3.00, 3.09, 1.51, 2.10, 0.52, 1.62, 1.31, 0.32, 0.59, 0.81, $
      2.81, 1.87, 1.18, 1.35, 4.75, 2.48, 0.96, 1.89, 0.90, 2.05]
; Define the data set.
table = IMSL_FREQTABLE(x, 10)
; Call IMSL_FREQTABLE with nxbins = 10.
PRINT, '    Bin Number  Count' &$
      PRINT, '    -----  -----' &$
      FOR i = 0, 9 DO PRINT, i + 1, table(i)
```

Bin Number	Count
-----	----
1	4.00000
2	8.00000
3	5.00000
4	5.00000
5	3.00000
6	1.00000
7	3.00000
8	0.00000
9	0.00000
10	1.00000

Example 2: Two-way Frequency Table

The data for x in this example is the same as in the example above. The data for y were created by adding small integers to x .

```
nxbins = 5
nybins = 6
; Define the data set.
x = [0.77, 1.74, 0.81, 1.20, 1.95, 1.20, 0.47, 1.43, 3.37, $
      2.20, 3.00, 3.09, 1.51, 2.10, 0.52, 1.62, 1.31, 0.32, $
      0.59, 0.81, 2.81, 1.87, 1.18, 1.35, 4.75, 2.48, 0.96, $
      1.89, 0.90, 2.05]
y = [1.77, 3.74, 3.81, 2.20, 3.95, 4.20, 1.47, 3.43, 6.37, $
      3.20, 5.00, 6.09, 2.51, 4.10, 3.52, 2.62, 3.31, 3.32, $
      1.59, 2.81, 5.81, 2.87, 3.18, 4.35, 5.75, 4.48, 3.96, $
```

2.89, 2.90, 5.05]

```
; Default usage of IMSL_FREQTABLE
table = IMSL_FREQTABLE(x, nxbins, y, nybins)
PM, table, FORMAT = '(6(F8.5, 2X))', $
    Title = '                                counts'

                                counts
4.00000  2.00000  4.00000  2.00000  0.00000  0.00000
0.00000  4.00000  3.00000  2.00000  1.00000  0.00000
0.00000  0.00000  1.00000  2.00000  0.00000  1.00000
0.00000  0.00000  0.00000  0.00000  1.00000  2.00000
0.00000  0.00000  0.00000  0.00000  0.00000  1.00000
lb = [1, 2]
up = [4, 6]
; Using user-defined bounds
table = IMSL_FREQTABLE(x, nxbins, y, nybins, Upper_Bound = up, $
    Lower_Bound = lb)
PM, table, FORMAT = '(6(F8.5, 2X))', $
    Title = '                                counts'

                                counts
3.00000  2.00000  4.00000  0.00000  0.00000  0.00000
0.00000  5.00000  5.00000  2.00000  0.00000  0.00000
0.00000  0.00000  1.00000  3.00000  2.00000  0.00000
0.00000  0.00000  0.00000  0.00000  0.00000  2.00000
0.00000  0.00000  0.00000  0.00000  1.00000  0.00000
cm = [0.5, 1.5, 2.5, 3.5, 4.5, 1.5, 2.5, 3.5, 4.5, 5.5, 6.5]
; Using class-marks
table = IMSL_FREQTABLE(x, nxbins, y, nybins, Class_Marks = cm)
PM, table, FORMAT = '(6(F8.5, 2X))', $
    Title = '                                counts'

                                counts
3.00000  2.00000  4.00000  0.00000  0.00000  0.00000
0.00000  5.00000  5.00000  2.00000  0.00000  0.00000
0.00000  0.00000  1.00000  3.00000  2.00000  0.00000
0.00000  0.00000  0.00000  0.00000  0.00000  2.00000
0.00000  0.00000  0.00000  0.00000  1.00000  0.00000
cp = [1, 2, 3, 4, 2, 3, 4, 5, 6]
; Using cutpoints
table = IMSL_FREQTABLE(x, nxbins, y, nybins, Cutpoints = cp)
PM, table, FORMAT = '(6(F8.5, 2X))', $
    Title = '                                counts'

                                counts
3.00000  2.00000  4.00000  0.00000  0.00000  0.00000
0.00000  5.00000  5.00000  2.00000  0.00000  0.00000
0.00000  0.00000  1.00000  3.00000  2.00000  0.00000
0.00000  0.00000  0.00000  0.00000  0.00000  2.00000
0.00000  0.00000  0.00000  0.00000  1.00000  0.00000
```

Version History

6.4	Introduced

IMSL_SORTDATA

The IMSL_SORTDATA function sorts observations by specified keys, with option to tally cases into a multiway frequency table.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_SORTDATA(x, n_keys [, ASCENDING=value]
    [, DESCENDING=value] [, /DOUBLE] [, FREQUENCIES=array]
    [, INDICES_KEYS=array] [, LIST_CELLS=variable] [, N_CELLS=variable]
    [, N_LIST_CELLS=variable] [, PERMUTATION=variable]
    [, TABLE_BAL=variable] [, TABLE_N=variable]
    [, TABLE_VALUES=variable] [, TABLE_UNBAL=variable])
```

Return Value

The sorted array.

Arguments

n_keys

Number of columns of *x* on which to sort. The first *n_keys* columns of *x* are used as the sorting keys. (Exception: See the keyword INDICES_KEYS).

x

One- or two-dimensional array containing the observations to be sorted.

Keywords

ASCENDING

If present and nonzero, the sort is in ascending order. (Default) The keywords ASCENDING and DESCENDING cannot be used together.

DESCENDING

If present and nonzero, the sort is in descending order. The keywords `ASCENDING` and `DESCENDING` cannot be used together.

DOUBLE

If present and nonzero, double precision is used.

FREQUENCIES

One-dimensional array containing the frequency for each observation in x . Default: `FREQUENCIES(*) = 1`

INDICES_KEYS

One-dimensional array of length n_keys giving the column numbers of x which are to be used in the sort. Default: `INDICES_KEYS(*) = 0, 1, ..., n\_keys - 1`

LIST_CELLS

Named variable into which the two-dimensional array of length `N_LIST_CELLS` $\times$ n_keys containing, for each row, a list of the levels of n_keys corresponding classification variables that describe a cell, is stored. The keywords `N_LIST_CELLS`, `LIST_CELLS`, and `TABLE_UNBAL` must be used together.

N_CELLS

Named variable into which the a one-dimensional array containing the number of observations per group is stored. A group contains observations (rows) in x that are equal with respect to the method of comparison. The first `N_CELLS` (0) rows of the sorted x are in group number 1. The next `N_CELLS` (1) rows of the sorted x are in group number 2, etc. The last $N\_Cells(N\_ELEMENTS(N\_Cells) - 1)$ rows of the sorted x are in group number `N_ELEMENTS(N_Cells)`.

N_LIST_CELLS

Named variable into which the number of nonempty cells is stored. The keywords `N_LIST_CELLS`, `LIST_CELLS`, and `TABLE_UNBAL` must be used together.

PERMUTATION

Named variable into which a one-dimensional array containing the rearrangement (permutation) of the observations (rows) is stored.

TABLE_BAL

Named variable into which an array of length

$Table\_N(0) + Table\_N(1) + \dots + Table\_N(n\_keys - 1)$, containing the frequencies in the cells of the table to be fit, is stored. Empty cells are included in TABLE_BAL, and each element of TABLE_BAL is nonnegative. The cells of TABLE_BAL are sequenced so that the first variable cycles through its $Table\_N(0)$ categories one time, the second variable cycles through its $Table\_N(1)$ categories $Table\_N(0)$ times, the third variable cycles through its $Table\_N(2)$ categories $Table\_N(0) \times Table\_N(1)$ times, etc., up to the n_keys -th variable, which cycles through its $Table\_N(n\_keys - 1)$ categories:

$$Table\_N(0) + Table\_N(1) + Table\_N(n\_keys - 2)$$

times. The keywords TABLE_N, TABLE_VALUES, and TABLE_BAL must be used together.

TABLE_N

Named variable into which a one-dimensional array of length n_keys , containing in its i -th element ($i = 0, 1, \dots, (n\_keys - 1)$) the number of levels or categories of the i -th classification variable (column), is stored. The keywords TABLE_N, TABLE_VALUES, and TABLE_BAL must be used together.

TABLE_VALUES

Named variable into which an array of length

$Table\_N(0) + Table\_N(1) + \dots + Table\_N(n\_keys - 1)$, containing the values of the classification variables, is stored. The first $Table\_N(0)$ elements of TABLE_VALUES contain the values for the first classification variable. The next $Table\_N(1)$ contain the values for the second variable. The last $Table\_N(n\_keys - 1)$ positions contain the values for the last classification variable. The keywords TABLE_N, TABLE_VALUES, and TABLE_BAL must be used together.

TABLE_UNBAL

Named variable into which the one-dimensional array of length N_LIST_CELLS containing the frequency for each cell is stored. The keywords N_LIST_CELLS, LIST_CELLS, and TABLE_UNBAL must be used together.

Discussion

The IMSL_SORTDATA function can perform both a key sort and/or tabulation of frequencies into a multiway frequency table.

Sorting

The `IMSL_SORTDATA` function sorts the rows of real matrix x using particular columns in x as the keys. The sort is algebraic with the first key as the most significant, the second key as the next most significant, etc. When x is sorted in ascending order, the resulting sorted array is such that the following is true:

- For $i = 0, 1, \dots, N\_ELEMENTS(x(*, 0)) - 2$,
 $x(1, INDICES\_KEYS(0)) \leq x(i + 1, INDICES\_KEYS(0))$
- For $k = 1, \dots, n\_keys - 1$, if
 $x(1, INDICES\_KEYS(j)) = x(i + 1, INDICES\_KEYS(j))$ for
 $j = 0, 1, \dots, k - 1$, then
 $x(1, INDICES\_KEYS(j)) = x(i + 1, INDICES\_KEYS(k))$

The observations also can be sorted in descending order.

The rows of x containing the missing value code NaN in at least one of the specified columns are considered as an additional group. These rows are moved to the end of the sorted x .

The sorting algorithm is based on a quicksort method given by Singleton (1969) with modifications by Griffin and Redish (1970) and Petro (1970).

Frequency Tabulation

The `IMSL_SORTDATA` function determines the distinct values in multivariate data and computes frequencies for the data. This function accepts the data in the matrix x but performs computations only for the variables (columns) in the first n_keys columns of x (Exception: see optional the keyword `INDICES_KEYS`). In general, the variables for which frequencies should be computed are discrete; they should take on a relatively small number of different values. Variables that are continuous can be grouped first. The `IMSL_FREQTABLE` function can be used to group variables and determine the frequencies of groups.

When the `TABLE_N`, `TABLE_VALUES`, and `TABLE_BAL` keywords are specified, `IMSL_SORTDATA` fills the vector `TABLE_VALUES` with the unique values of the variables and tallies the number of unique values of each variable in the vector `TABLE_BAL`. Each combination of one value from each variable forms a cell in a multiway table. The frequencies of these cells are entered in `TABLE_BAL` so that the first variable cycles through its values exactly once and the last variable cycles through its values most rapidly. Some cells cannot correspond to any observations in the data; in other words, “missing cells” are included in the `TABLE_BAL` table and have a value of zero.

When `N_LIST_CELLS`, `LIST_CELLS`, and `TABLE_UNBAL` are specified, the frequency of each cell is entered in `TABLE_UNBAL` so that the first variable cycles through its values exactly once and the last variable cycles through its values most rapidly. All cells have a frequency of at least 1, i.e., there is no “missing cell.” The array `LIST_CELLS` can be considered “parallel” to `TABLE_UNBAL` because row i of `LIST_CELLS` is the set of n_keys values that describes the cell for which row i of `TABLE_UNBAL` contains the corresponding frequency.

Examples

Example 1

The rows of a 10 x 3 matrix `x` are sorted in ascending order using Columns 0 and 1 as the keys. There are two missing values (NaNs) in the keys. The observations containing these values are moved to the end of the sorted array.

```
f = IMSL_MACHINE(/Float)
c0 =[1.0, 2.0, 1.0, 1.0, 2.0, 1.0, f.NaN, 1.0, 2.0, 1.0]
c1 =[1.0, 1.0, 1.0, 1.0, f.NaN, 2.0, 2.0, 1.0, 2.0, 1.0]
c2 =[1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 9.0]
x = [ [c0], [c1], [c2] ]
PM, x, Title = 'Unsorted Matrix'
```

Unsorted Matrix		
1.00000	1.00000	1.00000
2.00000	1.00000	2.00000
1.00000	1.00000	3.00000
1.00000	1.00000	4.00000
2.00000	NaN	5.00000
1.00000	2.00000	6.00000
NaN	2.00000	7.00000
1.00000	1.00000	8.00000
2.00000	2.00000	9.00000
1.00000	1.00000	9.00000

```
PM, IMSL_SORTDATA(x, 2), Title = 'Sorted Matrix'
```

Sorted Matrix:		
1.00000	1.00000	1.00000
1.00000	1.00000	9.00000
1.00000	1.00000	3.00000
1.00000	1.00000	4.00000
1.00000	1.00000	8.00000
1.00000	2.00000	6.00000
2.00000	1.00000	2.00000
2.00000	2.00000	9.00000
NaN	2.00000	7.00000
2.00000	NaN	5.00000

Example 2

This example uses the same data as the previous example. The permutation of the rows is output using the keyword *Permutation*.

```
f = IMSL_MACHINE(/Float)
c0 =[1.0, 2.0, 1.0, 1.0, 2.0, 1.0, f.NaN, 1.0, 2.0, 1.0]
c1 =[1.0, 1.0, 1.0, 1.0, f.NaN, 2.0, 2.0, 1.0, 2.0, 1.0]
c2 =[1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 9.0]
; Fill up a matrix, including some missing values.
x = [ [c0], [c1], [c2] ]
PM, x, Title = 'Unsorted Matrix'
; Output the unsorted matrix.
Unsorted Matrix
1.00000      1.00000      1.0000
2.00000      1.00000      2.00000
1.00000      1.00000      3.00000
1.00000      1.00000      4.00000
2.00000      NaN        5.00000
1.00000      2.00000      6.00000
NaN          2.00000      7.00000
1.00000      1.00000      8.00000
2.00000      2.00000      9.00000
1.00000      1.00000      9.00000
y = IMSL_SORTDATA(x, 2, Permutation = permutation)
; Use IMSL_SORTDATA to sort x.
PM, y, Title = 'Sorted Matrix:'
Sorted Matrix:
1.00000      1.00000      1.00000
1.00000      1.00000      9.00000
1.00000      1.00000      3.00000
1.00000      1.00000      4.00000
1.00000      1.00000      8.00000
1.00000      2.00000      6.00000
2.00000      1.00000      2.00000
2.00000      2.00000      9.00000
NaN          2.00000      7.00000
2.00000      NaN        5.00000
PM, permutation, Title = 'Permutation Matrix:'
; Print the permutation vector.
Permutation Matrix:
0
9
2
3
7
5
1
8
```

```
6
4
z = x(permutation, *)
PM, z, Title = 'Sorted Matrix'
; Use the permutation vector to sort the data.
Sorted Matrix
1.00000      1.00000      1.00000
1.00000      1.00000      9.00000
1.00000      1.00000      3.00000
1.00000      1.00000      4.00000
1.00000      1.00000      8.00000
1.00000      2.00000      6.00000
2.00000      1.00000      2.00000
2.00000      2.00000      9.00000
NaN          2.00000      7.00000
2.00000      NaN         5.00000
```

Version History

6.4	Introduced

IMSL_RANKS

The IMSL_RANKS function computes the ranks, normal scores, or exponential scores for a vector of observations.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_RANKS(x [, AVERAGE_TIE=value] [, BLOM_SCORES=value]  
[, /DOUBLE] [, EXP_NORM_SCORES=value] [, FUZZ=value]  
[, HIGHEST=value] [, LOWEST=value] [, RANDOM_SPLIT=value]  
[, RANKS=value] [, SAVAGE_SCORES=value] [, TUKEY_SCORES=value]  
[, VDW_SCORES=value])
```

Return Value

A one-dimensional array containing the rank (or optionally, a transformation of the rank) of each observation.

Arguments

x

One-dimensional array containing the observations to be ranked.

Keywords

AVERAGE_TIE

Average of the scores of the tied observations (default).

Note

At most, one of these keywords (AVERAGE_TIE, HIGHEST, LOWEST, RANDOM_SPLIT) can be set to a nonzero value to change the method used to assign a score to tied observations.

BLOM_SCORES

Blom version of normal scores.

Note

At most, one of these keywords (RANKS, BLOM_SCORES, TUKEY_SCORES, VDW_SCORES, EXP_NORM_SCORES, SAVAGE_SCORES) can be set to a nonzero value to specify the type of values returned.

DOUBLE

If present and nonzero, double precision is used.

EXP_NORM_SCORES

Expected value of normal order statistics (for tied observations, the average of the expected normal scores)

Note

At most, one of these keywords (RANKS, BLOM_SCORES, TUKEY_SCORES, VDW_SCORES, EXP_NORM_SCORES, SAVAGE_SCORES) can be set to a nonzero value to specify the type of values returned.

FUZZ

Value used to determine when two items are tied. If $\text{ABS}(x(I) - x(J))$ is less than or equal to FUZZ, then $x(I)$ and $x(J)$ are said to be tied. Default: FUZZ = 0.0

HIGHEST

Highest score in the group of ties.

Note

At most, one of these keywords (AVERAGE_TIE, HIGHEST, LOWEST, RANDOM_SPLIT) can be set to a nonzero value to change the method used to assign a score to tied observations.

LOWEST

Lowest score in the group of ties.

Note

At most, one of these keywords (AVERAGE_TIE, HIGHEST, LOWEST, RANDOM_SPLIT) can be set to a nonzero value to change the method used to assign a score to tied observations.

RANDOM_SPLIT

Tied observations are randomly split using a random-number generator.

Note

At most, one of these keywords (AVERAGE_TIE, HIGHEST, LOWEST, RANDOM_SPLIT) can be set to a nonzero value to change the method used to assign a score to tied observations.

RANKS

Ranks (default).

Note

At most, one of these keywords (RANKS, BLOM_SCORES, TUKEY_SCORES, VDW_SCORES, EXP_NORM_SCORES, SAVAGE_SCORES) can be set to a nonzero value to specify the type of values returned.

SAVAGE_SCORES

Savage scores (expected value of exponential order statistics).

Note

At most, one of these keywords (RANKS, BLOM_SCORES, TUKEY_SCORES, VDW_SCORES, EXP_NORM_SCORES, SAVAGE_SCORES) can be set to a nonzero value to specify the type of values returned.

TUKEY_SCORES

Tukey version of normal scores.

Note

At most, one of these keywords (RANKS, BLOM_SCORES, TUKEY_SCORES, VDW_SCORES, EXP_NORM_SCORES, SAVAGE_SCORES) can be set to a nonzero value to specify the type of values returned.

VDW_SCORES

Van der Waerden version of normal scores.

Note

At most, one of these keywords (RANKS, BLOM_SCORES, TUKEY_SCORES, VDW_SCORES, EXP_NORM_SCORES, SAVAGE_SCORES) can be set to a nonzero value to specify the type of values returned.

Discussion

Ties

If the assignment $\text{RANK} = \text{IMSL\_RANKS}(x)$ is made, then in data without ties, the output values are the ordinary ranks (or a transformation of the ranks) of the data in x . If $x(i)$ has the smallest value among the values in x and there is no other element in x with this value, then $\text{RANK}(i) = 1$. If both $x(i)$ and $x(j)$ have the same smallest value, then the output value depends on the option used to break ties. Table 13-3 shows the results for some of the keywords.

Keyword	Result
<i>Average_Tie</i>	$\text{Result}(i) = \text{Result}(j) = 1.5$
<i>Highest</i>	$\text{Result}(i) = \text{Result}(j) = 2.0$
<i>Lowest</i>	$\text{Result}(i) = \text{Result}(j) = 1.0$
<i>Random_Split</i>	$\text{Result}(i) = 1.0$ and $\text{Result}(j) = 2.0$ or, randomly, $\text{Result}(i) = 2.0$ and $\text{Result}(j) = 1.0$

Table 13-3: Tie Results

When the ties are resolved randomly, `IMSL_RANDOM` is used to generate random numbers. Different results occur from different executions of the program unless the “seed” of the random-number generator is set explicitly by use of `IMSL_RANDOMOPT ( )`.

Scores

Normal and other functions of the ranks can optionally be returned. Normal scores can be defined as the expected values, or approximations to the expected values, of order statistics from a normal distribution. The simplest approximations are obtained

by evaluating the inverse cumulative normal distribution function, IMSL_NORMALCDF (with the keyword INVERSE), at the ranks scaled into the open interval (0,1).

In the Blom version (Blom 1958), the scaling transformation for the rank r_i ($1 \leq r_i \leq n$, where n is the sample size) is $(r_i - 3/8) / (n + 1/4)$. The Blom normal score corresponding to the observation with rank r_i is:

$$\Phi^{-1}\left(\frac{r_i - 3/8}{n + 1/4}\right)$$

where $\Phi(\cdot)$ is the normal cumulative distribution function.

Adjustments for ties are made after the normal score transformation; that is, if $x(i)$ equals $x(j)$ (within FUZZ) and their value is the k -th smallest in the data set, the Blom normal scores are determined for ranks of k and $k + 1$. Then, these normal scores are averaged or selected in the manner specified. (Whether the transformations are made first or the ties are resolved first is irrelevant, except when *Average_Tie* is specified.)

In the Tukey version (Tukey 1962), the scaling transformation for the rank r_i is $(r_i - 1/3) / (n + 1/3)$. The Tukey normal score corresponding to the observation with rank r_i follows:

$$\Phi^{-1}\left(\frac{r_i - 1/3}{n + 1/3}\right)$$

Ties are handled in the same way as for the Blom normal scores.

In the Van der Waerden version (see Lehmann 1975, p. 97), the scaling transformation for the rank r_i is $r_i / (n + 1)$. The Van der Waerden normal score corresponding to the observation with rank r_i is as follows:

$$\Phi^{-1}\left(\frac{r_i}{n + 1}\right)$$

Ties are handled in the same way as for the Blom normal scores.

When option EXP_NORM_SCORES is nonzero, the output values are the expected values of the normal order statistics from a sample of size $n = N\_ELEMENTS(x)$. If the value in $x(i)$ is the k -th smallest, then the value output in RANK (i) is $E(z_k)$, where $E(\cdot)$ is the expectation operator, and z_k is the k -th order statistic in a sample of size n from a standard normal distribution. Ties are handled in the same way as for the Blom normal scores.

Savage scores are the expected values of the exponential order statistics from a sample of size n . These values are called Savage scores because of their use in a test discussed by Savage (1956) and Lehmann (1975). If the value in $x(i)$ is the k -th smallest, then the value output in `RANK (i)` is $E(y_k)$ where y_k is the k -th order statistic in a sample of size n from a standard exponential distribution. The expected value of the k -th order statistic from an exponential sample of size n follows:

$$\frac{1}{n} + \frac{1}{n-1} + \dots + \frac{1}{n-k+1}$$

Ties are handled in the same way as for the Blom normal scores.

Example

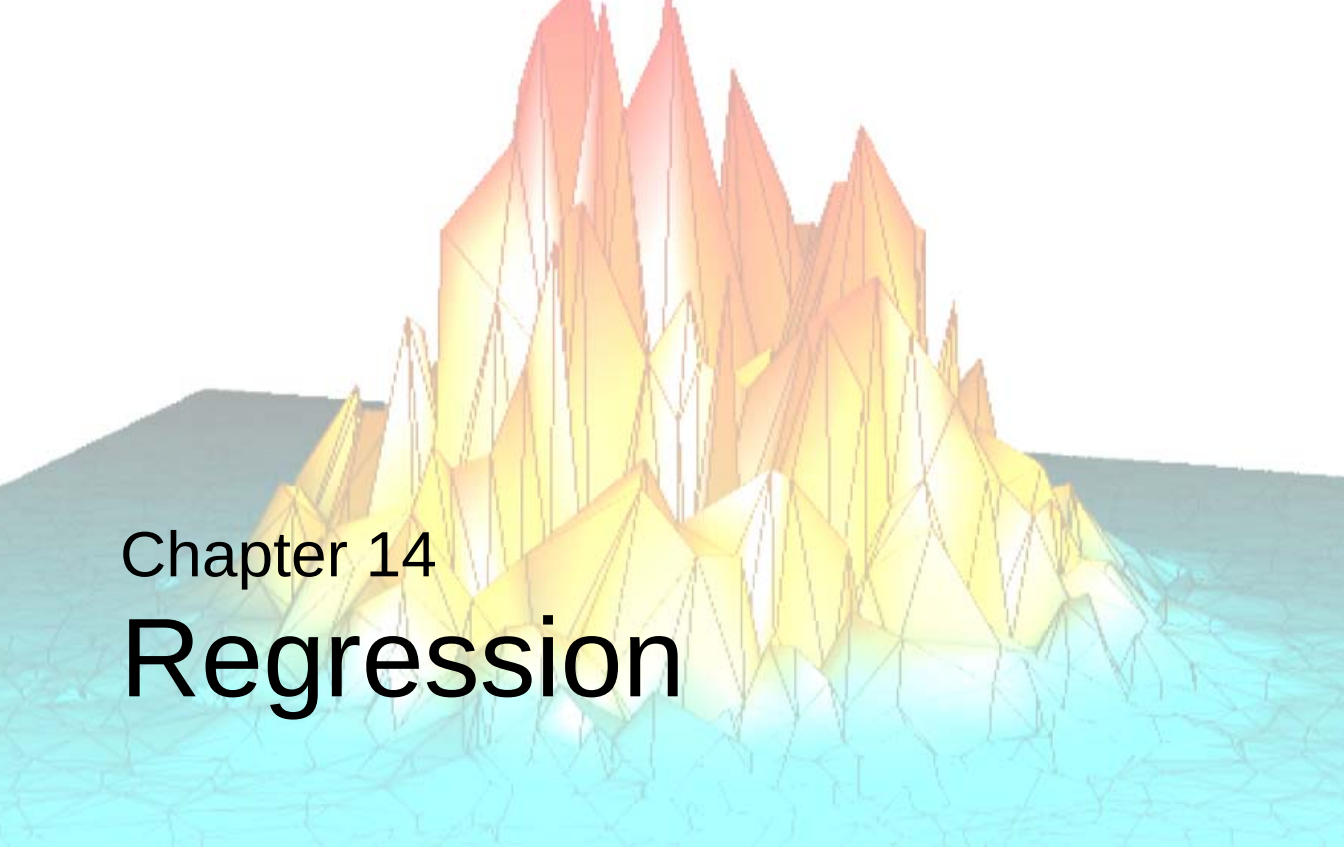
The data for this example, from Hinkley (1977), contains 30 observations. Note that the fourth and sixth observations are tied, and the third and twentieth observations are tied.

```
x = [0.77, 1.74, 0.81, 1.20, 1.95, 1.20, 0.47,$
      1.43, 3.37, 2.20, 3.00, 3.09, 1.51, 2.10,$
      0.52, 1.62, 1.31, 0.32, 0.59, 0.81, 2.81,$
      1.87, 1.18, 1.35, 4.75, 2.48, 0.96, 1.89, 0.90, 2.05]
r = IMSL_RANKS(x)
; Call IMSL_RANKS.
FOR i = 0, 29 DO PM, i + 1, r(i), FORMAT = '(i5, f7.1)'
1      5.0
2     18.0
3      6.5
4     11.5
5     21.0
6     11.5
7      2.0
8     15.0
9     29.0
10     24.0
11     27.0
12     28.0
13     16.0
14     23.0
15      3.0
16     17.0
```

17	13.0
18	1.0
19	4.0
20	6.5
21	26.0
22	19.0
23	10.0
24	14.0
25	30.0
26	25.0
27	9.0
28	20.0
29	8.0
30	22.0

Version History

6.4	Introduced



Chapter 14

Regression

This section contains the following topics:

Overview: Regression	586	Regression Routines	599

Overview: Regression

The regression models in this chapter include the simple and multiple linear regression models, the multivariate general linear model, the polynomial model, and the nonlinear regression model. Functions for fitting regression models, computing summary statistics from a fitted regression, computing diagnostics, and computing confidence intervals for individual cases are provided. Also provided are methods for building a model from a set of candidate variables.

Simple and Multiple Linear Regression

The simple linear regression model is:

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i \quad i = 1, 2, \dots, n$$

where the observed values of the y_i 's constitute the responses or values of the dependent variable, the x_i 's are the settings of the independent (explanatory) variable, β_0 and β_1 are the intercept and slope parameters (respectively), and the ε_i 's are independently distributed normal errors, each with mean zero and variance σ^2 . The multiple linear regression model is:

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_k x_{ik} + \varepsilon_i \quad i = 1, 2, \dots, n$$

where the observed values of the y_i 's constitute the responses or values of the dependent variable; the x_{i1} 's, x_{i2} 's, ..., x_{ik} 's are the settings of the k independent (explanatory) variables; $\beta_0, \beta_1, \dots, \beta_k$ are the regression coefficients; and the ε_i 's are independently distributed normal errors, each with mean zero and variance σ^2 .

“[IMSL_MULTIREGRESS](#)” on page 607 fits both the simple and multiple linear regression models using a fast Given's transformation and includes an option for excluding the intercept β_0 . The responses are input in array y , and the independent variables are input in array x , where the individual cases correspond to the rows and the variables correspond to the columns. In addition to computing the fit, MULTIREGRESS also can optionally compute summary statistics.

After the model has been fitted using [IMSL_MULTIREGRESS](#), “[IMSL_MULTIPREDICT](#)” on page 622 computes predicted values, confidence intervals, and case statistics for the fitted model. The information about the fit is communicated from [IMSL_MULTIREGRESS](#) to [MULTIPREDICT](#) by using keyword *Predict_Info*.

No Intercept Model

Several functions provide the option for excluding the intercept from a model. In most practical applications, the intercept should be included in the model. For functions that use the sum-of-squares and crossproducts matrix as input, the no-intercept case can be handled by using the raw sum-of-squares and crossproducts matrix as input in place of the corrected sum-of-squares and crossproducts. The raw sum-of-squares and crossproducts matrix can be computed as:

$$(x_1, x_2, \dots, x_k, y)^T (x_1, x_2, \dots, x_k, y)$$

Variable Selection

Variable selection can be performed by “[IMSL_ALLBEST](#)” on page 630, which computes all best-subset regressions, or by “[IMSL_STEPWISE](#)” on page 639, which computes stepwise regression. The method used by ALLBEST is generally preferred over that used by STEPWISE because ALLBEST implicitly examines all possible models in the search for a model that optimizes some criterion while stepwise does not examine all possible models. However, the computer time and memory requirements for ALLBEST can be much greater than that for STEPWISE when the number of candidate variables is large.

Polynomial Model

The polynomial model is:

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \dots + \beta_k x_i^k + \epsilon_i \quad i = 1, 2, \dots, n$$

where the observed values of the y_i 's constitute the responses or values of the dependent variable; the x_i 's are the settings of the independent (explanatory) variable; $\beta_0, \beta_1, \dots, \beta_k$ are the regression coefficients; and the ϵ_i 's are independently distributed normal errors each with mean zero and variance σ^2 .

Function “[IMSL_POLYREGRESS](#)” on page 649 fits a polynomial regression model with the option of determining the degree of the model and also produces summary information. Function “[IMSL_POLYPREDICT](#)” on page 657 computes predicted values, confidence intervals, and case statistics for the model fit by POLYREGRESS.

The information about the fit is communicated from `IMSL_POLYREGRESS` to `IMSL_POLYPREDICT` by using keyword *Predict_Info*.

Specification of X for the General Linear Model

Variables used in the general linear model are either continuous or classification variables. Typically, multiple regression models use continuous variables, whereas analysis of variance models use classification variables. Although the notation used to specify analysis of variance models and multiple regression models may look quite different, the models are essentially the same. The term “general linear model” emphasizes that a common notational scheme is used for specifying a model that may contain both continuous and classification variables.

A general linear model is specified by its effects (sources of variation). An effect is referred to in this text as a single variable or a product of variables. (The term “effect” is often used in a narrower sense, referring only to a single regression coefficient.) In particular, an “effect” is composed of one of the following:

- a single continuous variable
- a single classification variable
- several different classification variables
- several continuous variables, some of which may be the same
- continuous variables, some of which may be the same, and classification variables, which must be distinct

Effects of the first type are common in multiple regression models. Effects of the second type appear as main effects in analysis of variance models. Effects of the third type appear as interactions in analysis of variance models. Effects of the fourth type appear in polynomial models and response surface models as powers and crossproducts of some basic variables. Effects of the fifth type appear in analysis of covariance models as regression coefficients that indicate lack of parallelism of a regression function across the groups.

The analysis of a general linear model occurs in two stages. The first stage calls function `“IMSL_REGRESSORS”` on page 600 to specify all regressors except the intercept. The second stage calls `“IMSL_MULTIREGRESS”` on page 607, at which point the model is specified as either having (default) or not having an intercept.

For the sake of this discussion, define a variable *intcep* as shown in [Table 14-1](#):

Option	<i>intcep</i>	Action
No intercept	0	An intercept is not in the model.
Intercept (default)	1	An intercept is in the model.

Table 14-1: intcep Definitions

The remaining parameters and keywords (*n_continuous*, *n_class*, *Class_Columns*, *Var_Effects*, and *Indices_Effects*) are defined for `IMSL_REGRESSORS`. All have defaults except for *n_continuous* and *n_class*, both of which must be specified. (See the documentation for the “[IMSL_REGRESSORS](#)” on page 600 for a discussion of the defaults.) The meaning of each of these input parameters is as follows:

n_continuous—Number of continuous variables.

n_class—Number of classification variables.

Class_Columns—Index vector containing the column numbers of *x* that are the classification variables.

Var_Effects—Vector containing the number of variables associated with each effect in the model.

Indices_Effects—Index vector containing the column numbers of *x* for each variable for each effect.

Suppose the data matrix has as its first four columns two continuous variables in Columns 0 and 1 and two classification variables in Columns 2 and 3. The data might appear as shown in [Table 14-2](#):

Column 0	Column 1	Column 2	Column 3
11.23	1.23	1.0	5.0
12.12	2.34	1.0	4.0
12.34	1.23	1.0	4.0
4.34	2.21	1.0	5.0
5.67	4.31	2.0	4.0
4.12	5.34	2.0	1.0

Table 14-2: Data Matrix

Column 0	Column 1	Column 2	Column 3
4.89	9.31	2.0	1.0
9.12	3.71	2.0	1.0

Table 14-2: Data Matrix (Continued)

Each distinct value of a classification variable determines a level. The classification variable in Column 2 has two levels. The classification variable in Column 3 has three levels. (Integer values are recommended, but not required, for values of the classification variables. The values of the classification variables corresponding to the same level must be identical.)

Some examples of regression functions and their specifications are as shown in [Table 14-3](#):

Regression Functions	<i>intce p</i>	<i>n_clas s</i>	<i>Class_ Column s</i>	<i>Var_Effect s</i>	<i>Indices — Effects</i>
$\beta_0 + \beta_1 x_1$	1	0		1	0
$\beta_0 + \beta_1 x_1 + \beta_2 x_1^2$	1	0		1, 2	0, 0, 0
$\mu + \alpha_i$	1	1	2	1	2
$\mu + \alpha_i + \beta_j + \gamma_{ij}$	1	2	2, 3	1, 1, 2	2, 3, 2, 3
μ_{ij}	0	2	2, 3	2	2, 3
$\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1 x_2$	1	0		1, 1, 2	0, 1, 0, 1
$\mu + \alpha_i + \beta_{x_{1i}} + \beta_i x_{1i}$	1	1	2	1, 1, 2	2, 0, 0, 2

Table 14-3: Regression Functions

Functions for Fitting the Model

“[IMSL_MULTIREGRESS](#)” on page 607 fits a multiple general linear model, where regressors for the general linear model have been generated using “[IMSL_REGRESSORS](#)” on page 600.

Linear Dependence and the R Matrix

Linear dependence of the regressors frequently arises in regression models—sometimes by design and sometimes by accident. The functions in this chapter are designed to handle linear dependence of the regressors; i.e., the $n \times p$ matrix X (the matrix of regressors) in the general linear model can have rank less than p . Often, the models are referred to as nonfull rank models.

As discussed in Searle (1971, Chapter 5), be careful to correctly use the results of the fitted nonfull rank regression model for estimation and hypothesis testing. In the nonfull rank case, not all linear combinations of the regression coefficients can be estimated. Those linear combinations that can be estimated are called “estimable functions.” If the functions are used to attempt to estimate linear combinations that cannot be estimated, error messages are issued. A good general discussion of estimable functions is given by Searle (1971, pp. 180–188).

The check used by functions in this chapter for linear dependence is sequential. The j -th regressor is declared linearly dependent on the preceding $j - 1$ regressors if $1 - R_j^2$ ($1, 2, \dots, j - 1$) is less than or equal to keyword *Tolerance*. Here, R_j ($1, 2, \dots, j - 1$) is the multiple correlation coefficient of the j -th regressor with the first $j - 1$ regressors. When a function declares the j -th regressor to be linearly dependent on the first $j - 1$, the j -th regression coefficient is set to zero. Essentially, this removes the j -th regressor from the model.

The reason a sequential check is used is that practitioners frequently include the preferred variables to remain in the model first. Also, the sequential check is based on many of the computations already performed as this does not degrade the overall efficiency of the functions. There is no perfect test for linear dependence when finite precision arithmetic is used. Keyword *Tolerance* allows you some control over the check for linear dependence. If a model is full rank, input *Tolerance* = 0.0. However, *Tolerance* should be input as approximately 100 times the machine precision. (See `IMSL_MACHINE`.)

Functions performing least squares are based on the QR decomposition of X or on a Cholesky factorization $R^T R$ of $X^T X$. Maindonald (1984, Chapters 1–5) discusses these methods extensively. The R matrix used by the regression function is a $p \times p$ upper-triangular matrix, i.e., all elements below the diagonal are zero. The signs of the diagonal elements of R are used as indicators of linearly dependent regressors and as indicators of parameter restrictions imposed by fitting a restricted model. The rows of R can be partitioned into three classes by the sign of the corresponding diagonal element:

1. A positive diagonal element means the row corresponds to data.

2. A negative diagonal element means the row corresponds to a linearly independent restriction imposed on the regression parameters by $AB = Z$ in a restricted model.
3. A zero diagonal element means a linear dependence of the regressors was declared. The regression coefficients in the corresponding row of:

$$\hat{B}$$

are set to zero. This represents an arbitrary restriction that is imposed to obtain a solution for the regression coefficients. The elements of the corresponding row of R also are set to zero.

Nonlinear Regression Model

The nonlinear regression model is

$$y_i = f(x_i; \theta) + \varepsilon_i \quad i = 1, 2, \dots, n$$

where the observed values of the y_i 's constitute the responses or values of the dependent variable, the x_i 's are the known vectors of values of the independent (explanatory) variables, f is a known function of an unknown regression parameter vector θ , and the ε_i 's are independently distributed normal errors each with mean zero and variance σ^2 .

[“IMSL_NONLINREGRESS”](#) on page 665 performs the least-squares fit to the data for this model.

Weighted Least Squares

Functions throughout this chapter generally allow weights to be assigned to the observations. Keyword *Weights* is used throughout to specify the weighting for each row of X .

Computations that relate to statistical inference—e.g., t tests, F tests, and confidence intervals—are based on the multiple regression model except that the variance of ε_i is assumed to equal σ^2 times the reciprocal of the corresponding weight.

If a single row of the data matrix corresponds to n_i observations, keyword *Frequencies* can be used to specify the frequency for each row of X . Degrees of freedom for error are affected by frequencies but are unaffected by weights.

Summary Statistics

[“IMSL_MULTIREGRESS”](#) on page 607 can be used to compute statistics related to a regression for each of the q dependent variables fitted. The summary statistics include the model analysis of variance table, sequential sum of squares and F -

statistics, coefficient estimates, estimated standard errors, *t*-statistics, variance inflation factors, and estimated variance-covariance matrix of the estimated regression coefficients. “[IMSL_POLYREGRESS](#)” on page 649 includes most of the same functionality for polynomial regressions.

The summary statistics are computed under the model $y = X\beta + \varepsilon$, where y is the $n \times 1$ vector of responses, X is the $n \times p$ matrix of regressors with rank $(X) = r$, β is the $p \times 1$ vector of regression coefficients, and ε is the $n \times 1$ vector of errors whose elements are independently normally distributed with mean zero and variance σ^2/w_i .

Given the results of a weighted least-squares fit of this model (with the w_i 's as the weights), most of the computed summary statistics are output in the following keywords:

Anova_Table—One-dimensional array, usually of length 15. In `IMSL_STEPWISE`, *Anova_Table* is of length 13 because the last two elements of the array cannot be computed from the input. The array contains statistics related to the analysis of variance. The sources of variation examined are the regression, error, and total. The first 10 elements of *Anova_Table* and the notation frequently used for these is described in [Table 14-4](#) (here, *Aov* replaces *Anova_Table*):

Source of Variation	Degrees of Freedom	Sum of Squares	Mean Square	<i>F</i>	<i>p</i> -value
Regression	DFR = <i>Aov</i> (0)	SSR = <i>Aov</i> (3)	MSR = <i>Aov</i> (6)	<i>Aov</i> (8)	<i>Aov</i> (9)
Error	DFE = <i>Aov</i> (1)	SSE = <i>Aov</i> (4)	$s^2 = \text{Aov (7)}$		
Total	DFT = <i>Aov</i> (2)	SST = <i>Aov</i> (5)			

Table 14-4: Model Analysis of Variance

If the model has an intercept (default), the total sum of squares is the sum of squares of the deviations of y_i from its (weighted) mean:

$$\bar{y}$$

the so-called *corrected total sum of squares* denoted by the following:

$$SST = \sum_{i=1}^n w_i (y_i - \bar{y})^2$$

If the model does not have an intercept (*No_Intercept*), the total sum of squares is the sum of squares of y_i —the so-called *uncorrected total sum of squares* denoted by the following:

$$SST = \sum_{i=1}^n w_i y_i^2$$

The error sum of squares is given as follows:

$$SSE = \sum_{i=1}^n w_i (y_i - \hat{y}_i)^2$$

The error degrees of freedom is defined by $DFE = n - r$.

The estimate of σ^2 is given by $s^2 = SSE/DFE$, which is the error mean square.

The computed F statistic for the null hypothesis, $H_0: \beta_1 = \beta_2 = \dots = \beta_k = 0$, versus the alternative that at least one coefficient is nonzero is given by $F = MSR/s^2$. The p -value associated with the test is the probability of an F larger than that computed under the assumption of the model and the null hypothesis. A small p -value (less than 0.05) is customarily used to indicate there is sufficient evidence from the data to reject the null hypothesis.

The remaining five elements in *Anova_Table* frequently are displayed together with the actual analysis of variance table. The quantities R -squared ($R^2 = \text{Anova\_Table}(10)$) and adjusted R -squared ($R_a^2 = \text{Anova\_Table}(11)$) are expressed as a percentage and are defined as follows:

$$R^2 = 100(SSR/SST) = 100(1 - SSE/SST)$$

$$R_a^2 = 100 \left(1 - \frac{s^2}{SST/DFT} \right)$$

The square root of s^2 ($s = \text{Anova\_Table}(12)$) is frequently referred to as the estimated standard deviation of the model error.

The overall mean of the responses:

$$\bar{y}$$

is output in *Anova_Table* (13).

The coefficient of variation ($CV = \text{Anova\_Table}(14)$) is expressed as a percentage and defined by:

$$CV = 100s/\bar{y}$$

T_Tests—Two-dimensional matrix containing the regression coefficient vector $\hat{\beta}$

as one column and associated statistics (estimated standard error, t statistic and p -value) in the remaining columns.

Coef_Covariances—Estimated variance-covariance matrix of the estimated regression coefficients.

Tests for Lack-of-Fit

Tests for lack-of-fit are computed for the polynomial regression by “[IMSL_POLYREGRESS](#)” on page 649. Output keyword *Ssq_Lof* returns the lack-of-fit F tests for each degree polynomial 1, 2, ..., k , that is fit to the data. These tests are used to indicate the degree of the polynomial required to fit the data well.

Diagnostics for Individual Cases

Diagnostics for individual cases (observations) are computed by two functions in the regression chapter: “[IMSL_MULTIPREDICT](#)” on page 622 for linear and nonlinear regressions and “[IMSL_POLYPREDICT](#)” on page 657 for polynomial regressions.

Statistics computed include predicted values, confidence intervals, and diagnostics for detecting outliers and cases that greatly influence the fitted regression.

The diagnostics are computed under the model $y = X\beta + \varepsilon$, where y is the $n \times 1$ vector of responses, X is the $n \times p$ matrix of regressors with rank $(X) = r$, β is the $p \times 1$ vector of regression coefficients, and ε is the $n \times 1$ vector of errors whose elements are independently normally distributed with mean zero and variance σ^2/w_i .

Given the results of a weighted least-squares fit of this model (with the w_i 's as the weights), the following five diagnostics are computed:

1. leverage
2. standardized residual
3. jackknife residual
4. Cook's distance
5. DFFITS

The definitions of these terms are given in the discussion below.

Let x_i be a column vector containing the elements of the i -th row of X . A case can be unusual either because of x_i or because of the response y_i . The *leverage* h_i is a measure of uniqueness of the x_i . The leverage is defined by:

$$h_i = [x_i^T (X^T W X)^{-1} x_i] w_i$$

where $W = \text{diag}(w_1, w_2, \dots, w_n)$ and $(X^T W X)^{-}$ denotes a generalized inverse of $X^T W X$. The average value of the h_i 's is r/n . Regression functions declare x_i unusual if $h_i > 2r/n$. Hoaglin and Welsch (1978) call a data point highly influential (i.e., a leverage point) when this occurs.

Let e_i denote the residual

$$y_i - \hat{y}_i$$

for the i -th case.

The estimated variance of e_i is $(1 - h_i)s^2/w_i$, where s^2 is the estimated standard deviation of the model error. The i -th *standardized residual* (also called the internally studentized residual) is by definition

$$r_i = e_i \sqrt{\frac{w_i}{s^2(1 - h_i)}}$$

and r_i follows an approximate standard normal distribution in large samples.

The i -th *jackknife residual* or *deleted residual* involves the difference between y_i and its predicted value, based on the data set in which the i -th case is deleted. This difference equals $e_i/(1 - h_i)$. The jackknife residual is obtained by standardizing this difference. The residual mean square for the regression in which the i -th case is deleted is as follows:

$$s_i^2 = \frac{(n - r)s^2 - w_i e_i^2 / (1 - h_i)}{n - r - 1}$$

The jackknife residual is defined as

and t_i follows a t distribution with $n - r - 1$ degrees of freedom.

$$t_i = e_i \sqrt{\frac{w_i}{s_i^2(1 - h_i)}}$$

Cook's distance for the i -th case is a measure of how much an individual case affects the estimated regression coefficients. It is given as follows:

$$D_i = \frac{w_i h_i e_i^2}{rs^2(1 - h_i)^2}$$

Weisberg (1985) states that if D_i exceeds the 50-th percentile of the $F(r, n - r)$ distribution, it should be considered large. (This value is about 1. This statistic does not have an F distribution.)

DFFITS, like Cook's distance, is also a measure of influence. For the i -th case, DFFITS is computed by the following formula:

$$\text{DFFITS}_i = e_i \sqrt{\frac{w_i h_i}{s_i^2(1 - h_i)^2}}$$

Hoaglin and Welsch (1978) suggest that DFFITS greater than:

$$2\sqrt{r/n}$$

is large.

Transformations

Transformations of the independent variables are sometimes useful in order to satisfy the regression model. The inclusion of squares and crossproducts of the variables (x_1 , x_2 , x_1^2 , x_2^2 , x_1x_2) often is needed. Logarithms of the independent variables also are used. (See Draper and Smith 1981, pp. 218–222; Box and Tidwell 1962; Atkinson 1985, pp. 177–180; and Cook and Weisberg 1982, pp. 78–86.)

When the responses are described by a nonlinear function of the parameters, a transformation of the model equation often can be selected so that the transformed model is linear in the regression parameters. For example, by taking natural logarithms on both sides of the equation, the exponential model:

$$y = e^{\beta_0 + \beta_1 x_1} \epsilon$$

can be transformed to a model that satisfies the linear regression model provided the ϵ_i 's have a log-normal distribution (Draper and Smith 1981, pp. 222–225).

When the responses are nonnormal and their distribution is known, a transformation of the responses often can be selected so that the transformed responses closely satisfy the regression model assumptions. The square-root transformation for counts with a Poisson distribution and the arc-sine transformation for binomial proportions are common examples (Snedecor and Cochran 1967, pp. 325–330; Draper and Smith 1981, pp. 237–239).

Alternatives to Least Squares

The method of least squares has desirable characteristics when the errors are normally distributed, e.g., a least-squares solution produces maximum likelihood estimates of the regression parameters. However, when errors are not normally distributed, least squares may yield poor estimators. The IMSL_LNORMREGRESS function offers three alternatives to least squares methodology, Least Absolute Value, L_p Norm, and Least Maximum Value.

The least absolute value (LAV, L_1) criterion yields the maximum likelihood estimate when the errors follow a Laplace distribution. Keyword *Lav* (705) is often used when the errors have a heavy tailed distribution or when a fit is needed that is resistant to outliers.

A more general approach, minimizing the L_p norm ($p \leq 1$), is given by keyword *Llp* (704). Although the routine requires about 30 times the CPU time for the case $p = 1$ than would the use of keyword *Lav*, the generality of *Llp* allows you to try several choices for $p \geq 1$ by simply changing the input value of p in the calling program. The CPU time decreases as p gets larger. Generally, choices of p between 1 and 2 are of interest. However, the L_p norm solution for values of p larger than 2 can also be computed.

The minimax (LMV, L_∞ , Chebyshev) criterion is used by setting keyword *Lmv*. Its estimates are very sensitive to outliers, however, the minimax estimators are quite efficient if the errors are uniformly distributed.

Missing Values

NaN (Not a Number) is the missing value code used by the regression functions. Use IMSL_MACHINE to retrieve NaN. Any element of the data matrix that is missing must be set to NaN. In fitting regression models, any observation containing NaN for the independent, dependent, weight, or frequency variables is omitted from the computation of the regression parameters.

Regression Routines

Multiple Linear Regression

[IMSL_REGRESSORS](#)—Generates regressors for a general linear model.

[IMSL_MULTIREGRESS](#)—Fits a multiple linear regression model and optionally produces summary statistics for a regression model.

[IMSL_MULTIPREDICT](#)—Computes predicted values, confidence intervals, and diagnostics.

Variable Selection

[IMSL_ALLBEST](#)—All best regressions.

[IMSL_STEPWISE](#)—Stepwise regression.

Polynomial and Nonlinear Regression

[IMSL_POLYREGRESS](#)—Fits a polynomial regression model.

[IMSL_POLYPREDICT](#)—Computes predicted values, confidence intervals, and diagnostics.

[IMSL_NONLINREGRESS](#)—Fits a nonlinear regression model.

Multivariate Linear Regression—Statistical Inference and Diagnostics

[IMSL_HYPOTH_PARTIAL](#)—Construction of a completely testable hypothesis.

[IMSL_HYPOTH_SCPH](#)—Sums of cross products for a multivariate hypothesis.

[IMSL_HYPOTH_TEST](#)—Tests for the multivariate linear hypothesis.

Polynomial and Nonlinear Regression

[IMSL_NONLINOPT](#)—Fit a nonlinear regression model using Powell's algorithm.

Alternatives to Least Squares Regression

[IMSL_LNORMREGRESS](#)—LAV, Lpnorm, and LMV criteria regression.

IMSL_REGRESSORS

The IMSL_REGRESSORS function generates regressors for a general linear model.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_REGRESSORS(x, n_class, n_continuous
    [, CLASS_COLUMNS=array] [, /DOUBLE] [, DUMMY_METHOD=variable]
    [, INDICES_EFFECTS=array] [, ORDER=value] [, VAR_EFFECTS=array])
```

Return Value

A two-dimensional array containing the regressor variables generated from *x*.

Arguments

x

Two-dimensional array containing the data. The columns must be ordered such that the first *n_class* columns contain the class variables and the next *n_continuous* columns contain the continuous variables. (Exception: See keyword *Class_Columns*.)

n_class

Number of classification variables.

n_continuous

Number of continuous variables.

Keywords

CLASS_COLUMNS

One-dimensional array of length *n_class* containing the column numbers of *x* that are the classification variables. The remaining *n_continuous* variables are assumed to

correspond to the columns of x in the range 0, ..., $n\_class - 1$ that are *not* listed in *Class_Columns*. Default: *Class_Columns* = [0, 1, ..., $n\_class - 1$]

DOUBLE

If present and nonzero, double precision is used.

DUMMY_METHOD

Dummy variable option. Indicator variables are defined for each class variable as described in the *Discussion* section. Dummy variables are then generated from the n indicator variables in one of the following three ways:

- (Default)—The n indicator variables are the dummy variables.
- 1—Dummies are the first $n - 1$ indicator variables.
- 2—The $n - 1$ dummies are defined in terms of the indicator variables so that for balanced data, the usual summation restrictions are imposed on the regression coefficients.

INDICES_EFFECTS

One-dimensional array of length $Var\_Effects(0) + Var\_Effects(1) + \dots + Var\_Effects(N\_ELEMENTS(Var\_Effects) - 1)$. The first $Var\_Effects(0)$ elements give the column numbers of x for each variable in the first effect. The next $Var\_Effects(1)$ elements give the column numbers for each variable in the second effect. The last $Var\_Effects(N\_ELEMENTS(Var\_Effects) - 1)$ elements give the column numbers for each variable in the last effect. Keywords *Var_Effects* and *Indices_Effects* must be used together.

ORDER

Order of the model. Model order can be specified as 1 or 2. Use keyword *Indices_Effects* to specify more complicated models. The keywords *Var_Effects* and *Indices_Effects* must be used together. Default: *Order* = 1

VAR_EFFECTS

One-dimensional array containing the number of variables associated with each effect in the model. The keywords *Var_Effects* and *Indices_Effects* must be used together.

Discussion

The `IMSL_REGRESSORS` function generates regressors for a general linear model from a data matrix. The data matrix can contain classification variables as well as continuous variables. Regressors for effects composed solely of continuous variables are generated as powers and crossproducts. Consider a data matrix containing continuous variables as Columns 3 and 4. The effect indices (3, 3) generate a regressor whose i -th value is the square of the i -th value in Column 3. The effect indices (3, 4) generates a regressor whose i -th value is the product of the i -th value in Column 3 with the i -th value in Column 4.

Regressors for an effect (source of variation) composed of a single classification variable are generated using indicator variables. Let the classification variable A take on values $a_1, a_2, \dots, a_n$. From this classification variable, `IMSL_REGRESSORS` creates n indicator variables. For $k = 1, 2, \dots, n$:

$$I_k = \begin{cases} 1 & \text{if } A = a_k \\ 0 & \text{otherwise} \end{cases}$$

For each classification variable, another set of variables is created from the indicator variables. These new variables are called *dummy variables*. Dummy variables are generated from the indicator variables in one of three manners:

1. The dummies are the n indicator variables. (Default method)
2. The dummies are the first $n - 1$ indicator variables. (*Dummy_Method* = 1)
3. The $n - 1$ dummies are defined in terms of the indicator variables so that for balanced data, the usual summation restrictions are imposed on the regression coefficients. (*Dummy_Method* = 2)

In particular, for the default case, the dummy variables are

$A_k = I_k$ ($k = 1, 2, \dots, n$). For *Dummy_Method* = 1, the dummy variables are $A_k = I_k$ ($k = 1, 2, \dots, n - 1$). For *Dummy_Method* = 2, the dummy variables are $A_k = I_k - I_n$ ($k = 1, 2, \dots, n - 1$). The regressors generated for an effect composed of a single-classification variable are the associated dummy variables.

Let m_j be the number of dummies generated for the j -th classification variable. Suppose there are two classification variables A and B with dummies:

$$A_1, A_2, \dots, A_{m_1} \quad \text{and} \quad B_1, B_2, \dots, B_{m_2}$$

The regressors generated for an effect composed of two classification variables A and B are:

$$\begin{aligned}
A \otimes B &= (A_1, A_2, \dots, A_{m_1}) \otimes (B_1, B_2, \dots, B_{m_2}) \\
&= (A_1 B_1, A_1 B_2, \dots, A_1 B_{m_2}, A_2 B_1, A_2 B_2, \dots, A_2 B_{m_2}, \dots \\
&\quad A_{m_1} B_1, A_{m_1} B_2, \dots, A_{m_1} B_{m_2})
\end{aligned}$$

More generally, the regressors generated for an effect composed of several classification variables and several continuous variables are given by the Kronecker products of variables, where the order of the variables is specified in *Indices_Effects*. Consider a data matrix containing classification variables in Columns 0 and 1 and continuous variables in Columns 2 and 3. Label these four columns *A*, *B*, *X*₁, and *X*₂. The regressors generated by the effect indices (0, 1, 2, 2, 3) are:

$$A \otimes B \otimes X_1 X_1 X_2$$

Remarks

Let the data matrix $x = (A, B, X_1)$, where *A* and *B* are classification variables and *X*₁ is a continuous variable. The model containing the effects *A*, *B*, *AB*, *X*₁, *AX*₁, *BX*₁, and *ABX*₁ is specified as follows (use optional keyword *Indices_Effects*):

$$n\_class = 2$$

$$n\_continuous = 1$$

$$Var\_Effects = [1, 1, 2, 1, 2, 2, 3]$$

$$Indices\_Effects = [0, 1, 0, 1, 2, 0, 2, 1, 2, 0, 1, 2]$$

For this model, suppose that variable *A* has two levels, *A*₁ and *A*₂, and that variable *B* has three levels, *B*₁, *B*₂, and *B*₃. For each *Dummy_Method* option, the regressors in their order of appearance in *IMSL_REGRESSORS* are given below

- (Default)—*A*₁, *A*₂, *B*₁, *B*₂, *B*₃, *A*₁ *B*₁, *A*₁ *B*₂, *A*₁ *B*₃, *A*₂ *B*₁, *A*₂ *B*₂, *A*₂ *B*₃, *X*₁, *A*₁ *X*₁, *A*₂ *X*₁, *B*₁ *X*₁, *B*₂ *X*₁, *B*₃ *X*₁, *A*₁ *B*₁ *X*₁, *A*₁ *B*₂ *X*₁, *A*₁ *B*₃ *X*₁, *A*₂ *B*₁ *X*₁, *A*₂ *B*₂ *X*₁, *A*₂ *B*₃ *X*₁
- 1—*A*₁, *B*₁, *B*₂, *A*₁ *B*₁, *A*₁ *B*₂, *X*₁, *A*₁ *X*₁, *B*₁ *X*₁, *B*₂ *X*₁, —*A*₁ *B*₁ *X*₁, *A*₁ *B*₂ *X*₁
- 2—*A*₁ − *A*₂, *B*₁ − *B*₃, *B*₂ − *B*₃, (*A*₁ − *A*₂) (*B*₁ − *B*₂), (*A*₁ − *A*₂) (*B*₂ − *B*₃), *X*₁, (*A*₁ − *A*₂) *X*₁, (*B*₁ − *B*₃) *X*₁, (*B*₂ − *B*₃) *X*₁, (*A*₁ − *A*₂) (*B*₁ − *B*₂) *X*₁, (*A*₁ − *A*₂) (*B*₂ − *B*₃) *X*₁

Within a group of regressors corresponding to an interaction effect, the indicator variables composing the regressors vary most rapidly for the last classification variable, next most rapidly for the next to last classification variable, etc.

By default, *IMSL_REGRESSORS* internally generates values for *Var_Effects* and *Indices_Effects*, which correspond to a first order model with

$NEF = n\_continuous + n\_class$. The variables then are used to create the regressor variables. The effects are ordered such that the first effect corresponds to the first column of x , the second effect corresponds to the second column of x , etc. A second order model corresponding to the columns (variables) of x is generated if *Order* with *Order* = 2 is specified.

There are:

$$NEF = n\_class + 2 * n\_continuous + \binom{NVAR}{2}$$

effects, where $NVAR = n\_continuous + n\_class$. The first $NVAR$ effects correspond to the columns of x , such that the first effect corresponds to the first column of x , the second effect corresponds to the second column of x , ..., the $NVAR$ -th effect corresponds to the $NVAR$ -th column of x (i.e., $x(NVAR - 1)$). The next $n\_continuous$ effects correspond to squares of the continuous variables. The last:

$$\binom{NVAR}{2}$$

effects correspond to the two-variable interactions.

- Let the data matrix $x = (A, B, X_1)$, where A and B are classification variables and X_1 is a continuous variable. The effects generated and order of appearance is $A, B, X_1, X_1^2, AB, AX_1, BX_1$.
- Let the data matrix $x = (A, X_1, X_2)$, where A is a classification variable and X_1 and X_2 are continuous variables. The effects generated and order of appearance is $A, X_1, X_2, X_1^2, X_2^2, AX_1, AX_2, X_1X_2$.
- Let the data matrix $x = (X_1, A, X_2)$ (see *Class_Columns*), where A is a classification variable and X_1 and X_2 are continuous variables. The effects generated and order of appearance is $X_1, A, X_2, X_1^2, X_2^2, X_1A, X_1X_2, AX_2$.

Higher-order and more complicated models can be specified using *Indices_Effects*.

Examples

Example 1

In the following example, there are two classification variables, A and B , with two and three values, respectively. Regressors for a one-way model (the default model order) are generated using the ALL dummy method (the default dummy method). The five regressors generated are A_1, A_2, B_1, B_2, B_3 .


```

labels = ['A1', 'A2', 'B1', 'B2', 'B3']
; Define some labels for printing later.
RM, x, 6, 2
; Enter the data.
row 0: 10 5
row 1: 20 15
row 2: 20 10
row 3: 10 10
row 4: 10 15
row 5: 20 5
reg = IMSL_REGRESSORS(x, 2, 0)
; Call IMSL_REGRESSORS.
PM, labels, reg, FORMAT = '(5a8, /, 6(5f8.1, /))'
; Print the results.

```

A1	A2	B1	B2	B3
1.0	0.0	1.0	0.0	0.0
0.0	1.0	0.0	0.0	1.0
0.0	1.0	0.0	1.0	0.0
1.0	0.0	0.0	1.0	0.0
1.0	0.0	0.0	0.0	1.0
0.0	1.0	1.0	0.0	0.0

Example 2

In this example, a two-way analysis of covariance model containing all the interaction terms is fit. First, `IMSL_REGRESSORS` is called to produce a matrix of regressors, `reg`, from the data `x`. The regressors, generated using *Dummy_Method* = 1, are the model whose mean function is:

$$\mu + \alpha_i + \beta_j + \gamma_{ij} + \delta x_{ij} + \zeta_i x_{ij} + \eta_j x_{ij} + \theta_{ij} x_{ij} \quad i = 1, 2; j = 1, 2, 3$$

where $\alpha_2 = \beta_3 = \gamma_{21} = \gamma_{22} = \gamma_{23} = \zeta_2 = \eta_3 = \theta_{21} = \theta_{22} = \theta_{23} = 0$.

```

labels = ['Alpha1', 'Beta1', 'Beta2', 'Gamma11', 'Gamma12', $
'Delta', 'Zeta1', 'Eta1', 'Eta2', 'Theta11', 'Theta12']
; Define some labels to use in printing the results.
x = transpose([ [1.0, 1.0, 1.11], [1.0, 1.0, 2.22], $
[1.0, 1.0, 3.33], [1.0, 2.0, 1.11], [1.0, 2.0, 2.22], $
[1.0, 2.0, 3.33], [1.0, 3.0, 1.11], [1.0, 3.0, 2.22], $
[1.0, 3.0, 3.33], [2.0, 1.0, 1.11], [2.0, 1.0, 2.22], $
[2.0, 1.0, 3.33], [2.0, 2.0, 1.11], [2.0, 2.0, 2.22], $
[2.0, 2.0, 3.33], [2.0, 3.0, 1.11], [2.0, 3.0, 2.22], $
[2.0, 3.0, 3.33]])
Var_Effects = [1, 1, 2, 1, 2, 2, 3]
Indices_Effects = [0, 1, 0, 1, 2, 0, 2, 1, 2, 0, 1, 2]
reg = IMSL_REGRESSORS(x, 2, 1, Dummy_Method = 1, $
Var_Effects = var_effects, Indices_Effects = indices_effects)
; Call IMSL_REGRESSORS.
PM, labels(0:5), reg(*, 0:5), FORMAT = '(6a9, /, 18(6f9.2, /))'

```

```
; Output the results.
  Alpha1  Beta1  Beta2 Gamma11 Gamma12 Delta
    1.0    1.0    0.0    1.0    0.0    1.1
    1.00   1.00   0.00   1.00   0.00   2.22
    1.00   1.00   0.00   1.00   0.00   3.33
    1.00   0.00   1.00   0.00   1.00   1.11
    1.00   0.00   1.00   0.00   1.00   2.22
    1.00   0.00   1.00   0.00   1.00   3.33
    1.00   0.00   0.00   0.00   0.00   1.11
    1.00   0.00   0.00   0.00   0.00   2.22
    1.00   0.00   0.00   0.00   0.00   3.33
    0.00   1.00   0.00   0.00   0.00   1.11
    0.00   1.00   0.00   0.00   0.00   2.22
    0.00   0.00   1.00   0.00   0.00   1.11
    0.00   0.00   1.00   0.00   0.00   2.22
    0.00   0.00   1.00   0.00   0.00   3.33
    0.00   0.00   0.00   0.00   0.00   1.11
    0.00   0.00   0.00   0.00   0.00   2.22
    0.00   0.00   0.00   0.00   0.00   3.33
PM, labels(6:10), reg(*, 6:10), FORMAT = '(5a9, /, 18(5f9.2, /))'
  Zeta1  Eta1  Eta2  Theta11  Theta12
    1.1    1.1    0.0    1.1    0.0
    2.22   2.22   0.00   2.22   0.00
    3.33   3.33   0.00   3.33   0.00
    1.11   0.00   1.11   0.00   1.11
    2.22   0.00   2.22   0.00   2.22
    3.33   0.00   3.33   0.00   3.33
    1.11   0.00   0.00   0.00   0.00
    2.22   0.00   0.00   0.00   0.00
    3.33   0.00   0.00   0.00   0.00
    0.00   1.11   0.00   0.00   0.00
    0.00   2.22   0.00   0.00   0.00
    0.00   3.33   0.00   0.00   0.00
    0.00   0.00   1.11   0.00   0.00
    0.00   0.00   2.22   0.00   0.00
    0.00   0.00   3.33   0.00   0.00
    0.00   0.00   0.00   0.00   0.00
    0.00   0.00   0.00   0.00   0.00
    0.00   0.00   0.00   0.00   0.00
```

Version History

6.4	Introduced

IMSL_MULTIREGRESS

The IMSL_MULTIREGRESS function fits a multiple linear regression model using least squares and optionally compute summary statistics for the regression model.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_MULTIREGRESS(x, y [, ANOVA_TABLE=variable]
    [, COEF_COVARIANCES=variable] [, COEF_VIF=variable] [, /DOUBLE]
    [, FREQUENCIES=array] [, /NO_INTERCEPT] [, PREDICT_INFO=variable]
    [, RANK=variable] [, RESIDUAL=variable] [, T_TESTS=variable]
    [, TOLERANCE=value] [, WEIGHTS=array] [, XMEAN=variable])
```

Return Value

If keyword *No_Intercept* is not used, IMSL_MULTIREGRESS is an array of length N_ELEMENTS($x(*, 0)$) containing a least-squares solution for the regression coefficients. The estimated intercept is the initial component of the array.

Arguments

x

Two-dimensional matrix containing the independent (explanatory) variables. The data value for the i -th observation of the j -th independent (explanatory) variable should be in element $x(i, j)$.

y

Two-dimensional matrix containing of size N_ELEMENTS($x(*, 0)$) by $n_{dependent}$ containing the dependent (response) variable(s). The i -th column of y contains the i -th dependent variable.

Keywords

ANOVA_TABLE

Named variable into which the array containing the analysis of variance table is stored. Each column of *Anova_table* corresponds to a dependent variable. The analysis of variance statistics are shown in [Table 14-5](#):

Element	Analysis of Variance Statistic
0	degrees of freedom for the model
1	degrees of freedom for error
2	total (corrected) degrees of freedom
3	sum of squares for the model
4	sum of squares for error
5	total (corrected) sum of squares
6	model mean square
7	error mean square
8	overall <i>F</i> -statistic
9	<i>p</i> -value
10	R^2 (in percent)
11	adjusted R^2 (in percent)
12	estimate of the standard deviation
13	overall mean of <i>y</i>
14	coefficient of variation (in percent)

Table 14-5: Analysis of Variance Statistics

COEF_COVARIANCES

Named variable into which the $m \times m \times n\_dependent$ array containing estimated variances and covariances of the estimated regression coefficients is stored. Here, m is number of regression coefficients in the model. If *No_Intercept* is specified, $m = N\_ELEMENTS(x(0, *))$; otherwise, $m = (N\_ELEMENTS(x(0, *)) + 1)$.

COEF_VIF

Named variable into which a one-dimensional array of length NPAR containing the variance inflation factor, where NPAR is the number of parameters, is stored. The $(i + \text{INTCEP})$ -th element corresponds to the i -th independent variable, where $i = 0, 1, 2, \dots, \text{NPAR} - 1$, and INTCEP is equal to 1 if an intercept is in the model and 0 otherwise. The square of the multiple correlation coefficient for the i -th regressor after all others is obtained from *Coef_Vif* by the following formula:

$$1.0 - \frac{1.0}{\text{Coef\_Vif}(i)}$$

If there is no intercept or there is an intercept and $i = 0$, the multiple correlation coefficient is not adjusted for the mean.

DOUBLE

If present and nonzero, double precision is used.

FREQUENCIES

One-dimensional array containing the frequency for each observation. Default: *Frequencies*(*) = 1

NO_INTERCEPT

If present and nonzero, the intercept term:

$$\hat{\beta}_0$$

is omitted from the model. By default, the fitted value for observation i is:

$$\hat{\beta}_0 + \hat{\beta}_1 x_1 + \dots + \hat{\beta}_k x_k$$

where k is the number of independent variables.

PREDICT_INFO

Named variable into which the one-dimensional byte array containing information needed by IMSL_MULTIPREDICT is stored. The data contained in this array is in an encrypted format and should not be altered before it is used in subsequent calls to IMSL_MULTIPREDICT.

RANK

Named variable into which the rank of the fitted model is stored.

RESIDUAL

Variable into which the array containing the residuals is stored.

T_TESTS

Named variable into which the NPAR (where NPAR is equal to the number of parameters in the model) by 4 array containing statistics relating to the regression coefficients is stored.

Each row corresponds to a coefficient in the model, where NPAR is the number of parameters in the model. Row $i + \text{INTCEP}$ corresponds to the i -th independent variable, where INTCEP is equal to 1 if an intercept is in the model and 0 otherwise, and $i = 0, 1, 2, \dots, \text{NPAR} - 1$. The statistics in the columns are as follows:

- 0—coefficient estimate
- 1—estimated standard error of the coefficient estimate
- 2— t -statistic for the test that the coefficient is 0
- 3— p -value for the two-sided t test

TOLERANCE

Tolerance used in determining linear dependence. For MULTIGRESS, *Tolerance* = $100 \times \epsilon$, where ϵ is machine precision (default).

WEIGHTS

One-dimensional array containing the weight for each observation. Default: *Weights*(*) = 1

XMEAN

Named variable into which the array containing the estimated means of the independent variables is stored.

Discussion

The IMSL_MULTIREGRESS function fits a multiple linear regression model with or without an intercept.

By default, the multiple linear regression model is

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_k x_{ik} + \epsilon_i \quad i = 0, 2, \dots, n$$

where the observed values of the y_i 's (input in y) are the responses or values of the dependent variable; the x_{i1} 's, x_{i2} 's, ..., x_{ik} 's (input in x) are the settings of the k independent variables; $\beta_0, \beta_1, \dots, \beta_k$ are the regression coefficients whose estimated values are to be output by `IMSL_MULTIREGRESS`; and the ϵ_i 's are independently distributed normal errors, each with mean zero and variance σ^2 . Here, $n = (\text{N\_ELEMENTS}(x(*, 0)))$. Note that by default, β_0 is included in the model.

The `IMSL_MULTIREGRESS` function computes estimates of the regression coefficients by minimizing the weighted sum of squares of the deviations of the observed response y_i from the fitted response:

$$\hat{y}_i$$

for the n observations. This weighted minimum sum of squares (the error sum of squares) is output as one of the analysis of variance statistics if `Anova_Table` is specified and is computed as shown below:

$$\text{SSE} = \sum_{i=1}^n w_i (y_i - \hat{y}_i)^2$$

Another analysis of variance statistics is the total sum of squares. By default, the weighted total sum of squares is the weighted sum of squares of the deviations of y_i from its mean:

$$\bar{y}$$

the so-called *corrected total sum of squares*. This statistic is computed as follows:

$$\text{SST} = \sum_{i=1}^n w_i (y_i - \bar{y})^2$$

When `No_Intercept` is specified, the total weighted sum of squares is the sum of squares of y_i , the so called *uncorrected total weighted sum of squares*. This is computed as follows:

$$\text{SST} = \sum_{i=1}^n w_i y_i^2$$

See Draper and Smith (1981) for a good general treatment of the multiple linear regression model, its analysis, and many examples.

In order to compute a least-squares solution, `IMSL_MULTIREGRESS` performs an orthogonal reduction of the matrix of regressors to upper-triangular form. The reduction is based on one pass through the rows of the augmented matrix (x, y) using fast Givens transformations (Golub and Van Loan 1983, pp. 156–162; Gentleman

1974). This method has the advantage that it avoids the loss of accuracy that results from forming the crossproduct matrix used in the normal equations.

By default, the current means of the dependent and independent variables are used to internally center the data for improved accuracy. Let x_j be a column vector containing the j -th row of data for the independent variables. Let:

$$\bar{x}_i$$

represent the mean vector for the independent variables given the data for rows 0, 1, ..., i . The current mean vector is defined to be:

$$\bar{x}_i = \frac{\sum_{j=1}^i w_j f_j x_j}{w_j f_j}$$

where the w_j 's and the f_j 's are the weights and frequencies. The i -th row of data has:

$$\bar{x}_i$$

subtracted from it and is multiplied by:

$$w_i f_i \frac{a_i}{a_{i-1}}$$

Although a crossproduct matrix is not computed, the validity of this centering operation can be seen from the formula below for the sum-of-squares and crossproducts matrix:

$$\sum_{i=1}^n w_i f_i (x_i - \bar{x}_n)(x_i - \bar{x}_n)^T = \sum_{i=2}^n \frac{a_i}{a_{i-1}} w_i f_i (x_i - \bar{x}_i)(x_i - \bar{x}_i)^T$$

An orthogonal reduction on the centered matrix is computed. When the final computations are performed, the intercept estimate and the first row and column of the estimated covariance matrix of the estimated coefficients are updated (if *Coef_Covariances* is specified) to reflect the statistics for the original (uncentered) data. This means that the estimate of the intercept is for the uncentered data.

As part of the final computations, MULTIGRESS checks for linearly dependent regressors. In particular, linear dependence of the regressors is declared if any of the following three conditions is satisfied:

- A regressor equals zero.
- Two or more regressors are constant.
- The expression:

$$\sqrt{1 - R_{i \cdot 1, 2, \dots, i-1}^2}$$

is less than or equal to *Tolerance*. Here, $R_{i \cdot 1, 2, \dots, i-1}$ is the multiple correlation coefficient of the i -th independent variable with the first $i - 1$ independent variables. If no intercept is in the model, the “multiple correlation” coefficient is computed without adjusting for the mean.

On completion of the final computations, if the i -th regressor is declared to be linearly dependent upon the previous $i - 1$ regressors, then the i -th coefficient estimate and all elements in the i -th row and i -th column of the estimated variance-covariance matrix of the estimated coefficients (if *Coef_Covariances* is specified) are set to zero. Finally, if a linear dependence is declared, an informational (error) message, code `STAT_RANK_DEFICIENT`, is issued indicating the model is not full rank.

The `IMSL_MULTIREGRESS` function also can be used to compute summary statistics from a fitted general linear model. The model is $y = X\beta + \epsilon$, where y is the $n \times 1$ vector of responses, X is the $n \times p$ matrix of regressors, β is the $p \times 1$ vector of regression coefficients, and ϵ is the $n \times 1$ vector of errors whose elements are each independently distributed with mean zero and variance σ^2 . The `IMSL_MULTIREGRESS` function uses the results of this fit to compute summary statistics, including analysis of variance, sequential sum of squares, t tests, and an estimated variance-covariance matrix of the estimated regression coefficients.

Some generalizations of the general linear model are allowed. If the i -th element of ϵ has variance of:

$$\frac{\sigma^2}{w_i}$$

and the weights w_i are used in the fit of the model, `IMSL_MULTIREGRESS` produces summary statistics from the weighted least-squares fit. More generally, if the variance-covariance matrix of ϵ is $\sigma^2 V$, `IMSL_MULTIREGRESS` can be used to produce summary statistics from the generalized least-squares fit. The `IMSL_MULTIREGRESS` function can be used to perform a generalized least-squares fit by regressing y^* on X^* where $y^* = (T^{-1})^T y$, $X^* = (T^{-1})^T X$ and T satisfies $T^T T = V$.

The sequential sum of squares for the i -th regression parameter is given by:

$$(\hat{R}\hat{\beta})_0^2$$

The regression sum of squares is given by the sum of the sequential sum of squares. If an intercept is in the model, the regression sum of squares is adjusted for the mean, i.e.:

$$(\hat{R}\hat{\beta})_0^2$$

is not included in the sum.

The estimate of σ^2 is s^2 (stored in *Anova_Table*(7) that is computed as SSE/DFE.

If R is nonsingular, the estimated variance-covariance matrix of:

$$\hat{\beta}$$

(stored in *Coef_Covariances*) is computed by $s^2 R^{-1} (R^{-1})^T$.

If R is singular, corresponding to $\text{rank}(X) < p$, a generalized inverse is used. For a matrix G to be a g_i ($i = 1, 2, 3$, or 4) inverse of a matrix A , G must satisfy conditions j (for $j \leq i$) for the Moore-Penrose inverse but generally must fail conditions k (for $k > i$). The four conditions for G to be a Moore-Penrose inverse of A are as follows:

1. $AGA = A$
2. $GAG = G$
3. AG is symmetric
4. GA is symmetric

In the case where R is singular, the method for obtaining *Coef_Covariances* follows the discussion of Maindonald (1984, pp. 101–103). Let Z be the diagonal matrix with diagonal elements defined by the following:

$$z_{ii} = \begin{cases} 1 & \text{if } r_{ii} \neq 0 \\ 0 & \text{if } r_{ii} = 0 \end{cases}$$

Let G be the solution to $RG = Z$ obtained by setting the i -th ($\{i: r_{ii} = 0\}$) row of G to zero. Keyword *Coef_Covariances* is set to $s^2 GG^T$. (G is a g_3 inverse of R , represented by:

$$R^{g_3}$$

$$\text{the result } R^{g_3} R^{g_3 T}$$

is a symmetric g_2 inverse of $R^T R = X^T X$. See Sallas and Lioni 1988.)

Note that keyword *Coef_Covariances* can be used only to get variances and covariances of estimable functions of the regression coefficients, i.e., nonestimable functions (linear combinations of the regression coefficients not in the space spanned by the nonzero rows of R) must not be used. See, for example, Maindonald (1984, pp. 166–168) for a discussion of estimable functions.

The estimated standard errors of the estimated regression coefficients (stored in Column 1 of *T_Tests*) are computed as square roots of the corresponding diagonal entries in *Coef_Covariances*.

For the case where an intercept is in the model, set:

$$\bar{R}$$

equal to the matrix *R* with the first row and column deleted. Generally, the variance inflation factor (VIF) for the *i*-th regression coefficient is computed as the product of the *i*-th diagonal element of $R^T R$ and the *i*-th diagonal element of its computed inverse. If an intercept is in the model, the VIF for those coefficients not corresponding to the intercept uses the diagonal elements of:

$$\bar{R}^T \bar{R}$$

(see Maindonald 1984, p. 40).

Remarks

When *R* is nonsingular and comes from an unrestricted regression fit, *Coef_Covariances* is the estimated variance-covariance matrix of the estimated regression coefficients and $\text{Coef\_Covariances} = (\text{SSE/DFE}) (R^T R)^{-1}$.

Otherwise, variances and covariances of estimable functions of the regression coefficients can be obtained using *Coef_Covariances* and $\text{Coef\_Covariances} = (\text{SSE/DFE}) (GDG^T)$. Here, *D* is the diagonal matrix with diagonal elements equal to zero if the corresponding rows of *R* are restrictions and with diagonal elements equal to 1 otherwise. Also, *G* is a particular generalized inverse of *R*.

Examples

Example 1

A regression model:

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_3 x_{i3} + \varepsilon_i \quad i = 1, 2, \dots, 9$$

is fitted to data taken from Maindonald (1984, pp. 203–204).

```
RM, x, 9, 3
; Set up the data.
row 0:  7   5   6
row 1:  2  -1   6
row 2:  7   3   5
row 3: -3   1   4
row 4:  2  -1   0
row 5:  2   1   7
```

```

row 6: -3  -1   3
row 7:  2   1   1
row 8:  2   1   4
y = [7, -5, 6, 5, 5, -2, 0, 8, 3]
; Call IMSL_MULTIREGRESS to compute the coefficients.
coefs = IMSL_MULTIREGRESS(x, y)
; Output the results.
PM, coefs, TITLE = 'Least-Squares Coefficients', $
  FORMAT = '(f10.5)'

```

```

Least-Squares Coefficients
  7.73333
 -0.20000
  2.33333
 -1.66667

```

Example 2: Weighted Least-squares Fit

A weighted least-squares fit is computed using the model

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \varepsilon_i \quad i = 1, 2, \dots, 4$$

and weights $1/i^2$ discussed by Maindonald (1984, pp. 67–68).

In the example, *Weights* is specified. The minimum sum of squares for error in terms of the original untransformed regressors and responses for this weighted regression is:

$$\text{SSE} = \sum_{i=1}^4 w_i (y_i - \hat{y}_i)^2$$

where $w_i = 1/i^2$, represented in the C code as array *w*.

First, a procedure is defined to output the results, including the analysis of variance statistics.

```

PRO print_results, Coefs, Anova_Table
coef_labels = ['intercept', 'linear', 'quadratic']
PM, coef_labels, coefs, TITLE = $
  'Least-Squares Polynomial Coefficients', $
  FORMAT = '(3a20, /, 3f20.4, // )'
anova_labels = ['degrees of freedom for regression', $
  'degrees of freedom for error', $
  'total (corrected) degrees of freedom', $
  'sum of squares for regression', $
  'sum of squares for error', $
  'total (corrected) sum of squares', $
  'regression mean square', $
  'error mean square', 'F-statistic', $

```

```

    'p-value', 'R-squared (in percent)', $
    'adjusted R-squared (in percent)', $
    'est. standard deviation of model error', $
    'overall mean of y', $
    'coefficient of variation (in percent)']
PM, '* * * Analysis of Variance * * * ', FORMAT = '(a50, /)'
FOR i = 0, 14 DO PM, anova_labels(i), $
    anova_table(i), FORMAT = '(a40, f20.2)'
END

RM, x, 4, 2
; Input the values for x.
row 0: -2 0
row 1: -1 2
row 2:  2 5
row 3:  7 3

y = [-3.0, 1.0, 2.0, 6.0]
; Define the dependent variables.
weights = FLTARR(4)
FOR i = 0, 3 DO weights(i) = 1/((i + 1.0)^2)
; Define the weights and print them.
PM, weights
1.00000
0.250000
0.111111
0.0625000
coefs = IMSL_MULTIREGRESS(x, y, WEIGHTS = weights, $
    ANOVA_TABLE = anova_table)
print_results, coefs, anova_table
; Print results using the procedure defined above.
Least-Squares Polynomial Coefficients
    intercept          linear          quadratic
    -1.4307            0.6581            0.7485
* * * Analysis of Variance * * *
degrees of freedom for regression      2.00
degrees of freedom for error           1.00
total (corrected) degrees of freedom   3.00
sum of squares for regression          7.68
sum of squares for error               1.01
total (corrected) sum of squares       8.69
regression mean square                 3.84
error mean square                     1.01
F-statistic                           3.79
p-value                               0.34
R-squared (in percent)                 88.34
adjusted R-squared (in percent)        65.03
est. standard deviation of model error  1.01
overall mean of y                     -1.51

```

coefficient of variation (in percent) -66.55

Example 3: Plotting Results

This example uses `IMSL_MULTIREGRESS` to fit data with both simple linear regression and second order regression. The results, shown in [Figure 14-1](#), are plotted along with confidence bands and residual plots.

```

PRO IMSL_MULTIREGRESS_ex
  !P.MULTI = [0, 2, 2]
  x = [1.0, 1.0, 2.0, 2.0, 3.0, 3.0, 4.0, 4.0, 5.0, 5.0]

  y = [1.1, 0.1, -1.2, 0.3, 1.4, 2.6, 3.1, 4.2, 9.3, 9.6]
  z = FINDGEN(120)/20
  line = MAKE_ARRAY(120, VALUE = 0.0)
  ; Perform a simple linear regression.
  Coefs = IMSL_MULTIREGRESS(x, y, PREDICT_INFO = predict_info)
  y_hat = IMSL_MULTIPREDICT(predict_info, x, $
    RESIDUAL = residual, Y = y)
  y_hat = IMSL_MULTIPREDICT(predict_info, z, $
    CI_PTW_NEW_SAMP = ci)
  PLOT, x, y, Title = 'Simple linear regression', PSYM = 4, $
    XRange = [0.0, 6.0]
  ; Plot the regression.
  y2 = coefs(0) + coefs(1) * z
  OPLOT, z, y2
  OPLOT, z, ci(0, *), LINESTYLE = 1
  OPLOT, z, ci(1, *), LINESTYLE = 1
  PLOT, x, residual, PSYM = 4, TITLE = $
    'Residual plot for simple linear regression', $
    XRange = [0.0, 6.0], YRange = [-6, 6]
  ; Plot the residual.
  OPLOT, z, line
  x2 = [[x], [x * x]]
  ; Compute the second-order regression.
  coefs = IMSL_MULTIREGRESS(x2, y, PREDICT_INFO = predict_info)
  y_hat = IMSL_MULTIPREDICT(predict_info, x2, $
    RESIDUAL = residual, Y = y)
  y_hat = IMSL_MULTIPREDICT(predict_info, $
    [[z], [z * z]], CI_PTW_NEW_SAMP = ci)
  PLOT, x, y, Title = '2nd order regression', $
    PSYM = 4, XRange = [0.0, 6.0]
  ; Plot the second-order regression and the residual.
  y2 = coefs(0) + coefs(1) * z + coefs(2) * z * z
  OPLOT, z, y2
  OPLOT, z, ci(0, *), LINESTYLE = 1
  OPLOT, z, ci(1, *), LINESTYLE = 1
  PLOT, x2, residual, PSYM = 4, TITLE = $
    'Residual plot for 2nd order regression', $

```

```

      X RANGE = [0.0, 6.0], Y RANGE = [-6, 6]
      O PLOT, z, line
END

```

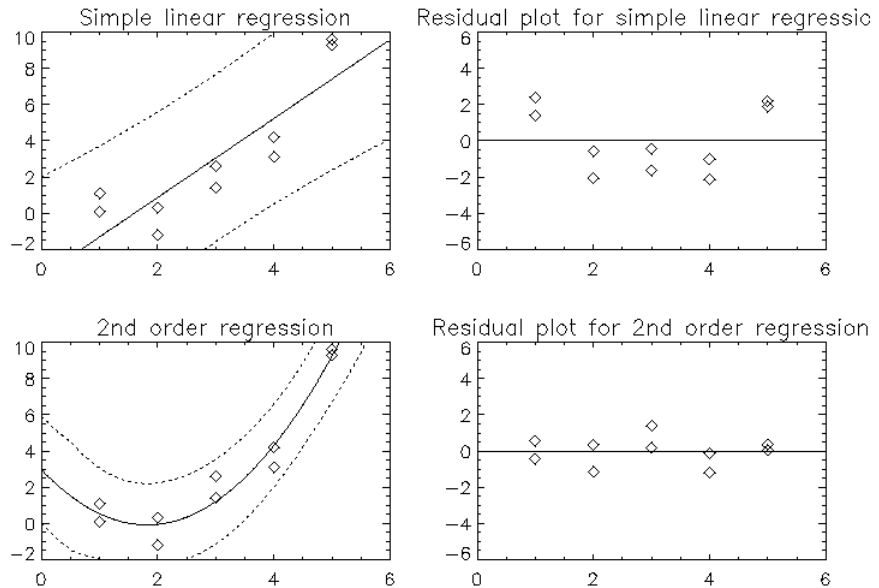


Figure 14-1: Plots of Fit

Example 4: Two-variable, Second-degree Fit

In this example, `IMSL_MULTIREGRESS` is used to compute a two variable second-degree fit to data. The results are shown in [Figure 14-2](#).

```

PRO IMSL_MULTIREGRESS_ex
; Define the data.
x1 = FLTARR(10, 5)
x1(*, 0) = [8.5, 8.9, 10.6, 10.2, 9.8, $
           10.8, 11.6, 12.0, 12.5, 10.9]
x1(*, 1) = [2, 3, 3, 20, 22, 20, 31, 32, 31, 28]
x1(*, 2) = x1(*, 0) * x1(*, 1)
x1(*, 3) = x1(*, 0) * x1(*, 0)
x1(*, 4) = x1(*, 1) * x1(*, 1)
y = [30.9, 32.7, 36.7, 41.9, 40.9, 42.9, 46.3, 47.6, 47.2, 44.0]
nxgrid = 30
nygrid = 30

```

```

; Setup vectors for surface plot. These will be (nxgrid x nygrid)
; elements each, evenly spaced over the range of the data
; in x1(*, 0) and x1(*, 1).
ax1 = min(x1(*, 0)) + (max(x1(*, 0)) - $
    min(x1(*, 0))) * FINDGEN(nxgrid)/(nxgrid - 1)
ax2 = MIN(x1(*, 1)) + (MAX(x1(*, 1)) - $
    MIN(x1(*, 1))) * FINDGEN(nxgrid)/(nxgrid - 1)
coefs = IMSL_MULTIREGRESS(x1, y, RESIDUAL = resid)
; Compute regression coefficients.
z = FLTARR(nxgrid, nygrid)
; Create two-dimensional array of evaluations of the regression
; model at points in grid established by ax1 and ax2.
FOR i = 0, nxgrid - 1 DO BEGIN
    FOR j = 0, nygrid-1 DO BEGIN
        z(i,j) = Coefs(0) $
            + Coefs(1) * ax1(i) + Coefs(2) * ax2(j) $
            + Coefs(3) * ax1(i) * ax2(j) $
            + Coefs(4) * ax1(i)^2 $
            + Coefs(5) * ax2(j)^2
    ENDFOR
ENDFOR
!P.CHARSIZE = 2
SURFACE, z, ax1, ax2, /SAVE, XTITLE = 'X1', YTITLE = 'X2'
PLOTS, x1(*, 0), x1(*, 1), y, /T3D, PSYM = 4, SYMSIZE = 3
XYOUTS, .3, .9, /NORMAL, 'Two-Variable Second-Degree Fit'
; Plot the results.
END

```

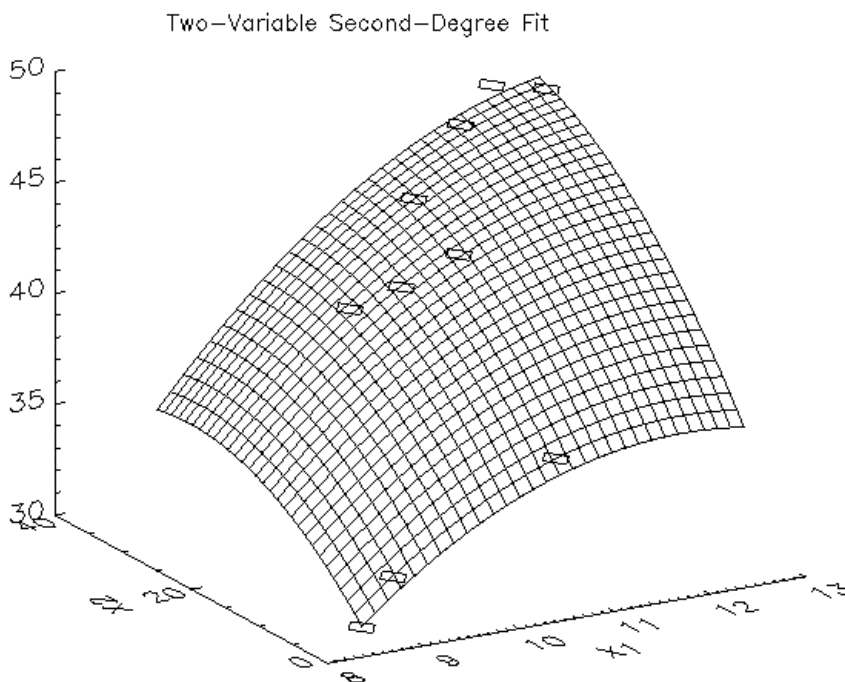



Figure 14-2: Two-variable, Second Degree Fit

Errors

Warning Errors

`STAT_RANK_DEFICIENT`—Model is not full rank. There is not a unique least-squares solution.

Version History

6.4	Introduced

IMSL_MULTIPREDICT

The IMSL_MULTIPREDICT function computes predicted values, confidence intervals, and diagnostics after fitting a regression model.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_MULTIPREDICT(predict_info, x [, CI_SCHEFFE=variable]
    [, CI_PTW_POP_MEAN=variable] [, CI_PTW_NEW_SAMP=variable]
    [, CONFIDENCE=value] [, COOKS_D=variable] [, DEL_RESIDUAL=variable]
    [, DFFITS=variable] [, /DOUBLE] [, LEVERAGE=variable]
    [, RESIDUAL=variable] [, STD_RESIDUAL=variable] [, WEIGHTS=array]
    [, Y=array])
```

Return Value

One-dimensional array of length N_ELEMENTS (x(*, 0)) containing the predicted values.

Arguments

predict_info

One-dimensional byte array containing information computed by IMSL_MULTIREGRESS and returned through keyword *predict_info*. The data contained in this array is in an encrypted format and should not be altered after it is returned by IMSL_MULTIREGRESS.

x

Two-dimensional array containing the combinations of independent variables in each row for which calculations are to be performed.

Keywords

CI_SCHEFFE

Named variable into which the two-dimensional array of size 2 by N_ELEMENTS ($x(*, 0)$) containing the Scheffé confidence intervals corresponding to the rows of x is stored. Element *Ci_Scheffe* (0, i) contains the i -th lower confidence limit; *Ci_Scheffe* (1, i) contains the i -th upper confidence limit.

CI_PTW_POP_MEAN

Named variable into which the two-dimensional array of size 2 by N_ELEMENTS ($x(*, 0)$) containing the confidence intervals for two-sided interval estimates of the means, corresponding to the rows of x , is stored. Element *Ci_Ptw_Pop_Mean* (0, i) contains the i -th lower confidence limit; *Ci_Ptw_Pop_Mean* (1, i) contains the i -th upper confidence limit.

CI_PTW_NEW_SAMP

Named variable into which the two-dimensional array of size 2 by N_ELEMENTS ($x(*, 0)$) containing the confidence intervals for two-sided prediction intervals, corresponding to the rows of x , is stored. Element *Ci_Ptw_New_Samp* (0, i) contains the i -th lower confidence limit; *Ci_Ptw_New_Samp* (1, i) contains the i -th upper confidence limit.

CONFIDENCE

Confidence level for both two-sided interval estimates on the mean and for two-sided prediction intervals, in percent. Keyword *Confidence* must be in the range [0.0, 100.0). For one-sided intervals with confidence level, where $50.0 \leq c < 100.0$, set *Confidence* = $100.0 - 2.0 * (100.0 - c)$. Default: *Confidence* = 95.0

COOKS_D

Named variable into which the one-dimensional array of length N_ELEMENTS ($x(*, 0)$) containing the Cook's D statistics is stored.

Note

You must specify the Y keyword when using this keyword.

DEL_RESIDUAL

Named variable into which the one-dimensional array of length N_ELEMENTS ($x(*, 0)$) containing the deleted residuals is stored.

Note _____

You must specify the *Y* keyword when using this keyword.

DFFITS

Named variable into which the one-dimensional array of length N_ELEMENTS ($x(*, 0)$) containing the DFFITS statistics is stored.

Note _____

You must specify the *Y* keyword when using this keyword.

DOUBLE

If present and nonzero, double precision is used.

LEVERAGE

Named variable into which the one-dimensional array of length N_ELEMENTS ($x(*, 0)$) containing the leverages is stored.

RESIDUAL

Named variable into which the one-dimensional array of length N_ELEMENTS ($x(*, 0)$) containing the residuals is stored.

Note _____

You must specify the *Y* keyword when using this keyword.

STD_RESIDUAL

Named variable into which the one-dimensional array of length N_ELEMENTS ($x(*, 0)$) containing the standardized residuals is stored.

Note _____

You must specify the *Y* keyword when using this keyword.

WEIGHTS

One-dimensional array containing the weight for each row of x . The computed prediction interval uses $SSE/(DFE * \text{Weights}(1))$ for the estimated variance of a future response. Default: $\text{Weights} (*) = 1$

Y

Array of length $N\_ELEMENTS(x(*, 0))$ containing observed responses.

Discussion

The general linear model used by `IMSL_MULTIPREDICT` is:

$$y = X\beta + \varepsilon$$

where y is the $n \times 1$ vector of responses, X is the $n \times p$ matrix of regressors, β is the $p \times 1$ vector of regression coefficients, and ε is the $n \times 1$ vector of errors whose elements are independently normally distributed with mean zero and the following variance:

$$\sigma^2/w_i$$

From a general linear model fit using the w_i 's as the weights, `IMSL_MULTIPREDICT` computes confidence intervals and statistics for the individual cases that constitute the data set. Let x_i be a column vector containing elements of the i -th row of X . Let $W = \text{diag}(w_1, w_2, \dots, w_n)$. The leverage is defined as $h_i = (x_i^T (X^T W X)^{-1} x_i) w_i$. Put $D = \text{diag}(d_1, d_2, \dots, d_p)$ with $d_j = 1$ if the j -th diagonal element of R is positive and zero otherwise. The leverage is computed as $h_i = (a^T D a) w_i$, where a is a solution to $R^T a = x_i$. The estimated variance of:

$$\hat{y} = x_i^T \hat{B}$$

is given by the following:

$$h_i s^2 / w_i, \text{ where } s^2 = SSE/DFE$$

The computation of the remainder of the case statistics follow easily from their definitions. See the chapter introduction for definitions of the case diagnostics.

Informational errors can occur if the input matrix X is not consistent with the information from the fit (contained in `predict_info`), or if excess rounding has occurred. The warning error `STAT_NONESTIMABLE` arises when X contains a row not in the space spanned by the rows of R . An examination of the model that was fitted and the X for which diagnostics are to be computed is required in order to ensure that only linear combinations of the regression coefficients that can be estimated from the

fitted model are specified in x . For further details, see the discussion of estimable functions given in Maindonald (1984, pp. 166–168) and Searle (1971, pp. 180–188).

Often predicted values and confidence intervals are desired for combinations of settings of the independent variables not used in computing the regression fit. This can be accomplished by defining a new data matrix. Since the information about the model fit is input in *predict_info*, it is not necessary to send in the data set used for the original calculation of the fit, i.e., only variable combinations for which predictions are desired need be entered in x .

Examples

Example 1

This example calls IMSL_MULTIPREDICT to compute predicted values after calling IMSL_MULTIREGRESS.

```
x = MAKE_ARRAY(13, 4)
; Define the data set.
x(0, *) = [7, 26, 6, 60]
x(1, *) = [1, 29, 15, 52]
x(2, *) = [11, 56, 8, 20]
x(3, *) = [11, 31, 8, 47]
x(4, *) = [7, 52, 6, 33]
x(5, *) = [11, 55, 9, 22]
x(6, *) = [3, 71, 17, 6]
x(7, *) = [1, 31, 22, 44]
x(8, *) = [2, 54, 18, 22]
x(9, *) = [21, 47, 4, 26]
x(10, *) = [1, 40, 23, 34]
x(11, *) = [11, 66, 9, 12]
x(12, *) = [10, 68, 8, 12]
y = [78.5, 74.3, 104.3, 87.6, 95.9, 109.2, $
     102.7, 72.5, 93.1, 115.9, 83.8, 113.3, 109.4]
coefs = IMSL_MULTIREGRESS(x, y, Predict_Info = predict_info)
; Call IMSL_MULTIREGRESS to compute the fit.
predicted = IMSL_MULTIPREDICT(predict_info, x)
; Call IMSL_MULTIPREDICT to compute predicted values.
PM, predicted, Title = 'Predicted values'
; Output the predicted values.
Predicted values
78.4952
72.7888
105.971
89.3271
95.6492
105.275
```

```

104.149
75.6750
91.7216
115.618
81.8090
112.327
111.694

```

Example 2

This example uses the same data set as the first example and also uses a number of keywords to retrieve additional information from `IMSL_MULTIPREDICT`. First, a procedure is defined to print the results.

```

PRO print_results, anova_table, t_tests, y, $
predicted, ci_scheffe, residual, dffits
labels = ['df for among groups', $
'df for within groups', $
'total (corrected) df', $
'ss for among groups', $
'ss for within groups', $
'total (corrected) ss', $
'mean square among groups', $
'mean square within groups', $
'F-statistic', $
'P-value', $
'R-squared (in percent)', $
'adjusted R-squared (in percent)', $
'est. std of within group error', $
'overall mean of y', $
'coef. of variation (in percent)']
PRINT, ' * * Analysis of Variance * *'
; Print the analysis of variance table.
PM, [[labels], [STRING(anova_table, FORMAT = '(f11.4)')]]
PRINT
PRINT, 'Coefficient s.e.      t      p-value'
PM, t_tests, FORMAT = '(f7.2, 4x, 3f7.2)'
PRINT
PRINT, ' observed predicted  lower upper residual dffits'
PM, [[y], [predicted], [transpose(ci_scheffe)], $
[residual], [dffits]], FORMAT = '(6f10.2)'
END

x = MAKE_ARRAY(13, 4)
; Define the data set.
x(0, *) = [7, 26, 6, 60]
x(1, *) = [1, 29, 15, 52]
x(2, *) = [11, 56, 8, 20]
x(3, *) = [11, 31, 8, 47]
x(4, *) = [7, 52, 6, 33]

```

```

x(5, *) = [11, 55, 9, 22]
x(6, *) = [3, 71, 17, 6]
x(7, *) = [1, 31, 22, 44]
x(8, *) = [2, 54, 18, 22]
x(9, *) = [21, 47, 4, 26]
x(10, *) = [1, 40, 23, 34]
x(11, *) = [11, 66, 9, 12]
x(12, *) = [10, 68, 8, 12]
y = [78.5, 74.3, 104.3, 87.6, 95.9, 109.2, $
     102.7, 72.5, 93.1, 115.9, 83.8, 113.3, 109.4]
coefs = IMSL_MULTIREGRESS(x, y, $
    Anova_Table      = anova_table, $
    T_Tests          = t_tests,      $
    Predict_Info     = predict_info, $
    Residual         = residual)
; Call IMSL_MULTIREGRESS to compute the fit.
predicted = IMSL_MULTIPREDICT(predict_info, x, $
    Ci_scheffe = ci_scheffe, $
    Y          = y,          $
    Dffits     = dffits)
print_results, anova_table, t_tests, y, $
predicted, ci_scheffe, residual, dffits
    * * Analysis of Variance * *
    df for among groups          4.0000
    df for within groups         8.0000
    total (corrected) df        12.0000
    ss for among groups         2667.8997
    ss for within groups         47.8637
    total (corrected) ss        2715.7634
    mean square among groups     666.9749
    mean square within groups     5.9830
    F-statistic                  111.4791
    P-value                      0.0000
    R-squared (in percent)       98.2376
    adjusted R-squared (in percent) 97.3563
    est. std of within group error 2.4460
    overall mean of y           95.4231
    coef. of variation (in percent) 2.5633
Coefficient   s.e.    t      p-value
62.41        70.07   0.89   0.40
1.55         0.74   2.08   0.07
0.51         0.72   0.70   0.50
0.10         0.75   0.14   0.90
-0.14        0.71  -0.20   0.84
observed    predicted    lower    upper    residual    dffits
78.50       78.50       70.70    86.29     0.00       0.00
74.30       72.79       66.73    78.85     1.51       0.52
104.30      105.97       97.99    113.95    -1.67     -1.24
87.60       89.33       83.62    95.03    -1.73     -0.53

```


95.90	95.65	89.37	101.93	0.25	0.09
109.20	105.27	101.57	108.98	3.93	0.76
102.70	104.15	97.79	110.51	-1.45	-0.55
72.50	75.67	68.96	82.39	-3.17	-1.64
93.10	91.72	86.02	97.42	1.38	0.42
115.90	115.62	106.83	124.41	0.28	0.30
83.80	81.81	74.96	88.66	1.99	0.93
113.30	112.33	106.94	117.71	0.97	0.26
109.40	111.69	105.91	117.48	-2.29	-0.76

Errors

Warning Errors

STAT_NONESTIMABLE—Within the preset tolerance, the linear combination of regression coefficients is nonestimable.

STAT_LEVERAGE_GT_1—Leverage (= #) much greater than 1.0 is computed. It is set to 1.0.

STAT_DEL_MSE_LT_0—Deleted residual mean square (= #) much less than zero is computed. It is set to zero.

Fatal Errors

STAT_NONNEG_WEIGHT_REQUEST_2—Weight for row # was #. Weights must be nonnegative.

Version History

6.4	Introduced

IMSL_ALLBEST

The IMSL_ALLBEST procedure selects the best multiple linear regression models.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
IMSL_ALLBEST, x, y [, ADJ_R_SQUARED=value] [, COEFS=variable]
[, COV_INPUT=array] [, COV_NOBS=value] [, CRITERIONS=variable]
[, /DOUBLE] [, FREQUENCIES=array] [, IDX_COEFS=variable]
[, IDX_CRITERIONS=variable] [, IDX_VARS=variable]
[, INDEP_VARS=variable] [, WEIGHTS=array] [, MALLOWS_CP=value]
[, MAX_N_BEST=value] [, MAX_N_GOOD=value] [, MAX_SUBSET=value])
```

Arguments

x

Two-dimensional array containing the data for the candidate variables.

y

One-dimensional array of length N_ELEMENTS ($x(*, 0)$) containing the responses for the dependent variable.

Keywords

ADJ_R_SQUARED

The adjusted R^2 criterion is used, where subset sizes 1, 2, ..., N_ELEMENTS ($x(*, 0)$) are examined. Keywords *Max_Subset*, *Adj_R_Squared*, and *MalloWS_Cp* cannot be used together.

COEFS

Named variable into which the two-dimensional array of size (*Idx_Coefs* (NTBEST)) x 5 containing statistics relating to the regression coefficients of the best models is

stored. Each row corresponds to a coefficient for a particular regression. The regressions are in order of increasing subset size. Within each subset size, the regressions are ordered so that the better regressions appear first. The statistic in the columns are as follows (inferences are conditional on the selected model):

- 0—variable number
- 1—coefficient estimate
- 2—estimated standard error of the estimate
- 3— t -statistic for the test that the coefficient is 0
- 4— p -value for the two-sided t test

Keywords *Coefs* and *Idx_Coefs* must be used together.

COV_INPUT

Two-dimensional square array of size $(N\_ELEMENTS(x(0, *)) + 1)$ by $(N\_ELEMENTS(x(0, *)) + 1)$ containing a variance-covariance or sum-of-squares and crossproducts matrix, in which the last column must correspond to the dependent variable.

Array *Cov_Input* can be computed using *IMSL_COVARIANCES*. Parameters x and y , and keywords *Frequencies* and *Weights* are not accessed when this option is specified. Normally, *IMSL_ALLBEST* computes *Cov_Input* from the input data matrices x and y . However, there may be cases when you will want to calculate the covariance matrix and manipulate it before calling *IMSL_ALLBEST*. See the *Discussion* section for a discussion of such cases.

Note

Keywords *Cov_Input* and *Cov_Nobs* must be used together.

COV_NOBS

Number of observations associated with array *Cov_Input*. Keywords *Cov_Input* and *Cov_Nobs* must be used together.

Note

Keywords *Cov_Input* and *Cov_Nobs* must be used together.

CRITERIONS

Named variable into which the one-dimensional array of length $\max(Idx\_Criteria(N\_SIZE - 1), N\_ELEMENTS(x(0, *)))$ containing in its first *Idx_Criteria* ($N\_SIZE -$

1) elements the criterion values for each subset considered, in increasing subset size order, is stored. Keywords *Criteria*s and *Idx_Criteria*s must be used together.

DOUBLE

If present and nonzero, double precision is used.

FREQUENCIES

One-dimensional array of length $N\_ELEMENTS$ ($x(*, 0)$) containing the frequency for each row of x . Default: *Frequencies* (*) = 1

IDX_COEFS

Named variable into which the one-dimensional array of length $NBEST + 1$ containing the locations of *Coefficients* the first row of each of the best regressions is stored. Here, $NBEST$ is the total number of best regression found and is $Max\_Subset * Max\_N\_Best$ if *Max_Subset* is specified, Max_N_Best if either *Mallows_Cp* or *Adj_R_Squared* is specified, and $Max\_N\_Best * (N\_ELEMENTS(x(0, *)))$ otherwise. For $i = 0, 1, \dots, NBEST$, rows *Idx_Coefs* (i), *Idx_Coefs*($i + 1$), ..., *Idx_Coefs* ($i + 1$) - 1 of *Coefs* correspond to the ($i + 1$)-st regression. Keywords *Coefs* and *Idx_Coefs* must be used together.

IDX_CRITERIONS

Named variable into which the one-dimensional array of length $NSIZE$ containing the locations in *Criteria*s of the first element for each subset size is stored. $NSIZE$ is calculated as follows: $NSIZE = (Max\_Subset + 1)$ if *Max_Subset* is set. $NSIZE = (N\_ELEMENTS(x(0, *)) + 1)$ otherwise. For $i = 0, 1, \dots, NSIZE - 2$, element numbers *Idx_Criteria*s(i), *Idx_Criteria*s ($i + 1$), ..., *Idx_Criteria*s($i + 1$) - 1 of *Criteria*s correspond to the ($i + 1$)-st subset size. Keywords *Criteria*s and *Idx_Criteria*s must be used together.

IDX_VARS

Named variable into which the one-dimensional array of length $NSIZE$ containing the locations in *Indep_Vars* of the first element for each subset size. $NSIZE$ is calculated as follows: $NSIZE = (Max\_Subset + 1)$ if *Max_Subset* is set. $NSIZE = (N\_ELEMENTS(x(0, *)) + 1)$ otherwise. For $i = 0, 1, \dots, NSIZE - 2$, element numbers *Idx_Vars*(i), *Idx_Vars* ($i + 1$), ..., *Idx_Vars* ($i + 1$) - 1 of *Indep_Vars* correspond to the ($i + 1$)-st subset size. Keywords *Indep_Vars* and *Idx_Vars* must be used together.

INDEP_VARS

Named variable into which the one-dimensional array of length *Idx_Vars* ($\text{NSIZE} - 1$) containing the variable numbers for each subset considered and in the same order as in *Criteria* is stored. Keywords *Indep_Vars* and *Idx_Vars* must be used together.

WEIGHTS

One-dimensional array of length N_ELEMENTS ($x(*, 0)$) containing the weight for each row of x . Default: $\text{Weights}(*) = 1$

MALLOWS_CP

Mallows C_p criterion is used, where subset sizes 1, 2, ..., N_ELEMENTS ($x(*, 0)$) are examined. Keywords *Max_Subset*, *Adj_R_Squared*, and *Mallows_Cp* cannot be used together.

MAX_N_BEST

Number of best regressions to be found. If the R^2 criterion is selected, the *Max_N_Best* best regressions for each subset size examined are found. If the adjusted R^2 or Mallows C_p criterion is selected, the *Max_N_Best* overall regressions are found. Default: $\text{Max\_N\_Best} = 1$

MAX_N_GOOD

Maximum number of good regressions of each subset size to be saved in finding the best regressions. Keyword *Max_N_Good* must be greater than or equal to *Max_N_Best*. Normally, *Max_N_Good* should be less than or equal to 10. It need not ever be larger than the maximum number of subsets for any subset size. Computing time required is inversely related to *Max_N_Good*. Default: $\text{Max\_N\_Good} = 10$

MAX_SUBSET

The R^2 criterion is used, where subset sizes 1, 2, ..., *Max_Subset* are examined. This option is the default with $\text{Max\_Subset} = \text{N\_ELEMENTS}$ ($x(0, *)$). Keywords *Max_Subset*, *Adj_R_Squared*, and *Mallows_Cp* cannot be used together.

Discussion

The IMSL_ALLBEST procedure finds the best subset regressions for a regression problem with

$$n_{\text{candidate}} = (\text{N\_ELEMENTS } (x(0, *)))$$

independent variables. Typically, the intercept is forced into all models and is not a candidate variable. In this case, a sum-of-squares and crossproducts matrix for the independent and dependent variables corrected for the mean is computed internally. There may be cases when it is convenient for you to calculate the matrix; see the description of the *Cov_Input* optional parameter.

“Best” is defined, on option, by one of the following three criteria:

- R^2 (in percent):

$$R^2 = 100 \left(1 - \frac{SSE_p}{SST} \right)$$

- R_a^2 (adjusted R^2 in percent):

$$R_a^2 = 100 \left[1 - \left(\frac{n-1}{n-p} \right) \frac{SSE_p}{SST} \right]$$

Note that maximizing the criterion is equivalent to minimizing the residual mean square:

$$\frac{SSE_p}{(n-p)}$$

- Mallows' C_p statistic:

$$C_p = \frac{SSE_p}{s_{n\_candidate}^2} + 2p - n$$

Here, n is equal to the sum of the frequencies (or $N\_ELEMENTS(x(*, 0))$ if *Frequencies* is not specified) and SST is the total sum of squares. SSE_p is the error sum of squares in a model containing p regression parameters including β_0 (or $p - 1$ of the $n\_candidate$ candidate variables). Variable is the $s_{n\_candidate}^2$ error mean square from the model with all $n\_candidate$ variables in the model. Hocking (1972) and Draper and Smith (1981, pp. 296–302) discuss these criteria.

The *IMSL_ALLBEST* procedure is based on the algorithm of Furnival and Wilson (1974). This algorithm finds *Max_N_Good* candidate regressions for each possible subset size. These regressions are used to identify a set of best regressions. In large problems, many regressions are not computed. They may be rejected without computation based on results for other subsets; this yields an efficient technique for considering all possible regressions.

There are cases when you may wish to input the variance-covariance matrix rather than allow the *IMSL_ALLBEST* procedure to calculate it. This can be accomplished

using keyword *Cov_Input*. Three situations in which you may want to do this are as follows:

1. The intercept is not in the model. A raw (uncorrected) sum-of-squares and crossproducts matrix for the independent and dependent variables is required. Keyword *Cov_Nobs* must be set to 1 greater than the number of observations. Form $A^T A$, where $A = [A, Y]$, to compute the raw sum-of-squares and crossproducts matrix.
2. An intercept is to be a candidate variable. A raw (uncorrected) sum-of-squares and crossproducts matrix for the constant regressor ($= 1.0$), independent variables, and dependent variables is required for *Cov_Input*. In this case, *Cov_Input* contains one additional row and column corresponding to the constant regressor. This row/column contains the sum of squares and crossproducts of the constant regressor with the independent and dependent variables. The remaining elements in *Cov_Input* are the same as in the previous case. Keyword *Cov_Nobs* must be set to 1 greater than the number of observations.
3. There are m variables to be forced into the models. A sum-of-squares and crossproducts matrix adjusted for the m variables is required (calculated by regressing the candidate variables on the variables to be forced into the model). Keyword *Cov_Nobs* must be set to m less than the number of observations.

Programming Notes

The IMSL_ALLBEST procedure saves considerable CPU time over explicitly computing all possible regressions. However, the procedure has some limitations that can cause unexpected results for users who are unaware of the limitations of the software.

1. For $n\_candidate + 1 > -\log_2(\epsilon)$, where ϵ is machine precision, some results may be incorrect. This limitation arises because the possible models indicated (the model numbers 1, 2, ..., $2^{n\_candidate}$) are stored as floating-point values; for sufficiently large $n\_candidate$, the model numbers cannot be stored exactly. On many computers, this means IMSL_ALLBEST (for $n\_candidate > 24$; single precision) and IMSL_ALLBEST (for $n\_candidate > 49$; double precision) can produce incorrect results.
2. The IMSL_ALLBEST procedure eliminates some subsets of candidate variables by obtaining lower bounds on the error sum of squares from fitting larger models. First, the full model containing all $n\_candidate$ is fit sequentially using a forward stepwise procedure in which one variable enters the model at a time, and criterion values and model numbers for all the

candidate variables that can enter at each step are stored. If linearly dependent variables are removed from the full model, error `STAT_VARIABLES_DELETED` is issued. If this error is issued, some submodels that contain variables removed from the full model because of linear dependency can be overlooked if they have not already been identified during the initial forward stepwise procedure. If error `STAT_VARIABLES_DELETED` is issued and you want the variables that were removed from the full model to be considered in smaller models, rerun the program with a set of linearly independent variables.

Example

This example uses a data set from Draper and Smith (1981, pp. 629-630). The `IMSL_ALLBEST` procedure is used to find the best regression for each subset size using the Mallows's C_p statistic as the criterion. Note that when Mallows's C_p statistic (or adjusted R^2) is specified, the variable `Max_N_Best` indicates the *total* number of “best” regressions (rather than indicating the number of best regressions *per subset* size, as in the case of the R^2 criterion). In this example, the three best regressions are found to be (1, 2), (1, 2, 4), and (1, 2, 3).

```
PRO IMSL_ALLBEST_ex1
; Define the data set.
x = transpose( [ [7., 26., 6., 60.], [1., 29., 15., 52.], $
  [11., 56., 8., 20.], [11., 31., 8., 47.], $
  [7., 52., 6., 33.], [11., 55., 9., 22.], $
  [3., 71., 17., 6.], [1., 31., 22., 44.], $
  [2., 54., 18., 22.], [21., 47., 4., 26.], $
  [1., 40., 23., 34.], [11., 66., 9., 12.], $
  [10., 68., 8., 12.]] )
y = [78.5, 74.3, 104.3, 87.6, 95.9, 109.2, 102.7, 72.5, $
  93.1, 115.9, 83.8, 113.3, 109.4]
Max_N_Best = 3
IMSL_ALLBEST, x, y, Max_N_Best = max_n_best, /Mallows_Cp, $
  Idx_Coefs = idx_coefs, $
Coefs = coefs
PRINT, '          * * * Idx_Coefs and Coefs in raw form * * * '
; First, the two important matrices, Idx_Coefs and Coefs,
; are printed to display how they appear as output from
; IMSL_ALLBEST.
PRINT
PM, idx_coefs, Title = 'Idx_Coefs:'
PRINT
PM, Coefs, Title = 'Coefs'
PRINT
ntbest = max_n_best
; Next, describe how to break apart Coefs by regressions
; based on values of Idx_Coefs. Note: NTBEST is defined under
```



```

; description of keyword Idx_Coefs.
PRINT, '          * * * How Idx_Coefs describes Coefs * * *'
PRINT
FOR i = 0, ntbest - 1 DO $
    PRINT, 'regression', i+1, 'begins at row', Idx_Coefs(i), $
        ' of Coefs.', FORMAT = '(a, i2, a, i2, a)'
PRINT
PRINT, '* * * Coefs separated by ', 'regressions * * *'
; Next, Coefs is broken apart by regressions, using Idx_Coefs.
; Note: The final element of Idx_Coefs is not a row number but
; instead is equal to the total number of rows in Coefs.
PRINT
FOR i = 0, ntbest - 1 DO begin
    start = idx_coefs(i)
    stop = idx_coefs(i + 1) - 1
    FOR j = start, stop DO begin
        PRINT, coefs(j, *), FORMAT = '(5f9.4)'
    END
END
PRINT
END
PRINT, '          * * * Best Regressions* * *'
; Finally, regression labels, column labels, etc., are added.
PRINT
FOR i = 0, ntbest - 1 DO begin
    start = idx_coefs(i)
    stop = idx_coefs(i + 1) - 1
    count = stop - start + 1
    PRINT, 'Best Regression with', count, $
        'variables(s) (Mallows CP)', FORMAT = '(a, i2, a)'
PRINT, 'variable    coefficient std error      t          p-value'
    FOR j = start, stop DO $
        PRINT, coefs(j, *), FORMAT = '(i5, 2x, 4f11.4)'
    PRINT
END
END
* * * Idx_Coefs and Coefs in raw form * * *
PM, Idx_Coefs
0
2
5
8
PM, Coefs
1.00000      1.46831      0.121301      12.1046  2.38419e-07
2.00000      0.662251     0.0458547      14.4424      0.00000
1.00000      1.45194      0.116998      12.4099  5.96046e-07
2.00000      0.416112     0.185611      2.24185     0.0516866
4.00000     -0.236538     0.173288     -1.36500     0.205401
1.00000      1.69589      0.204582      8.28953  1.66893e-05
2.00000      0.656915     0.0442343     14.8508  1.19209e-07

```

```

      3.00000      0.250018      0.184711      1.35356      0.208889
* * * How Idx_Coefs describes Coefs * * *
regression 1 begins at row  0 of Coefs.
regression 2 begins at row  2 of Coefs.
regression 3 begins at row  5 of Coefs.
* * * Coefs separated by regressions * * *
1.0000  1.4683  0.1213 12.1046  0.0000
2.0000  0.6623  0.0459 14.4424  0.0000
1.0000  1.4519  0.1170 12.4099  0.0000
2.0000  0.4161  0.1856  2.2419  0.0517
4.0000 -0.2365  0.1733 -1.3650  0.2054
1.0000  1.6959  0.2046  8.2895  0.0000
2.0000  0.6569  0.0442 14.8508  0.0000
3.0000  0.2500  0.1847  1.3536  0.2089
* * * Best Regressions* * *
Best Regression with 2 variable(s) (Mallows CP)
variable coefficient      std error      t      p-value
  1      1.4683      0.1213      12.1046      0.0000
  2      0.6623      0.0459      14.4424      0.0000
Best Regression with 3 variable(s) (Mallows CP)
variable coefficient      std error      t      p-value
  1      1.4519      0.1170      12.4099      0.0000
  2      0.4161      0.1856      2.2419      0.0517
  4     -0.2365      0.1733     -1.3650      0.2054
Best Regression with 3 variable(s) Mallows CP)
variable coefficient std error      t      p-value
  1      1.6959      0.2046      8.2895      0.0000
  2      0.6569      0.0442     14.8508      0.0000
  3      0.2500      0.1847      1.3536      0.2089
```

Errors

Warning Errors

STAT_VARIABLES_DELETED—At least one variable is deleted from the full model because the variance-covariance matrix Cov is singular.

Fatal Errors

STAT_NO_VARIABLES—No variables can enter any model.

Version History

6.4	Introduced

IMSL_STEPWISE

The IMSL_STEPWISE procedure builds multiple linear regression models using forward, backward, or stepwise selection.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
IMSL_STEPWISE, x, y [, /ALL_STEPS] [, ANOVA_TABLE=variable]
    [, /BACKWARD] [, COV_NOBS=value] [, COV_INPUT=array]
    [, COEF_T_TESTS=variable] [, COEF_VIF=variable]
    [, COV_SWEPT=variable] [, /DOUBLE] [, /FIRST_STEP] [, FORCE=value]
    [, /FORWARD] [, FREQUENCIES=array] [, HISTORY=variable]
    [, /INTER_STEP] [, /LAST_STEP] [, IEND=variable] [, LEVEL=array]
    [, N_STEPS=value] [, P_IN=value] [, P_OUT=value]
    [, /STEPWISE] [, SWEPT=value] [, /TOLERANCE] [, WEIGHTS=array])
```

Arguments

x

Two-dimensional array containing the data for the candidate variables.

y

Array of length N_ELEMENTS(x(*, 0)) containing the responses for the dependent variable.

Keywords

ALL_STEPS

This is the only invocation. Initialization, stepping, and wrap-up computations are performed.

Note

One or none of these options — **First_Step**, **Inter_Step**, **Last_Step**, and **All_Steps** — can be specified. If none of these is specified, the action defaults to *All_Steps*.

ANOVA_TABLE

Named variable into which the one-dimensional array containing the analysis of variance table is stored. The analysis of variance statistics are as follows:

- 0—degrees of freedom for regression
- 1—degrees of freedom for error
- 2—total degrees of freedom
- 3—sum of squares for regression
- 4—sum of squares for error
- 5—total sum of squares
- 6—regression mean square
- 7—error mean square
- 8— F -statistic
- 9— p -value
- 10— R^2 (in percent)
- 11—adjusted R^2 (in percent)
- 12—estimate of the standard deviation

BACKWARD

An attempt is made to remove a variable from the model. A variable is removed if its p -value exceeds P_Out . During initialization, all candidate independent variables enter the model.

Note

One or none of these options — **Forward**, **Backward**, **Stepwise** — can be specified. If none is specified, the action defaults to *Backward*.

COV_NOBS

The number of observations associated with array *Cov_Input*. Keywords *Cov_Input* and *Cov_Nobs* must be used together.

Note

Keywords *Cov_Input* and *Cov_Nobs* must be used together.

COV_INPUT

Two-dimensional square array of size $(N\_ELEMENTS(x(0,*)) + 1) \times (N\_ELEMENTS(x(0,*)) + 1)$ containing a variance-covariance or sum-of-squares and crossproducts matrix, in which the last column must correspond to the dependent variable.

Array *Cov_Input* can be computed using *IMSL_COVARIANCES*. Parameters *x* and *y*, and keywords *Frequencies* and *Weights* are not accessed when this option is specified. Normally, *IMSL_ALLBEST* computes *Cov_Input* from the input data matrices *x* and *y*. However, there may be cases when you want to calculate the covariance matrix and manipulate it before calling *IMSL_ALLBEST*. See the *Discussion* section for a discussion of such cases.

Note

Keywords *Cov_Input* and *Cov_Nobs* must be used together.

COEF_T_TESTS

Named variable into which the two-dimensional array containing statistics relating to the regression coefficient for the final model in this invocation is stored. The rows correspond to the $N\_ELEMENTS(x(0,*))$ in dependent variables. The rows are in the same order as the variables in *x* (or, if *Cov_Input* is specified, the rows are in the same order as the variables in *Cov_Input*). Each row corresponding to a variable not in the model contains statistics for a model which includes the variables of the final model and the variable corresponding to the row in question.

- 0—coefficient estimate
- 1—estimated standard error of the coefficient estimate
- 2—*t*-statistic for the test that the coefficient is zero
- 3—*p*-value for the two-sided *t* test

COEF_VIF

Named variable into which the two-dimensional array containing variance inflation factors for the final model in this invocation is stored. The elements correspond to the `N_ELEMENTS (x(0, *))` in dependent variables. The elements are in the same order as the variables in `x` (or, if `Cov_Input` is specified, the elements are in the same order as the variables in `Cov_Input`). Each element corresponding to a variable not in the model contains statistics for a model which includes the variables of the final model and the variables corresponding to the element in question.

The square of the multiple correlation coefficient for the i -th regressor after all others have been obtained from $VIF = Coef\_Vif(i)$ by the following formula:

$$1.0 - (1.0/VIF)$$

COV_SWEEP

Named variable into which the two-dimensional array of size `N_ELEMENTS (x(0, *)) + 1` x `N_ELEMENTS (x(0, *)) + 1` that results after `Cov_Swept` has been swept on the columns corresponding to the variables in the model. The estimated variance-covariance matrix of the estimated regression coefficients in the final model can be obtained by extracting the rows and columns of `Cov_Swept` corresponding to the independent variables in the final model and multiplying the elements of this matrix by `Anova_Table(7)`.

DOUBLE

If present and nonzero, double precision is used.

FIRST_STEP

This is the first invocation; additional calls will be made. Initialization and stepping is performed.

Note

One or none of these options — **First_Step**, **Inter_Step**, **Last_Step**, and **All_Steps** — can be specified. If none of these is specified, the action defaults to *All_Steps*.

FORCE

Scalar integer specifying how variables are forced into the model as independent variables. Variable with levels 1, 2, ..., *Force* are forced into the model as independent variables. See *Level*.

FORWARD

An attempt is made to add a variable to the model. A variable is added if its p -value is less than P_In . During initialization, only the forced variables enter the model.

Note

One or none of these options — **Forward**, **Backward**, **Stepwise** — can be specified. If none is specified, the action defaults to *Backward*.

FREQUENCIES

One-dimensional array containing the frequency for each row of x . Default: $Frequencies(*) = 1$

HISTORY

Named variable into which the one-dimensional array of length $N\_ELEMENTS(x(0, *) + 1)$ containing the recent history of the independent variables is stored.

Element $History(N\_ELEMENTS(x(0, *)))$ usually corresponds to the dependent variable (see *Level*) as shown in [Table 14-6](#).

<i>History (i)</i>	Status of i -th Variable
0.0	Variable has never been added to model.
0.5	Variable was added into the model during initialization.
$k > 0.0$	Variable was added to the model during the k -th step.
$k < 0.0$	Variable was deleted from model during the k -th step.

Table 14-6: History Variable

INTER_STEP

This is an intermediate invocation. Stepping is performed.

Note

One or none of these options — **First_Step**, **Inter_Step**, **Last_Step**, and **All_Steps** — can be specified. If none of these is specified, the action defaults to *All_Steps*.

LAST_STEP

This is the final invocation. Stepping and wrap-up computations are performed.

Note

One or none of these options — **First_Step**, **Inter_Step**, **Last_Step**, and **All_Steps** — can be specified. If none of these is specified, the action defaults to *All_Steps*.

IEND

Named variable into which an integer which indicates whether additional steps are possible is stored.

- 0—Additional steps may be possible.
- 1—No additional steps are possible.

LEVEL

Array of length $N\_ELEMENTS(x(0, *)) + 1$ containing levels of priority for variables entering and leaving the regression. Each variable is assigned a positive value that indicates its level of entry into the model. A variable can enter the model only after all variables with smaller nonzero levels of entry have entered. Similarly, a variable can only leave the model after all variables with higher levels of entry have left. Variables with the same level of entry compete for entry (deletion) at each step. $Level(i) = 0$ means the i -th variable is never to enter the model. $Level(i) = -1$ means the i -th variable is the dependent variable. $Level(N\_ELEMENTS(x(0, *)))$ must correspond to the dependent variable, except when *Cov_Input* is specified. Default: 1, 1, ..., 1, -1, where -1 corresponds to $Level(N\_ELEMENTS(x(0, *)))$

N_STEPS

For nonnegative N_Steps , N_Steps steps are taken. If $N\_Steps = -1$, stepping continues until completion. Default: $N\_Steps = 1$

Note

Keyword N_Steps is not referenced if *All_Steps* is used.

P_IN

Largest p -value for variable entering the model. Variables with p -values less than P_In may enter the model. Default: $P\_In = 0.05$

P_OUT

Smallest p -value for removing variables with p -values greater than P_Out may leave the model. Keyword P_Out must be greater than or equal to P_In . A common choice for P_Out is $2 * P\_In$. Default: $P\_Out = 0.10$

STEPWISE

A backward step is attempted. If a variable is not removed, a forward step is attempted. This is a stepwise step. Only the forced variables enter the model during initialization.

Note

One or none of these options — **Forward, Backward, Stepwise** — can be specified. If none is specified, the action defaults to *Backward*.

SWEPT

Named variable into which the one-dimensional array of length $(N\_ELEMENTS(x(0, *) + 1)$ with information to indicate the independent variables in the model is stored. Keyword *Swept* ($N\_ELEMENTS(x(0, *))$) usually corresponds to the dependent variable (see *Level*).

- -1—Variable i is not in model.
- 1—Variable i is in model.

TOLERANCE

Tolerance used in determining linear dependence. Default: $Tolerance = 100 * \epsilon$, where ϵ is machine precision.

WEIGHTS

One-dimensional array containing the weight for each row of x . Default: $Weights(*) = 1$

Discussion

The `IMSL_STEPWISE` procedure builds a multiple linear regression model using forward, backward, or forward stepwise (with a backward glance) selection. The `IMSL_STEPWISE` procedure is designed so you can monitor, and perhaps change, the variables added (deleted) to (from) the model after each step. In this case, multiple calls to `IMSL_STEPWISE` (using keywords *First_Step*, *Inter_Step*, or

Last_Step) are made. Alternatively, `IMSL_STEPWISE` can be invoked once (default, or specify keyword *All_Steps*) in order to perform the stepping until a final model is selected.

Levels of priority can be assigned to the candidate independent variables (use keyword *Level*). All variables with a priority level of 1 must enter the model before variables with a priority level of 2. Similarly, variables with a level of 2 must enter before variables with a level of 3, etc. Variables also can be forced into the model (see keyword *Force*). Note that specifying keyword *Force* without also specifying keyword *Level* results in all variables being forced into the model.

Typically, the intercept is forced into all models and is not a candidate variable. In this case, a sum-of-squares and crossproducts matrix for the independent and dependent variables corrected for the mean is used. Other possibilities are as follows:

- The intercept is not in the model. A raw (uncorrected) sum-of-squares and crossproducts matrix for the independent and dependent variables is required as input in *Cov_Input*. Keyword *Cov_Nobs* must be set to 1 greater than the number of observations.
- An intercept is to be a candidate variable. A raw (uncorrected) sum-of-squares and crossproducts matrix for the constant regressor (=1), independent and dependent variables are required for *Cov_Input*. In this case, *Cov_Input* contains one additional row and column corresponding to the constant regressor. This row/column contains the sum-of-squares and crossproducts of the constant regressor with the independent and dependent variables. The remaining elements in *Cov_Input* are the same as in the previous case. Keyword *Cov_Nobs* must be set to 1 greater than the number of observations.

The stepwise regression algorithm is due to Efron (1960). The `IMSL_STEPWISE` procedure uses sweeps of the covariance matrix (input using keyword *Cov_Input*, if specified, or generated internally by default) to move variables in and out of the model (Hemmerle 1967, Chapter 3). The SWEEP operator discussed in Goodnight (1979) is used. A description of the stepwise algorithm also is given by Kennedy and Gentle (1980, pp. 335–340). The advantage of stepwise model building over all possible regression (see “[IMSL_ALLBEST](#)” on page 630) is that it is less demanding computationally when the number of candidate independent variables is very large. However, there is no guarantee that the model selected will be the best model (highest R^2) for any subset size of independent variables.

Example

This example uses a data set from Draper and Smith (1981, pp. 629-630). Backwards stepping is performed by default. First, a procedure to output the results is defined.

```

PRO print_results, anova_table, t, s
labels = ['df for regression', 'df for error', 'total df', 'ss for regression', 'ss for error', 'total ss', 'mean square for regression', 'mean square error', 'F-statistic', 'p-value', 'R-squared (in percent)', 'adjusted R-squared (in percent)']

PRINT
PRINT, '          * * Analysis of Variance * *'
; Print the table.
FOR i = 0, 11 DO PRINT, labels(i), $
    anova_table(i), FORMAT = '(a32,f8.2)'
PRINT
PRINT, '* * Inference on Coefficients * *'
PRINT, '          Estimate      s.e.          t' + $
        '          prob>t      swept'
PRINT, '$(a, 4f10.4)', 'variable 1', t(0,*), s(0)
PRINT, '$(a, 4f10.4)', 'variable 2', t(1,*), s(1)
PRINT, '$(a, 4f10.4)', 'variable 3', t(2,*), s(2)
PRINT, '$(a, 4f10.4)', 'variable 4', t(3,*), s(3)
END

x = MAKE_ARRAY(13, 4)
; Define the data.
x(0, *) = [7., 26., 6., 60.]
x(1, *) = [1., 29., 15., 52.]
x(2, *) = [11., 56., 8., 20.]
x(3, *) = [11., 31., 8., 47.]
x(4, *) = [7., 52., 6., 33.]
x(5, *) = [11., 55., 9., 22.]
x(6, *) = [3., 71., 17., 6.]
x(7, *) = [1., 31., 22., 44.]
x(8, *) = [2., 54., 18., 22.]
x(9, *) = [21., 47., 4., 26.]
x(10, *) = [1., 40., 23., 34.]
x(11, *) = [11., 66., 9., 12.]
x(12, *) = [10., 68., 8., 12.]
y = [78.5, 74.3, 104.3, 87.6, 95.9, $
    109.2, 102.7, 72.5, 93.1, 115.9, 83.8, 113.3, 109.4]
IMSL_STEPWISE, x, y, Anova_Table = anova_table, $
    Coef_T_Tests = t, swept = s
; Backward stepwise regression.
print_results, anova_table, t, s
          * * Analysis of Variance * *

```

```
df for regression          2.00
df for error              10.00
total df                  12.00
ss for regression         2657.86
ss for error              57.90
total ss                  2715.76
mean square for regression 1328.93
mean square error         5.79
F-statistic               229.50
P-value                   0.00
R-squared (in percent)    97.87
adjusted R-squared (in percent) 97.44
* * Inference on Coefficients * *
      Estimate    s.e.      t      prob>t      swept
variable 1      1.4683    0.1213   12.1046   0.0000       1.
variable 2      0.6623    0.0459   14.4423   0.0000       1.
variable 3      0.2500    0.1847    1.3536   0.2089      -1.
variable 4     -0.2365    0.1733   -1.3650   0.2054      -1.
```

Errors

Warning Errors

STAT_LINEAR_DEPENDENCE_1—Based on *Tolerance* = #, there are linear dependencies among the variables to be forced.

Fatal Errors

STAT_NO_VARIABLES_ENTERED—No variables entered the model. All elements of *Anova_Table* are set to NaN.

Version History

6.4	Introduced

IMSL_POLYREGRESS

The IMSL_POLYREGRESS function performs a polynomial least-squares regression.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_POLYREGRESS(x, y, degree [, ANOVA_TABLE=variable]
    [, DF_PURE_ERROR=variable] [, /DOUBLE] [, PREDICT_INFO=variable]
    [, RESIDUAL=variable] [, SSQ_LOF=variable] [, SSQ_POLY=variable]
    [, SSQ_PURE_ERROR=variable] [, WEIGHTS=array] [, XMEAN=variable]
    [, XVARIANCE=variable])
```

Return Value

An array of size *degree* + 1 containing the coefficients of the fitted polynomial.

Arguments

degree

Degree of the polynomial.

x

One-dimensional array containing the independent variable.

y

One-dimensional array containing the dependent variable.

Keywords

ANOVA_TABLE

Named variable into which the array containing the analysis of variance table is stored. The analysis of variance statistics are given as follows:

- 0—degrees of freedom for the model
- 1—degrees of freedom for error
- 2—total (corrected) degrees of freedom
- 3—sum of squares for the model
- 4—sum of squares for error
- 5—total (corrected) sum of squares
- 6—model mean square
- 7—error mean square
- 8—overall F -statistic
- 9— p -value
- 10— R^2 (in percent)
- 11—adjusted R^2 (in percent)
- 12—estimate of the standard deviation
- 13—overall mean of y
- 14—coefficient of variation (in percent)

DF_PURE_ERROR

Named variable into which the degrees of freedom for pure error is stored.

DOUBLE

If present and nonzero, double precision is used.

PREDICT_INFO

Named variable into which the one-dimensional byte array containing information needed by IMSL_POLYPREDICT is stored. The data contained in this array is in an encrypted format and should not be altered before it is used in subsequent calls to IMSL_POLYPREDICT.

RESIDUAL

Named variable into which the array containing the residuals is stored.

SSQ_LOF

Named variable into which the array containing the lack-of-fit statistics is stored.

Elements ($i, *$) correspond to x^{i+1} , $i = 0, \dots, (\text{degree} - 1)$, and the contents of the array are described in [Table 14-7](#).

Element	Description
($i, 0$)	degrees of freedom
($i, 1$)	lack-of-fit sum of squares
($i, 2$)	F -statistic for testing lack-of-fit for a polynomial model of degree i
($i, 3$)	p -value for the test

Table 14-7: Ssq_Lof Array Elements

SSQ_POLY

Named variable into which the array containing the sequential sum of squares and other statistics are stored.

Elements ($i, *$) correspond to x^{i+1} , $i = 0, \dots, (\text{degree} - 1)$, and the contents of the array are described in [Table 14-8](#).

Element	Description
($i, 0$)	degrees of freedom
($i, 1$)	sum of squares
($i, 2$)	F -statistic
($i, 3$)	p -value

Table 14-8: Ssq_Poly Array Elements

SSQ_PURE_ERROR

Named variable into which the sum of squares for pure error is stored.

WEIGHTS

Array containing the vector of weights for the observation. If this option is not specified, all observations have equal weights of 1.

XMEAN

Named variable into which the mean of x is stored.

XVARIANCE

Named variable into which the variance of x is stored.

Discussion

The `IMSL_POLYREGRESS` function computes estimates of the regression coefficients in a polynomial (curvilinear) regression model. In addition to the computation of the fit, `IMSL_POLYREGRESS` computes some summary statistics. Sequential sum of squares attributable to each power of the independent variable (returned by using `Ssq_Poly`) are computed. These are useful in assessing the importance of the higher order powers in the fit. Draper and Smith (1981, pp. 101–102) and Neter and Wasserman (1974, pp. 278–287) discuss the interpretation of the sequential sum of squares.

The statistic R^2 is the percentage of the sum of squares of y about its mean explained by the polynomial curve. Specifically:

$$R^2 = \frac{\sum w_i (\hat{y}_i - \bar{y})^2}{\sum w_i (y_i - \bar{y})^2} 100\%$$

where w_i is the weight.

$$\hat{y}_i$$

is the fitted y value at x_i and

$$\bar{y}$$

is the mean of y . This statistic is useful in assessing the overall fit of the curve to the data. R^2 must be between 0% and 100%, inclusive. $R^2 = 100\%$ indicates a perfect fit to the data.

Estimates of the regression coefficients in a polynomial model are computed using orthogonal polynomials as the regressor variables. This reparameterization of the polynomial model in terms of orthogonal polynomials has the advantage that the loss

of accuracy resulting from forming powers of the x-values is avoided. All results are returned to you for the original model (power form).

The IMSL_POLYREGRESS function is based on the algorithm of Forsythe (1957). A modification to Forsythe's algorithm suggested by Shampine (1975) is used for computing the polynomial coefficients. A discussion of Forsythe's algorithm and Shampine's modification appears in Kennedy and Gentle (1980, pp. 342–347).

Examples

Example 1

A polynomial model is fitted to data discussed by Neter and Wasserman (1974, pp. 279–285). The data set contains the response variable y measuring coffee sales (in hundred gallons) and the number of self-service coffee dispensers. Responses for fourteen similar cafeterias are in the data set. The results are shown in [Figure 14-3](#).

```
x = [0, 0, 1, 1, 2, 2, 4, 4, 5, 5, 6, 6, 7, 7]
y = [508.1, 498.4, 568.2, 577.3, 651.7, 657.0, 755.3, 758.9, $
     787.6, 792.1, 841.4, 831.8, 854.7, 871.4]
; Define the data vectors.
coefs = IMSL_POLYREGRESS(x, y, 2)
PM, Coefs, Title = 'Least-Squares Polynomial Coefficients'
Least-Squares Polynomial Coefficients
    503.346
    78.9413
   -3.96949
x2 = 9 * FINDGEN(100)/99 - 1
PLOT, x2, coefs(0) + coefs(1) * x2 + coefs(2) * x2^2
OPLOT, x, y, Psym = 1
```

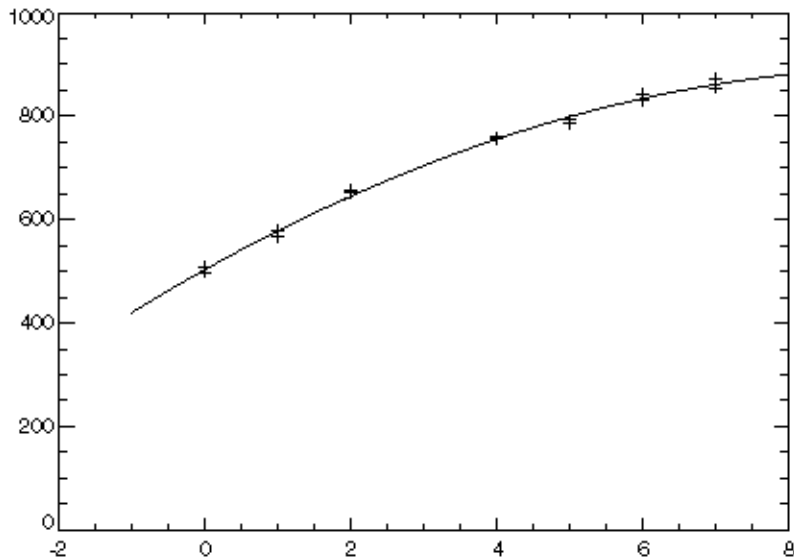


Figure 14-3: Least-Squares Regression Plot

Example 2

This example is a continuation of the initial example. Here, a procedure is called and defined to output the coefficients and analysis of variance table.

```

PRO print_results, coefs, anova_table
; The following procedure prints coefficients and the analysis of
; variance table.
coef_labels = ['intercept', 'linear', 'quadratic']
PM, coef_labels, coefs, Title = $
    'Least-Squares Polynomial Coefficients', $
    FORMAT = '(3a20, /, 3f20.4, /)'
anova_labels = ['degrees of freedom for regression', $
    'degrees of freedom for error', $
    'total (corrected) degrees of freedom', $
    'sum of squares for regression', $
    'sum of squares for error', $
    'total (corrected) sum of squares', $
    'regression mean square', $
    'error mean square', 'F-statistic', $
    'p-value', 'R-squared (in percent)', $
    'adjusted R-squared (in percent)', $
    'est. standard deviation of model error', $

```

```

    'overall mean of y', 'coefficient of variation (in percent)']
FOR i = 0, 14 DO PM, anova_labels(i), $
    anova_table(i), FORMAT = '(a40, f20.2)'
END
x = [0, 0, 1, 1, 2, 2, 4, 4, 5, 5, 6, 6, 7, 7]
y = [508.1, 498.4, 568.2, 577.3, 651.7, $
    657.0, 755.3, 758.9, 787.6, 792.1, 841.4, 831.8, 854.7, 871.4]
; Define the data vectors.
Coefs = IMSL_POLYREGRESS(x, y, 2, Anova_Table = anova_table)
; Call IMSL_POLYREGRESS with keyword Anova_Table.
print_results, coefs, anova_table
; Call the procedure defined above to output the results.
Least-Squares Polynomial Coefficients
intercept          linear          quadratic
503.3459           78.9413          -3.9695
* * * Analysis of Variance * * *
degrees of freedom for regression      2.00
degrees of freedom for error          11.00
total (corrected) degrees of freedom  13.00
sum of squares for regression         225031.94
sum of squares for error               710.55
total (corrected) sum of squares      225742.48
regression mean square                112515.97
error mean square                     64.60
F-statistic                          1741.86
p-value                              0.00
R-squared (in percent)                99.69
adjusted R-squared (in percent)       99.63
est. standard deviation of model error 8.04
overall mean of y                     710.99
coefficient of variation (in percent) 1.13

```

Errors

Warning Errors

STAT_CONSTANT_YVALUES—The y values are constant. A zero order polynomial is fit. High order coefficients are set to zero.

STAT_FEW_DISTINCT_XVALUES—There are too few distinct x values to fit the desired degree polynomial. High order coefficients are set to zero.

STAT_PERFECT_FIT—A perfect fit was obtained with a polynomial of degree less than *degree*. High order coefficients are set to zero.

Fatal Errors

STAT_NONNEG_WEIGHT_REQUEST_2—All weights must be nonnegative.

STAT_ALL_OBSERVATIONS_MISSING—Each (x, y) point contains NaN. There are no valid data.

STAT_CONSTANT_XVALUES—The x values are constant.

Version History

6.4	Introduced

IMSL_POLYPREDICT

The IMSL_POLYPREDICT function computes predicted values, confidence intervals, and diagnostics after fitting a polynomial regression model.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_POLYPREDICT(predict_info, x
    [, CI_PTW_NEW_SAMP=variable] [, CI_PTW_POP_MEAN=variable]
    [, CI_SCHEFFE=variable] [, CONFIDENCE=value] [, COOKS_D=variable]
    [, DEL_RESIDUAL=variable] [, DFFITS=variable] [, /DOUBLE]
    [, LEVERAGE=variable] [, RESIDUAL=variable]
    [, STD_RESIDUAL=variable] [, WEIGHTS=array] [, Y=array])
```

Return Value

One-dimensional array containing the predicted values.

Arguments

predict_info

One-dimensional byte array containing information computed by IMSL_POLYREGRESS and returned through keyword *Predict_Info*. The data contained in this array is in an encrypted format and should not be altered after it is returned by IMSL_POLYREGRESS.

x

One-dimensional array containing the values of the independent variable for which calculations are to be performed.

Keywords

CI_PTW_NEW_SAMP

Named variable into which the two-dimensional array of size 2 by N_ELEMENTS(*x*) containing the confidence intervals for two-sided prediction intervals, corresponding to the elements of *x*, is stored. Element *Ci_Ptw_New_Samp*(0, *i*) contains the *i*-th lower confidence limit, *Ci_Ptw_New_Samp*(1, *i*) contains the *i*-th upper confidence limit.

CI_PTW_POP_MEAN

Named variable into which the two-dimensional array of size 2 by N_ELEMENTS(*x*) containing the confidence intervals for two-sided interval estimates of the means, corresponding to the elements of *x*, is stored. Element *Ci_Ptw_Pop_Mean*(0, *i*) contains the *i*-th lower confidence limit, *Ci_Ptw_Pop_Mean*(1, *i*) contains the *i*-th upper confidence limit.

CI_SCHEFFE

Named variable into which the two-dimensional array of size 2 by N_ELEMENTS(*x*) containing the Scheffé confidence intervals, corresponding to the rows of *x*, is stored. Element *Ci_Scheffe*(0, *i*) contains the *i*-th lower confidence limit; *Ci_Scheffe*(1, *i*) contains the *i*-th upper confidence limit.

CONFIDENCE

Confidence level for both two-sided interval estimates on the mean and for two-sided prediction intervals, in percent. Keyword *Confidence* must be in the range (0.0, 100.0). For one-sided intervals with confidence level, where $50.0 \leq c < 100.0$, set $Confidence = 100.0 - 2.0 * (100.0 - c)$. Default: *Confidence* = 95.0

COOKS_D

Named variable into which the one-dimensional array of length N_ELEMENTS(*x*) containing the Cook's *D* statistics is stored.

Note

You must specify *Y* when using this keyword

DEL_RESIDUAL

Named variable into which the one-dimensional array of length N_ELEMENTS(*x*) containing the deleted residuals is stored.

Note

You must specify Y when using this keyword

DFFITS

Named variable into which the one-dimensional array of length N_ELEMENTS(*x*) containing the DFFITS statistics is stored.

Note

You must specify Y when using this keyword

DOUBLE

If present and nonzero, double precision is used.

LEVERAGE

Named variable into which the one-dimensional array of length N_ELEMENTS(*x*) containing the leverages is stored.

RESIDUAL

Named variable into which the one-dimensional array of length N_ELEMENTS(*x*) containing the residuals is stored.

Note

You must specify Y when using this keyword

STD_RESIDUAL

Named variable into which the one-dimensional array of length N_ELEMENTS(*x*) containing the standardized residuals is stored.

Note

You must specify Y when using this keyword

WEIGHTS

One-dimensional array containing the weight for each element of x . The computed prediction interval uses $SSE/(DFE * \text{Weights}(i))$ for the estimated variance of a future response. Default: $\text{Weights} (*) = 1$

Y

Array of length $N\_ELEMENTS(x)$ containing the observed responses.

Discussion

The `IMSL_POLYPREDICT` function assumes a polynomial model

$$y_i = \beta_0 + \beta_1 x_i + \dots, \beta_k x_i^k + \epsilon_i \quad i = 1, 2, \dots, n$$

where the observed values of the y_i 's constitute the response, the x_i 's are the settings of the independent variable, the β_j 's are the regression coefficients, and the ϵ_i 's are the errors that are independently distributed normal with mean zero and the following variance:

$$\sigma^2/w_i$$

Given the results of a polynomial regression, fitted using orthogonal polynomials and weights w_i , `IMSL_POLYPREDICT` produces predicted values, residuals, confidence intervals, prediction intervals, and diagnostics for outliers and in influential cases.

Often, a predicted value and confidence interval are desired for a setting of the independent variable not used in computing the regression fit. This is accomplished by simply using a different x matrix than was used for the fit when calling `IMSL_POLYPREDICT` (`IMSL_POLYREGRESS`, 649).

Results from `IMSL_POLYREGRESS`, which produces the fit using orthogonal polynomials, are used for input by the array `predict_info`. The fitted model from `IMSL_POLYREGRESS` is:

$$\hat{y}_i = \hat{\alpha}_0 p_0(z_i) + \hat{\alpha}_1 p_1(z_i) + \dots + \hat{\alpha}_k p_k(z_i)$$

where the z_i 's are settings of the independent variable x scaled to the interval $[-2, 2]$ and the $p_j(z)$'s are the orthogonal polynomials. The $X^T X$ matrix for this model is a diagonal matrix with elements d_j . The case statistics are easily computed from this model and are equal to those from the original polynomial model with β_j 's as the regression coefficients.

The leverage is computed as follows:

The estimated variance of:

$$h_i = w_i \sum_{j=0}^k d_j^{-1} p_j^2(z_i)$$

$$\hat{y}_i$$

is given by the following:

$$\frac{h_i s^2}{w_i}$$

The computation of the remainder of the case statistics follow easily from their definitions. See the chapter introduction for the definition of the case diagnostics.

Often, predicted values and confidence intervals are desired for combinations of settings of the independent variables not used in computing the regression fit. This can be accomplished by defining a new data matrix. Since the information about the model fit is input in *predict_info*, it is not necessary to send in the data set used for the original calculation of the fit, i.e., only variable combinations for which predictions are desired need be entered in *x*.

Examples

Example 1

A polynomial model is fit to data using the “[IMSL_POLYREGRESS](#)” on page 649), then [IMSL_POLYPREDICT](#) is used to compute predicted values. The results are shown in [Figure 14-4](#).

```
x = [0, 0, 1, 1, 2, 2, 4, 4, 5, 5, 6, 6, 7, 7]
y = [58, 48, 58, 57, 61, 67, 70, 74, 77, 72, 81, 85, 84, 81]
; Define the sample data set.
degree = 3
Coefs = IMSL_POLYREGRESS(x, y, degree, $
    Predict_Info = predict_info)
x2 = 8 * FINDGEN((100)/99)
; Call IMSL_POLYREGRESS using keyword Predict_Info.
predicted = IMSL_POLYPREDICT(predict_info, x2)
; Call IMSL_POLYPREDICT with Predict_Info.
PLOT, x, y, Psym = 4
; Plot the results.
OPLOT, x2, predicted
```

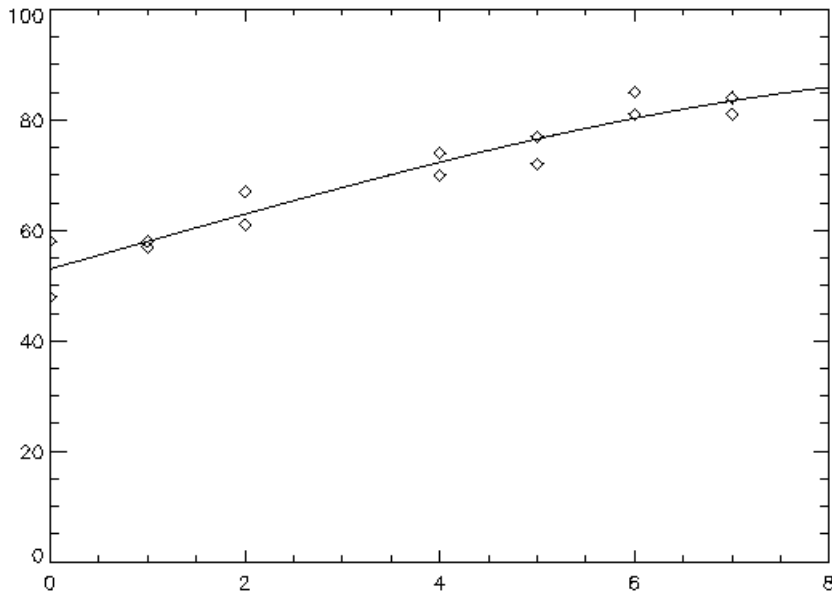


Figure 14-4: Original and Predicted Values Plot

Example 2

A polynomial model is fit to the data discussed by Neter and Wasserman (1974, pp. 279-285). The data set contains the response variable y measuring coffee sales (in hundreds of gallons) and the number of self-service dispensers. Responses for 14 similar cafeterias are in the data set. First, a procedure is defined to print the ANOVA table. The results are shown in Figure 14-5.

```
.RUN
PRO print_results, anova_table
; Define some labels for the anova table.
labels = ['df for among groups      ', $
          'df for within groups      ', $
          'total (corrected) df      ', $
          'ss for among groups        ', $
          'ss for within groups        ', $
          'total (corrected) ss       ', $
          'mean square among groups    ', $
          'mean square within groups   ', $
          'F-statistic                 ', $
          'P-value                     ', $
```

```

'R-squared (in percent)          ', $
'adjusted R-squared (in percent)', $
'est. std of within group error ', $
'overall mean of y              ', $
'coef. of variation (in percent)']
PRINT, '          * * Analysis of Variance * *'
; Print the analysis of variance table.
FOR i = 0, 13 DO PRINT, labels(i), $
    anova_table(i), FORMAT = '(a32,f10.2)'
END

x = [0, 0, 1, 1, 2, 2, 4, 4, 5, 5, 6, 6, 7, 7]
y = [508.1, 498.4, 568.2, 577.3, 651.7, $
    657.0, 755.3, 758.9, 787.6, 792.1, $
    841.4, 831.8, 854.7, 871.4]
degree = 2
coefs = IMSL_POLYREGRESS(x, y, degree, $
    Anova_Table = anova_table, predict_info = predict_info)
; Call IMSL_POLYREGRESS to compute the fit.
predicted = IMSL_POLYPREDICT(predict_info, x, $
    Ci_Scheffe = ci_scheffe, Y = y, Dffits = dffits)
; Call IMSL_POLYPREDICT.
PLOT, x, ci_scheffe(1, *), Yrange = [450, 900], Linestyle = 2
; Plot the results; confidence bands are dashed lines.
OPLOT, x, ci_scheffe(0, *), Linestyle = 2
OPLOT, x, y, Psym = 4
x2 = 7 * FINDGEN(100)/99
OPLOT, x2, IMSL_POLYPREDICT(predict_info, x2)
print_results, anova_table

; Print the ANOVA table.
* * Analysis of Variance * *
df for among groups          2.00
df for within groups         11.00
total (corrected) df         13.00
ss for among groups          225031.94
ss for within groups          710.55
total (corrected) ss         225742.48
mean square among groups     112515.97
mean square within groups     64.60
F-statistic                   1741.86
P-value                       0.00
R-squared (in percent)       99.69
adjusted R-squared (in percent) 99.63
est. std of within group error 8.04
overall mean of y            710.99
coef. of variation (in percent) 1.13

```

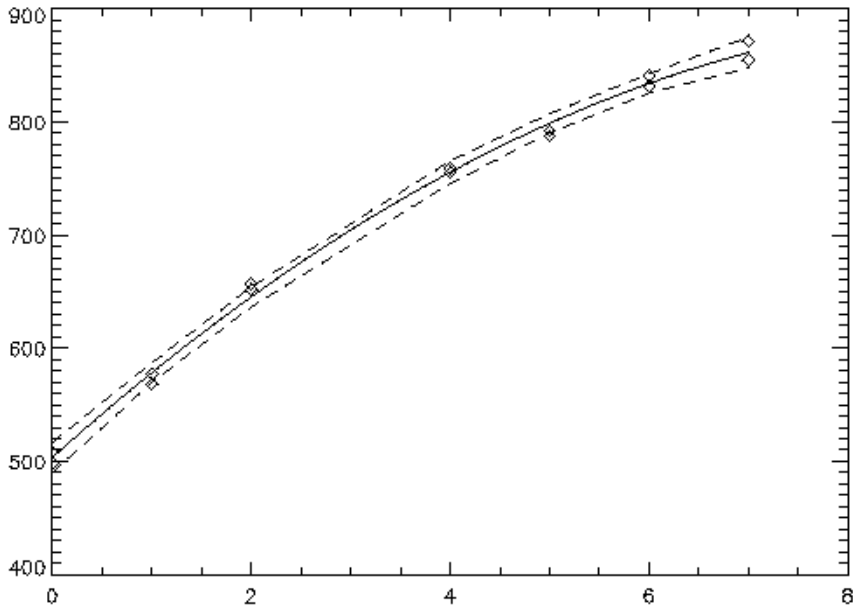


Figure 14-5: Predicted Values with Confidence Bands Plot

Errors

Warning Errors

- STAT_LEVERAGE_GT_1—Leverage (= #) much greater than 1 is computed. It is set to 1.0.
- STAT_DEL_MSE_LT_0—Deleted residual mean square (= #) much less than zero is computed. It is set to zero.

Fatal Errors

- STAT_NEG_WEIGHT—Keyword *Weights*(#) = #. Weights must be nonnegative.

Version History

6.4	Introduced

IMSL_NONLINREGRESS

The IMSL_NONLINREGRESS function fits a nonlinear regression model.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_NONLINREGRESS(fcn, n_parameters, x, y
    [, ABS_EPS_SSE=value] [, DF=variable] [, /DOUBLE] [, JACOBIAN=string]
    [, GRAD_EPS=value] [, ITMAX=value] [, MAX_JAC_EVALS=value]
    [, MAX_SSE_EVALS=value] [, MAX_STEP=value] [, N_DIGIT=value]
    [, PREDICTED=variable] [, R_MATRIX=variable] [, R_RANK=variable]
    [, REL_EPS_SSE=value] [, RESIDUAL=variable] [, STEP_EPS=value]
    [, SSE=variable] [, THETA_GUESS=array] [, THETA_SCALE=array]
    [, TOLERANCE=value] [, TRUST_REGION=value])
```

Return Value

One-dimensional array of length *n_parameters* containing solution:

$$\hat{\theta}$$

for the nonlinear regression coefficients.

Arguments

fcn

Scalar string specifying the name of a user-supplied function to evaluate the function that defines the nonlinear regression problem. Function *fcn* accepts the following input parameters and returns a scalar float:

- **x**—One-dimensional array containing the point at which point the function is evaluated.
- **theta**—One-dimensional array containing the current values of the regression coefficients. Function *fcn* returns a predicted value at the point *x*. In the

following, $f(x_i; \theta)$, or just f_i , denotes the value of this function at the point x_i , for a given value of θ . (Both x_i and θ are arrays.)

n_parameters

Number of parameters to be estimated.

x

Two-dimensional array containing the matrix of independent (explanatory) variables.

y

One-dimensional array of length N_ELEMENTS ($x(*, 0)$) containing the dependent (response) variable.

Keywords

ABS_EPS_SSE

Absolute SSE function tolerance. Default: $Abs\_Eps\_Sse = \max(10^{-20}, \epsilon^2), \max(10^{-40}, \epsilon^2)$ in double, where ϵ is the machine precision

DF

Named variable into which the degrees of freedom is stored.

DOUBLE

If present and nonzero, double precision is used.

JACOBIAN

Scalar string specifying the name of a user-supplied function to compute the i -th row of the Jacobian. This function accepts the following parameters:

- **X**—One-dimensional array of length N_ELEMENTS ($x(0, *)$) containing the data values corresponding to the i -th row.
- **Theta**—One-dimensional array of length $n\_parameters$ containing the regression coefficients for which the Jacobian is evaluated. The return value of this function is an array of length $n\_parameters$ containing the computed $n\_parameters$ row of the Jacobian for observation i at $Theta$. Note that each derivative $\partial f(x_i) / \partial \theta_j$ should be returned in element $(j - 1)$ of the returned array for $j = 1, 2, \dots, n\_parameters$.

GRAD_EPS

Scaled gradient tolerance. The j -th component of the scaled gradient at θ is calculated as:

$$\frac{|g_j| * \max(|\theta_j|, 1/t_j)}{\frac{1}{2} \|F(\theta)\|_2^2}$$

where $g = \nabla F(\theta)$, $t = \text{Theta\_Scale}$, and

$$\|F(\theta)\|_2^2 = \sum_{i=1}^n (y_i - f(x_i; \theta))^2$$

The value $F(\theta)$ is the vector of the residuals at the point θ . Default:

$$\text{Grad\_Eps} = \sqrt{\epsilon} \text{ } (\sqrt[3]{\epsilon} \text{ in double}),$$

where ϵ is the machine precision.

ITMAX

Maximum number of iterations. Default: *Itmax* = 100

MAX_JAC_EVALS

Maximum number of Jacobian evaluations. Default: *Max Jac Evals* = 400

MAX_SSE_EVALS

Maximum number of SSE function evaluations. Default: *Max Sse Evals* = 400

MAX_STEP

Maximum allowable step size. Default: *Max_Step* = $1000 \max(\epsilon_1, \epsilon_2)$, where $\epsilon_1 = (t^T \theta_0)^{1/2}$, $\epsilon_2 = \|t\|_2$, $t = \text{Theta\_Scale}$, and $\theta_0 = \text{Theta\_Guess}$

N_DIGIT

Number of good digits in the function. Default: machine dependent

PREDICTED

Named variable into which the one-dimensional array, containing the predicted values at the approximate solution, is stored.

R_MATRIX

Named variable into which the two-dimensional array of size $n\_parameters \times n\_parameters$, containing the R matrix from a QR decomposition of the Jacobian, is stored.

R_RANK

Named variable into which the rank of the R matrix is stored. A rank of less than $n\_parameters$ may indicate the model is overparameterized.

RESIDUAL

Named variable into which the one-dimensional array, containing the residuals at the approximate solution, is stored.

STEP_EPS

Scaled step tolerance. The j -th component of the scaled step from points θ and θ' is computed as:

$$\frac{|\theta_j - \theta'_j|}{\max(|\theta_j|, 1/t_j)}$$

where $t = Theta\_Scale$. Default: $Step\_Eps = \epsilon^{2/3}$, where ϵ is machine precision

SSE

Named variable into which the residual sum of squares is stored.

REL_EPS_SSE

Relative SSE function tolerance. Default: $Rel\_Eps\_Sse = \max(10^{-10}, \epsilon^{2/3}), \max(10^{-20}, \epsilon^{2/3})$ in double, where ϵ is the machine precision

THETA_GUESS

Array with $n\_parameters$ components containing an initial guess. Default: $Theta\_Guess(*) = 0$

THETA_SCALE

One-dimensional array of length $n\_parameters$ containing the scaling array for θ . Keyword $Theta\_Scale$ is used mainly in scaling the gradient and the distance between

two points. See keywords *Grad_Eps* and *Step_Eps* for more details. Default:
Theta_Scale(*) = 1

TOLERANCE

False convergence tolerance. Default: *Tolerance* = 100 * ϵ , where ϵ is machine precision.

TRUST_REGION

Size of initial trust region radius. The default is based on the initial scaled Cauchy step.

Discussion

The IMSL_NONLINREGRESS function fits a nonlinear regression model using least squares. The nonlinear regression model is

$$y_i = f(x_i; \theta) + \epsilon_i \quad i = 1, 2, \dots, n$$

where the observed values of the y_i 's constitute the responses or values of the dependent variable, the known x_i 's are the vectors of the values of the independent (explanatory) variables, θ is the vector of p regression parameters, and the ϵ_i 's are independently distributed normal errors with mean zero and variance σ^2 . For this model, a least-squares estimate of θ is also a maximum likelihood estimate of θ .

The residuals for the model are as follows:

$$e_i(\theta) = y_i - f(x_i; \theta) \quad i = 1, 2, \dots, n$$

A value of θ that minimizes:

$$\sum_{i=1}^n [e_i(\theta)]^2$$

is a least-squares estimate of θ . IMSL_NONLINREGRESS is designed so that the values of the function $f(x_i; \theta)$ are computed one at a time by a user-supplied function.

The IMSL_NONLINREGRESS function is based on MINPACK routines LMDIF and LMDER by Moré *et al.* (1980) that use a modified Levenberg-Marquardt method to generate a sequence of approximations to a minimum point. Let:

$$\hat{\theta}_c$$

be the current estimate of θ . A new estimate is given by:

$$\hat{\theta}_c + s_c$$

where s_c is a solution to the following:

$$(J(\hat{\theta}_c)^T J(\hat{\theta}_c) + \mu_c I) s_c = J(\hat{\theta}_c)^T e(\hat{\theta}_c)$$

Here:

$$J(\hat{\theta}_c)$$

is the Jacobian evaluated at:

$$\hat{\theta}_c$$

The algorithm uses a “trust region” approach with a step bound of δ_c . A solution is first obtained for $\mu_c = 0$. If:

$$\|s_c\|_2 < \delta_c$$

this update is accepted; otherwise, μ_c is set to a positive value and another solution is obtained. The method is discussed by Levenberg (1944), Marquardt (1963), and Dennis and Schnabel (1983, pp. 129–147, 218–338).

If a user-supplied function is specified in *Jacobian*, the Jacobian is computed analytically; otherwise, forward finite differences are used to estimate the Jacobian numerically. In the latter case, especially if single precision is used, the estimate of the Jacobian may be so poor that the algorithm terminates at a noncritical point. In such instances, you should either supply a Jacobian function, use the *Double* keyword, or do both.

Programming Notes

Nonlinear regression allows substantial flexibility over linear regression because you can specify the functional form of the model. This added flexibility can cause unexpected convergence problems for users who are unaware of the limitations of the software. Also, in many cases, there are possible remedies that may not be immediately obvious. The following is a list of possible convergence problems and some remedies. There is no one-to-one correspondence between the problems and the remedies. Remedies for some problems also may be relevant for other problems.

- A local minimum is found. Try a different starting value. Good starting values often can be obtained by fitting simpler models. For example, for a nonlinear function:

$$f(x; \theta) = \theta_1 e^{\theta_2 x}$$

- good starting values can be obtained from the estimated linear regression coefficients:

$$\hat{\beta}_0 \text{ and } \hat{\beta}_1$$

- from a simple linear regression of $\ln y$ on x . The starting values for the nonlinear regression in this case would be:

$$\theta_1 = e^{\hat{\beta}_0} \text{ and } \theta_2 = \hat{\beta}_1$$

- If an approximate linear model is not clear, then simplify the model by reducing the number of nonlinear regression parameters. For example, some nonlinear parameters for which good starting values are known could be set to these values in order to simplify the model for computing starting values for the remaining parameters.
- The estimate of θ is incorrectly returned as the same or very close to the initial estimate. This occurs often because of poor scaling of the problem, which might result in the residual sum of squares being either very large or very small relative to the precision of the computer. The keywords allow control of the scaling.
- The model is discontinuous as a function of θ . (The function $f(x;\theta)$ can be a discontinuous function of x .)
- Overflow occurs during the computations. Make sure the supplied functions do not overflow at some value of θ .
- The estimate of θ is going to infinity. A parameterization of the problem in terms of reciprocals may help.
- Some components of θ are outside known bounds. This can sometimes be handled by making a function that produces artificially large residuals outside of the bounds (even though this introduces a discontinuity in the model function).

Examples

Example 1

In this example (Draper and Smith 1981, p. 518), the following nonlinear model is fit:

$$Y = \alpha + (0.49 - \alpha)e^{-\beta(X-8)} + \varepsilon$$

```
.RUN
FUNCTION fcn, x, theta
  RETURN, theta(0) + (0.49 - theta(0)) $
    *EXP(theta(1)*(x(0) - 8))
END

x = [10, 20, 30, 40]
y = [0.48, 0.42, 0.40, 0.39]
n_parameters = 2
theta_hat = IMSL_NONLINREGRESS('fcn', n_parameters, x, y)
PRINT, 'Estimated Coefficients:', theta_hat
```

Example 2

Consider the nonlinear regression model and data set discussed by Neter *et al.* (1983, pp. 475–478):

$$y_i = \theta_1 e^{\theta_2 x_i} + \varepsilon_i$$

There are two parameters and one independent variable. The data set considered consists of 15 observations. The results are shown in [Figure 14-6](#).

```
.RUN
FUNCTION fcn, x, theta
    ; Define function that defines nonlinear regression problem.
    RETURN, theta(0) * EXP(x(0) * theta(1))
END

.RUN
FUNCTION jac, x, theta
    ; Define the Jacobian function.
    fjac = theta
    ; The following assignment produces array of correct size to
    ; use as the return value of the Jacobian.
    fjac[0] = exp(theta[1] * x[0])
    fjac[1] = theta[0] * x[0] * EXP(theta[1] * x[0])
    RETURN, fjac
    ; Compute the Jacobian.
END

.RUN
PRO nlnreg_ex
    ; Define x and y.
    x = [2, 5, 7, 10, 14, 19, 26, 31, 34, 38, 45, 52, 53, 60, 65]
    y = [54, 50, 45, 37, 35, 25, 20, 16, 18, 13, 8, 11, 8, 4, 6]
    theta_hat = IMSL_NONLINREGRESS('fcn', 2, x, y, $
        Theta_Guess = [60, -0.03], $
        Grad_Eps = 0.001, Jacobian = 'jac')
    PLOT, x, y, Psym = 4, Title = 'Nonlinear Regression'
    ; Plot original data.
    xtmp = 80 * FINDGEN(200)/199
    OPLLOT, xtmp, theta_hat(0) * EXP(xtmp * theta_hat(1))
    ; Plot regression.
END
```

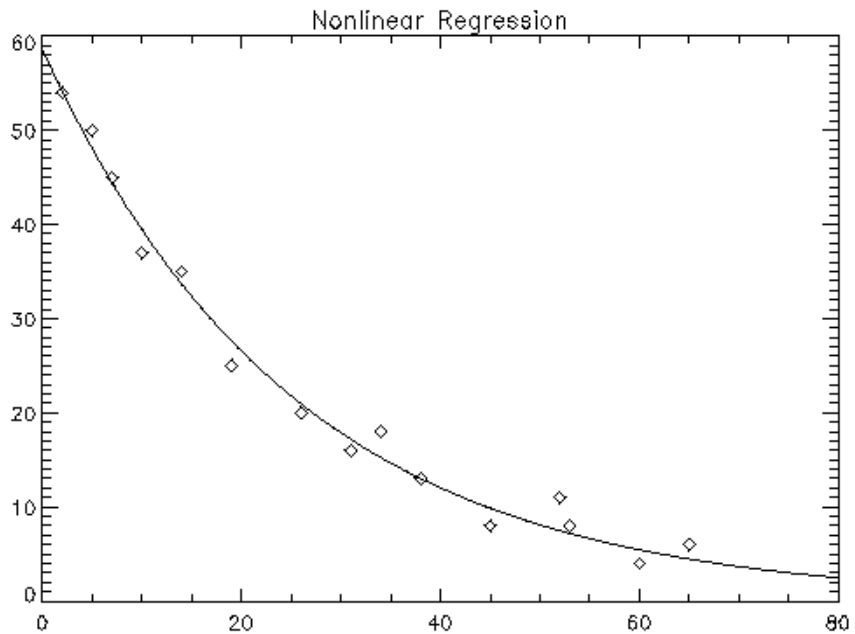


Figure 14-6: Original Data and Nonlinear Regression Fit Plot

Errors

Informational Errors

`STAT_STEP_TOLERANCE`—Scaled step tolerance satisfied. The current point may be an approximate local solution, but it is also possible that the algorithm is making very slow progress and is not near a solution or that *Step_Eps* is too big.

Warning Errors

`STAT_LITTLE_FCN_CHANGE`—Both actual and predicted relative reductions in the function are less than or equal to the relative function tolerance.

`STAT_TOO_MANY_ITN`—Maximum number of iterations exceeded.

`STAT_TOO_MANY_FCN_EVAL`—Maximum number of function evaluations exceeded.

`STAT_TOO_MANY_JACOBIAN_EVAL`—Maximum number of Jacobian evaluations exceeded.

STAT_UNBOUNDED—Five consecutive steps have been taken with the maximum step length.

STAT_FALSE_CONVERGENCE—Iterates appear to be converging to noncritical point.

Version History

6.4	Introduced

IMSL_HYPOTH_PARTIAL

The IMSL_HYPOTH_PARTIAL function constructs an equivalent completely testable multivariate general linear hypothesis $H\beta U = G$ from a partially testable hypothesis $H_p\beta U = G_p$.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_HYPOTH_PARTIAL(info_v, hp [, /DOUBLE]
    [, G_MATRIX=variable] [, GP=array] [, H_MATRIX=variable]
    [, RANK_HP=variable])
```

Return Value

Number of rows in the completely testable hypothesis, nh . This value is also the degrees of freedom for the hypothesis. The value nh classifies the hypothesis $H_p\beta U = G_p$ as nontestable ($nh = 0$), partially testable ($0 < nh < Rank\_Hp$) or completely testable ($0 < nh = Rank\_Hp$), where $Rank\_Hp$ is the rank of H_p (see keyword *Rank_Hp*).

Arguments

hp

The H_p array of size nhp by $n\_coefficients$ with each row corresponding to a row in the hypothesis and containing the constants that specify a linear combination of the regression coefficients. Here, $n\_coefficients$ is the number of coefficients in the fitted regression model.

info_v

One-dimensional array of type BYTE containing information about the regression fit. See IMSL_MULTIREGRESS.

Keywords

DOUBLE

If present and nonzero, double precision is used.

G_MATRIX

Named variable into which a one-dimensional array of length nu containing the G matrix is stored. The elements of G_Matrix contain the null hypothesis values for the completely testable hypothesis.

GP

Two-dimensional array of size nhp by nu containing the G_p matrix, the null hypothesis values. By default, each value of G_p is equal to 0.

H_MATRIX

Named variable into which a two-dimensional array of size nh by $n\_parameters$ containing the H matrix is stored. Each row of H_Matrix corresponds to a row in the completely testable hypothesis and contains the constants that specify an estimable linear combination of the regression coefficients.

RANK_HP

Named variable into which the rank of H_p is stored.

Discussion

Once a general linear model $y = X\beta + \epsilon$ is fitted, particular hypothesis tests are frequently of interest. If the matrix of regressors X is not full rank (as evidenced by the fact that some diagonal elements of the R matrix output from the fit are equal to zero), methods that use the results of the fitted model to compute the hypothesis sum of squares (see “[IMSL_HYPOTH_SCPH](#)” on page 681) require specification in the hypothesis of only linear combinations of the regression parameters that are estimable. A linear combination of regression parameters $c^T\beta$ is *estimable* if there exists some vector a such that $c^T = a^TX$, i.e., c^T is in the space spanned by the rows of X . For a further discussion of estimable functions, see Maindonald (1984, pp. 1661168) and Searle (1971, pp. 1802188). The `IMSL_HYPOTH_PARTIAL` function is only useful in the case of non-full rank regression models, i.e., when the problem of estimability arises.

Peixoto (1986) noted that the customary definition of testable hypothesis in the context of a general linear hypothesis test $H\beta = g$ is overly restrictive. He extended the notion of a testable hypothesis (a hypothesis composed of estimable functions of the regression parameters) to include partially testable and completely testable hypothesis. A hypothesis $H\beta = g$ is *partially testable* if the intersection of the row space H (denoted by $\mathfrak{R}(H)$) and the row space of X ($\mathfrak{R}(X)$) is not essentially empty and is a proper subset of $\mathfrak{R}(H)$, i.e., $\{0\} \subset \mathfrak{R}(H) \cap \mathfrak{R}(X) \subset \mathfrak{R}(H)$. A hypothesis $H\beta = g$ is *completely testable* if $\{0\} \subset \mathfrak{R}(H) \cap \mathfrak{R}(X) \subset \mathfrak{R}(X)$. Peixoto also demonstrated a method for converting a partially testable hypothesis to one that is completely testable so that the usual method for obtaining sums of squares for the hypothesis from the results of the fitted model can be used. The method replaces H_p in the partially testable hypothesis $H_p\beta = g_p$ by a matrix H whose rows are a basis for the intersection of the row space of H_p and the row space of X . A corresponding conversion of the null hypothesis values from g_p to g is also made. A sum of squares for the completely testable hypothesis can then be computed (see IMSL_HYPOTH_SCPH). The sum of squares that is computed for the hypothesis $H\beta = g$ equals the difference in the error sums of squares from two fitted models—the restricted model with the partially testable hypothesis $H_p\beta = g_p$ and the unrestricted model.

For the general case of the multivariate model $Y = X\beta + \varepsilon$ with possible linear equality restrictions on the regression parameters, IMSL_HYPOTH_PARTIAL converts the partially testable hypothesis $H_p\beta = g_p$ to a completely testable hypothesis $H\beta U = G$. For the case of the linear model with linear equality restrictions, the definitions of the estimable functions, nontestable hypothesis, partially testable hypothesis, and completely testable hypothesis are similar to those previously given for the unrestricted model with the exception that $\mathfrak{R}(X)$ is replaced by $\mathfrak{R}(R)$ where R is the upper triangular matrix based on the linear equality restrictions. The nonzero rows of R form a basis for the rowspace of the matrix $(X^T, A^T)^T$. The rows of H form an orthonormal basis for the intersection of two subspaces—the subspace spanned by the rows of H_p and the subspace spanned by the rows of R . The algorithm used for computing the intersection of these two subspaces is based on an algorithm for computing angles between linear subspaces due to Björk and Golub (1973). (See also Golub and Van Loan 1983, pp. 429430). The method is closely related to a canonical correlation analysis discussed by Kennedy and Gentle (1980, pp. 561565). The algorithm is as follows:

1. Compute a QR factorization of:

$$H_p^T$$

with column permutations so that

$$H_p^T = Q_1 R_1 P_1^T$$

Here, P_1 is the associated permutation matrix that is also an orthogonal matrix. Determine the rank of H_p as the number of nonzero diagonal elements of R_1 , for example n_1 . Partition $Q_1 = (Q_{11}, Q_{12})$ so that Q_{11} is the first n_1 column of Q_1 . Set $\text{Rank}_{Hp} = n$.

2. Compute a QR factorization of the transpose of the R matrix (input through *info_v*) with column permutations so that:

$$R^T = Q_2 R_2 P_2^T$$

Determine the rank of R from the number of nonzero diagonal elements of R , for example n_2 . Partition $Q_2 = (Q_{21}, Q_{22})$ so that Q_{21} is the first n_2 columns of Q_2 .

3. Form:

$$A = Q_{11}^T Q_{21}$$

4. Compute the singular values of A :

$$\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_{\min(n_1, n_2)}$$

and the left singular vectors W of the singular value decomposition of A so that:

$$W^T A V = (\sigma_1, \dots, \sigma_{\min(n_1, n_2)})$$

If $\sigma_1 < 1$, then the dimension of the intersection of the two subspaces is $s = 0$. Otherwise, assume the dimension of the intersection to be s if $\sigma_s = 1 > \sigma_{s+1}$. Set $nh = s$.

5. Let W_1 be the first s columns of W . Set $H = (Q_1 W_1)^T$.
6. Assume R_{11} to be a nhp by nhp matrix related to R_1 as follows: If $nhp < n\_parameters$, R_{11} equals the first nhp rows of R_1 . Otherwise, R_{11} contains R_1 in its first $n\_parameters$ rows and zeros in the remaining rows. Compute a solution Z to the linear system:

$$R_{11}^T Z = P_1^T G_p$$

If this linear system is declared inconsistent, an error message with error code equal to 2 is issued.

7. Partition
so that Z_1 is the first n_1 rows of Z . Set:

$$Z^T = \begin{pmatrix} Z_1^T & Z_2^T \end{pmatrix}$$

$$G = W_1^T Z_1$$

The degrees of freedom (nh) classify the hypothesis $H_p \beta U = G_p$ as nontestable ($nh = 0$), partially testable ($0 < nh < \text{Rank\_Hp}$), or completely testable ($0 < nh = \text{Rank\_Hp}$).

For further details concerning the algorithm, see Sallas and Lioni (1988).

Example

A one-way analysis-of-variance model discussed by Peixoto (1986) is fitted to data. The model is:

$$y_{ij} = \mu + \alpha_i + \varepsilon_{ij}, (i, j) = (1, 1) (2, 1) (2, 2)$$

The model is fitted using the “[IMSL_MULTIREGRESS](#)” on page 607. The partially testable hypothesis:

$$H_0: \begin{matrix} \alpha_1=5 \\ \alpha_2=3 \end{matrix}$$

is converted to a completely testable hypothesis.

```
nrows = 3
n_indep = 1
n_dep = 1
n_param = 3
n_class = 1
n_cont = 0
nhp = 2
z = [1, 2, 2]
y = [17.3, 24.1, 26.3]
gp = [5, 3]
hp = TRANSPOSE([[0, 1, 0], [0, 0, 1]])
x = IMSL_REGRESSORS(z, n_class, n_cont)
size_x = SIZE(x)
nreg = size_x(2)
coefs = IMSL_MULTIREGRESS(x, y, Predict_Info = info_v)
% IMSL_MULTIREGRESS: Warning: STAT_RANK_DEFICIENT
The model is not full rank. There is not a unique least
squares solution. The rank of the matrix of regressors is 2.
nh = IMSL_HYPOTH_PARTIAL(info_v, hp, Gp = gp, $
    G_Matrix = g_matrix, H_Matrix = h_matrix, Rank_Hp = rank_hp)
IF (nh EQ 0) THEN PRINT, 'Nontestable Hypothesis' $
ELSE IF (nh LT rank_hp) THEN $
```

```
PRINT, 'Partially Testable Hypothesis' $
ELSE PRINT, 'Completely Testable Hypothesis'
Partially Testable Hypothesis
PM, h_matrix, title = 'H Matrix'
H Matrix
      0.00000      0.707107      -0.707107
PM, g_matrix, title = 'G'
G
      1.41421
```

Errors

Warning Errors

STAT_HYP_NOT_CONSISTENT—The hypothesis is inconsistent within the computed tolerance.

Version History

6.4	Introduced

IMSL_HYPOTH_SCPH

The IMSL_HYPOTH_SCPH function computes the matrix of sums of squares and crossproducts for the multivariate general linear hypothesis $H\beta U = G$ given the regression fit.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_HYPOTH_SCPH(info_v, h [, DFH=variable] [, /DOUBLE]
    [, G=array] [, U=array])
```

Return Value

Two-dimensional array, *scph*, containing the sums of squares and crossproducts attributable to the hypothesis.

Arguments

info_v

One-dimensional array of type BYTE containing information about the regression fit. See IMSL_MULTIREGRESS.

h

Two-dimensional array of size nh by $n\_coefficients$ with each row corresponding to a row in the hypothesis and containing the constants that specify a linear combination of the regression coefficients. Here, $n\_coefficients$ is the number of coefficients in the fitted regression model.

Keywords

DFH

Named variable into which the degrees of freedom for the sums of squares and crossproducts matrix is stored. This is equal to the rank of input matrix h .

DOUBLE

If present and nonzero, double precision is used.

G

Two-dimensional array of size nh by nu containing the G matrix, the null hypothesis values. By default, each value of G is equal to 0.

U

Two-dimensional array of size $n\_dependent$ by nu containing the U matrix for the test $H_p\beta U = G_p$ where nu is the number of linear combinations of the dependent variables to be considered. The value nu must be greater than 0 and less than or equal to $n\_dependent$. Default: $nu = n\_dependent$ and U is the identity matrix

Discussion

The `IMSL_HYPOTH_SCPH` function computes the matrix of sums of squares and crossproducts for the general linear hypothesis $H\beta U = G$ for the multivariate general linear model $Y = X\beta + \epsilon$.

The rows of H must be linear combinations of the rows of R , i.e., $H\beta = G$ must be completely testable. If the hypothesis is not completely testable, the [“IMSL_HYPOTH_PARTIAL” on page 675](#) can be used to construct an equivalent completely testable hypothesis.

Computations are based on an algorithm discussed by Kennedy and Gentle (1980, p. 317) that is extended by Sallas and Lioni (1988) for multivariate non-full rank models with possible linear equality restrictions. The algorithm is as follows:

1. Form

$$W = H\hat{\beta}U - G$$
2. Find C as the solution of $R^T C = H^T$. If the equations are declared inconsistent within a computed tolerance, a warning error message is issued that the hypothesis is not completely testable.
3. For all rows of R corresponding to restrictions, i.e., containing negative diagonal elements from a restricted least-squares fit, zero out the corresponding rows of C , i.e., from DC .

4. Decompose DC with Householder transformations and column pivoting for a square, upper triangular matrix T with diagonal elements of nonincreasing magnitude and permutation matrix P such that:

$$DCP = Q \begin{bmatrix} T \\ 0 \end{bmatrix}$$

where Q is an orthogonal matrix.

5. Determine the rank of T , say r . If $t_{11} = 0$, then $r = 0$. Otherwise, the rank of T is r if:

$$|t_{rr}| > |t_{11}| \varepsilon \quad |t_{r+1, r+1}|$$

where $\varepsilon = 10.0 * (\text{machine epsilon})$.

Then, zero out all rows of T below r . Set the degrees of freedom for the hypothesis, Dfh , to r .

6. Find V as a solution to $T^T V = P^T W$. If the equations are inconsistent, a warning error message is issued that the hypothesis is inconsistent within a computed tolerance, i.e., the linear system:

$$H\beta = U = G$$

$$Ab = Z$$

does not have a solution for β .

Form $V^T V$, which is the required matrix of sum of squares and crossproducts, *scph*.

In general, the two warning errors described above are serious user errors that require you to correct the hypothesis before any meaningful sums of squares from this function can be computed. However, in some cases, You may know the hypothesis is consistent and completely testable, but the checks in `IMSL_HYPOTH_SCPH` are too tight. For this reason, `IMSL_HYPOTH_SCPH` continues with the calculations.

`IMSL_HYPOTH_SCPH` gives a matrix of sums of squares and crossproducts that could also be obtained from separate fittings of the two models:

$$Y^\# = X\beta^\# + \varepsilon^\# \quad (1)$$

$$A\beta^\# = Z^\#$$

$$H\beta^\# = G$$

and:

$$Y^\# = X\beta^\# + \varepsilon^\# \quad (2)$$

$$A\beta^\# = Z^\#$$

where $Y^{\#} = YU$, $\beta^{\#} = \beta U$, $\varepsilon^{\#} = \varepsilon U$, and $Z^{\#} = ZU$. The error sum of squares and crossproducts matrix for (1) minus that for (2) is the matrix sum of squares and crossproducts output in *scph*. Note that this approach avoids the question of testability.

Example

The data for this example are from Maindonald (1984, pp. 203204). A multivariate regression model containing two dependent variables and three independent variables is fit using `IMSL_MULTIREGRESS` and the results stored in the structure *info_v*. The sum of squares and crossproducts matrix, *scph*, is then computed by calling `IMSL_HYPOTH_SCPH` for the test that the third independent variable is in the model (determined by the specification of *h*). The degrees of freedom for *scph* also is computed.

```
x = TRANSPOSE([[7.0, 5.0, 6.0], [2.0, -1.0, 6.0], $
               [7.0, 3.0, 5.0], [-3.0, 1.0, 4.0], [2.0, -1.0, 0.0], $
               [2.0, 1.0, 7.0], [-3.0, -1.0, 3.0], [2.0, 1.0, 1.0], $
               [2.0, 1.0, 4.0]])
y = TRANSPOSE([[7.0, 1.0], [-5.0, 4.0], [6.0, 10.0], $
               [5.0, 5.0], [5.0, -2.0], [-2.0, 4.0], [0.0, -6.0], $
               [8.0, 2.0], [3.0, 0.0]])
h = FLTARR(1, 4)
h(*) = 0
h(0, 3) = 1.0
coefs = IMSL_MULTIREGRESS(x, y, Predict_Info = p)
scph = IMSL_HYPOTH_SCPH(p, h, Dfh = dfh)
PRINT, 'Degrees of Freedom Hypothesis =', dfh
Degrees of Freedom Hypothesis =      1.00000
PM, scph, Title = 'Sum of Squares and Crossproducts'
Sum of Squares and Crossproducts
      100.000      -40.0000
     -40.0000      16.0000
```

Errors

Warning Errors

`STAT_HYP_NOT_TESTABLE`—The hypothesis is not completely testable within the computed tolerance. Each row of “h” must be a linear combination of the rows of “r”.

`STAT_HYP_NOT_CONSISTENT`—The hypothesis is inconsistent within the computed tolerance.

Version History

6.4	Introduced

IMSL_HYPOTH_TEST

The IMSL_HYPOTH_TEST function performs tests for a multivariate general linear hypothesis $H\beta U = G$ given the hypothesis sums of squares and crossproducts matrix S_H .

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_HYPOTH_TEST(info_v, dfh, scph [, /DOUBLE]
    [, HOTELLING_TRACE=variable] [, PILLAI_TRACE=variable]
    [, ROY_MAX_ROOT=variable] [, U=array] [, WILK_LAMBDA=variable])
```

Return Value

The p -value corresponding to Wilks' lambda test.

Arguments

dfh

Degrees of freedom for the sums of squares and crossproducts matrix.

info_v

One-dimensional array of type BYTE containing information about the regression fit. See IMSL_MULTIREGRESS.

scph

Two-dimensional array of size nu by nu containing S_H , the sums of squares and crossproducts attributable to the hypothesis.

Keywords

DOUBLE

If present and nonzero, double precision is used.

HOTELLING_TRACE

Named variable into which the one-dimensional array containing the Hotelling's trace and p -value is stored.

PILLAI_TRACE

Named variable into which the one-dimensional array containing the Pillai's trace and p -value is stored.

ROY_MAX_ROOT

Named variable into which the one-dimensional array containing the Roy's maximum root criterion and p -value is stored.

U

Two-dimensional array of size $n\_dependent$ by nu containing the U matrix for the test $H_p: \beta U = G_p$ where nu is the number of linear combinations of the dependent variables to be considered. The value nu must be greater than 0 and less than or equal to $n\_dependent$. Default: $nu = n\_dependent$ and U is the identity matrix

WILK_LAMBDA

Named variable into which the one-dimensional array containing the Wilk's lambda and p -value is stored.

Discussion

IMSL_HYPOTH_TEST computes test statistics and p -values for the general linear hypothesis $H\beta U = G$ for the multivariate general linear model.

The hypothesis sum of squares and crossproducts matrix input in *scph* is:

$$S_H = (H\hat{\beta}U - G)^T (C^T D C)^{-1} (H\hat{\beta}U - G)$$

where C is a solution to $R^T C = H$ and where D is a diagonal matrix with diagonal elements:

See the section [Linear Dependence and the R Matrix](#).

$$d_{ii} = \begin{cases} 1 & \text{if } r_{ii} > 0 \\ 0 & \text{otherwise} \end{cases}$$

Error sum of squares and crossproducts matrix for model $Y = X\beta + \epsilon$ is:

$$(Y - X\hat{\beta})^T (Y - X\hat{\beta})$$

which is input in IMSL_MULTIREGRESS. The error sum of squares and crossproducts matrix for the hypothesis $H\beta U = G$ computed by

IMSL_HYPOTH_TEST is:

$$S_E = U^T (Y - X\hat{\beta})^T (Y - X\hat{\beta}) U$$

Let p equal the order of the matrices S_E and S_H , i.e.:

$$p = \begin{cases} NU & \text{if } NU > 0 \\ NDEP & \text{otherwise} \end{cases}$$

Let q (stored in *dffh*) be the degrees of freedom for the hypothesis. Let v (input in *info_v*) be the degrees of freedom for error. The IMSL_HYPOTH_TEST function computed three test statistics based on eigenvalues λ_i ($i = 1, 2, \dots, p$) of the generalized eigenvalue problem $S_H x = \lambda S_E x$. These test statistics are as follows:

Wilk's lambda:

$$\Lambda = \frac{\det(S_E)}{\det(S_H + S_E)} = \prod_{i=1}^p \frac{1}{1 + \lambda_i}$$

The associated p -value is based on an approximation discussed by Rao (1973, p. 556). The statistic:

$$F = \frac{ms - pq / 2 + 1}{pq} \frac{1 - \Lambda^{1/s}}{\Lambda^{1/s}}$$

has an approximate F distribution with pq and $ms - pq/2 + 1$ numerator and denominator degrees of freedom, respectively, where:

$$s = \begin{cases} 1 & \text{if } p = 1 \text{ or } q = 1 \\ \sqrt{\frac{p^2 q^2 - 4}{p^2 + q^2 - 5}} & \text{otherwise} \end{cases}$$

and:

$$m = v - \frac{(p + q - 1)}{2}$$

The F test is exact if $\min(p, q) \leq 2$ (Kshirsagar, 1972, Theorem 4, p. 2994300).

Roy's maximum root:

$$c = \max \lambda_i \text{ over all } i$$

where c is output as `value = Roy_Max_Root(0)`. The p -value is based on the approximation:

$$F = \frac{v + q - s}{s} c$$

where $s = \max(p, q)$ has an approximate F distribution with s and $v + q - s$ numerator and denominator degrees of freedom, respectively. The F test is exact if $s = 1$; the p -value is also exact. In general, the value output in `p_value = Roy_Max_Root(1)` is lower bound on the actual p -value.

Hotelling's trace:

$$U = \text{tr}(HE^{-1}) = \sum_{i=1}^p \lambda_i$$

U is output as `value = Hotelling_Trace(0)`. The p -value is based on the approximation of McKeon (1974) that supersedes the approximation of Hughes and Saw (1972). McKeon's approximation is also discussed by Seber (1984, p. 39). For:

$$b = 4 + \frac{pq + 2}{\frac{(v + q - p - 1)(v - 1)}{(v - p - 3)(v - p)}}$$

the p -value is based on the result that:

$$F = \frac{b(v - p - 1)}{(b - 2)pq}$$

has an approximate F distribution with pq and b degrees of freedom. The test is exact if $\min(p, q) = 1$. For $v \leq p + 1$, the approximation is not valid, and `p_value = Hotelling_Trace(1)` is set to NaN.

These three test statistics are valid when S_E is positive definite. A necessary condition for S_E to be positive definite is $v \geq p$. If S_E is not positive definite, a warning error message is issued, and both `value` and `p_value` are set to NaN.

Because the requirement $v \geq p$ can be a serious drawback, IMSL_HYPOTH_TEST computes a fourth test statistic based on eigenvalues θ_i ($i = 1, 2, \dots, p$) of the generalized eigenvalue problem $S_H w = \theta(S_H + S_E) w$. This test statistic requires a less restrictive assumption— $S_H + S_E$ is positive definite. A necessary condition for $S_H + S_E$ to be positive definite is $v + q \geq p$. If S_E is positive definite, IMSL_HYPOTH_TEST avoids the computation of the generalized eigenvalue problem from scratch. In this case, the eigenvalues θ_i are obtained from λ_i by:

$$\theta_i = \frac{\lambda_i}{1 + \lambda_i}$$

The fourth test statistic is as follows:

Pillai's trace:

$$V = \text{tr}[S_H(S_H + S_E)^{-1}] = \sum_{i=1}^p \theta_i$$

V is output as `value = Pillai_Trace(0)`. The p -value is based on an approximation discussed by Pillai (1985). The statistic:

$$F = \frac{2n + s + 1}{2m + s + 1} \frac{V}{s - V}$$

has an approximate F distribution with $s(2m + s + 1)$ and $s(2n + s + 1)$ numerator and denominator degrees of freedom, respectively, where:

$$s = \min(p, q)$$

$$m = 1/2(|p - q| - 1)$$

$$n = 1/2(v - p - 1)$$

The F test is exact if $\min(p, q) = 1$.

Examples

Example 1

The data for this example are from Maindonald (1984, p. 20310204). A multivariate regression model containing two dependent variables and three independent variables is fit using IMSL_MULTIREGRESS and the results stored in `info_v`. The sum of squares and crossproducts matrix, `scph`, is then computed using HYPOYH_SCPH for the test that the third independent variable is in the model (determined by specification of h). Finally, IMSL_HYPOTH_TEST is used to compute the p -value for the test statistic (Wilk's lambda).

```

x = TRANSPOSE([[7.0, 5.0, 6.0], [2.0, -1.0, 6.0], $
               [7.0, 3.0, 5.0], [-3.0, 1.0, 4.0], [2.0, -1.0, 0.0], $
               [2.0, 1.0, 7.0], [-3.0, -1.0, 3.0], [2.0, 1.0, 1.0], $
               [2.0, 1.0, 4.0]])
y = TRANSPOSE([[7.0, 1.0], [-5.0, 4.0], [6.0, 10.0], $
               [5.0, 5.0], [5.0, -2.0], [-2.0, 4.0], [0.0, -6.0], $
               [8.0, 2.0], [3.0, 0.0]])
h = FLTARR(1, 4)
h(*) = 0
h(0, 3) = 1.0
coefs = IMSL_MULTIREGRESS(x, y, Predict_Info = p)
scph = IMSL_HYPOTH_SCPH(p, h, Dfh = dfh)
pvalue = IMSL_HYPOTH_TEST(p, dfh, scph)
PM, pvalue, format = '(F10.6)', Title = 'P-value'
P-value
      0.000010

```

Example 2

This example is the same as the first example, but more statistics are computed. Also, the U matrix, U , is explicitly specified as the identity matrix (which is the same default configuration of U).

```

x = TRANSPOSE([[7.0, 5.0, 6.0], [2.0, -1.0, 6.0], $
               [7.0, 3.0, 5.0], [-3.0, 1.0, 4.0], [2.0, -1.0, 0.0], $
               [2.0, 1.0, 7.0], [-3.0, -1.0, 3.0], [2.0, 1.0, 1.0], $
               [2.0, 1.0, 4.0]])
y = TRANSPOSE([[7.0, 1.0], [-5.0, 4.0], [6.0, 10.0], $
               [5.0, 5.0], [5.0, -2.0], [-2.0, 4.0], [0.0, -6.0], $
               [8.0, 2.0], [3.0, 0.0]])
h = FLTARR(1, 4)
h(*) = 0
h(0, 3) = 1.0
u = [[1, 0], [0, 1]]
coefs = IMSL_MULTIREGRESS(x, y, Predict_Info = p)
scph = IMSL_HYPOTH_SCPH(p, h, Dfh = dfh)
pvalue = IMSL_HYPOTH_TEST(p, dfh, scph, U = u, $
                          Wilk_Lambda = wilk_lambda, Roy_Max_Root = roy_max_root, $
                          Hotelling_Trace = hotelling_trace, $
                          Pillai_Trace = pillai_trace)
PRINT, 'Wilk value = ', wilk_lambda(0), ' p-value =', $
      wilk_lambda(1)
Wilk value = 0.00314861 p-value = 9.89437e-06
PRINT, 'Roy value = ', roy_max_root(0), ' p-value =', $
      roy_max_root(1)
Roy value = 316.601 p-value = 9.89437e-06
PRINT, 'Hotelling value = ', hotelling_trace(0), ' p-value =', $
      hotelling_trace(1)
Hotelling value = 316.601 p-value = 9.89437e-06

```



```
PRINT, 'Pillai value = ', pillai_trace(0), ' p-value =', $
      pillai_trace(1)
Pillai value = 0.996851 p-value = 9.89437e-06
```

Errors

Warning Errors

STAT_SINGULAR_1—“u”*“scpe”*“u” is singular. Only Pillai’s trace can be computed. Other statistics are set to NaN.

Fatal Errors

STAT_NO_STAT_1—“scpe” + “scph” is singular. No tests can be computed.

STAT_NO_STAT_2—No statistics can be computed. Iterations for eigenvalues for the generalized eigenvalue problem “scph”* $x = (\text{lambda}) * (“\text{scph}” + “\text{scpe}”) * x$ failed to converge.

STAT_NO_STAT_3—No statistics can be computed. Iterations for eigenvalues for the generalized eigenvalue problem “scph”* $x = (\text{lambda}) * (“\text{scph}” + “\text{u}” * “\text{scpe}” * “\text{u}”) * x$ failed to converge.

STAT_SINGULAR_2—“u”*“scpe”*“u” + “scph” is singular. No tests can be computed.

STAT_SINGULAR_TRI_MATRIX—The input triangular matrix is singular. The index of the first zero diagonal element is equal to #.

Version History

6.4	Introduced

IMSL_NONLINOPT

The IMSL_NONLINOPT function fits data to a nonlinear model (possibly with linear constraints) using the successive quadratic programming algorithm (applied to the sum of squared errors, $SSE = \sum (y_i - f(x_i; \theta))^2$) and either a finite difference gradient or a user-supplied gradient.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_NONLINOPT(f, n_parameters, x, y [, A_MATRIX=array]
[, ACC=value] [, ACTIVE_CONST=variable] [, B=array] [, /DOUBLE]
[, FREQUENCIES=array] [, JACOBIAN=string]
[, LAGRANGE_MULT=variable] [, MAX_SSE_EVALS=value] [, MEQ=value]
[, NUM_ACTIVE=variable] [, PREDICTED=variable] [, RESIDUAL=variable]
[, SSE=variable] [, STOP_INFO=variable] [, THETA_GUESS=array]
[, WEIGHTS=array] [, XLB=array] [, XUB=array])
```

Return Value

One-dimensional array of length $n\_parameters$ containing solution:

$$\hat{\theta}$$

for the nonlinear regression coefficients.

Arguments

f

Scalar string specifying a user-supplied function that defines the nonlinear regression problem at a given point. Function f has the following parameters:

- **xi**—One-dimensional array of length $n\_independent$ at which point the function is evaluated.
- **theta**—One-dimensional array of length $n\_parameters$ containing the current values of the regression coefficients.

Function f returns a predicted value at the point x_i . In the following, $f(x_i; \theta)$, or just f_i , denotes the value of this function at the point x_i , for a given value of θ . (Both x_i and θ are arrays.).

n_parameters

Number of parameters to be estimated.

x

Two-dimensional array of size $n\_observations$ by $n\_independent$ containing the matrix of independent (explanatory) variables where $n\_observations$ is the number of observations and $n\_independent$ is the number of independent variables.

y

One-dimensional array of length $n\_observations$ containing the dependent (response) variable.

Keywords

A_MATRIX

Two-dimensional array of size $n\_constraints$ by $n\_parameters$ containing the equality constraint gradients in the first Meq rows, followed by the inequality constraint gradients. Here $n\_constraints$ is the total number of linear constraints (excluding simple bounds). A_Matrix and B must be used together. Default: There are no default linear constraints.

ACC

The nonnegative tolerance on the first order conditions at the calculated solution.

ACTIVE_CONST

Named variable into which a one-dimensional array of length Num_Active containing the indices of the final active constraints is stored.

B

One-dimensional array of length $n\_constraints$ containing the right-hand sides of the linear constraints. Keywords A_Matrix and B must be used together. Default: There are no default linear constraints.

A_Matrix and B are the linear constraints, specifically, the constraints on θ are:

$$a_{i1} \theta_1 + \dots + a_{ij} \theta_j = b_i$$

for $i = 1, n\_equality$ and $j = 1, n\_parameter$, and:

$$a_{k1} \theta_1 + \dots + a_{kj} \theta_j \leq b_k$$

for $k = n\_equality + 1, n\_constraints$ and $j = 1, n\_parameter$.

DOUBLE

If present and nonzero, double precision is used.

FREQUENCIES

One-dimensional array of length $n\_observations$ containing the frequency for each observation. Default: $Frequencies(*) = 1$

JACOBIAN

Scalar string specifying a user-supplied function to compute the i -th row of the Jacobian. The function specified by *Jacobian* has the following parameters:

- ***Xi***—One-dimensional array containing the $n\_independent$ data values corresponding to the i -th row. (Input)
- ***Theta***—One-dimensional array of length $n\_parameters$ containing the regression coefficients for which the Jacobian is evaluated. (Input)

The return value of this function is a one-dimensional array containing the computed $n\_parameters$ row of the Jacobian for observation i at *Theta*. Note that each derivative $f(x_i)/\theta$ should be returned in element $(j - 1)$ of the returned array for $j = 1, 2, \dots, n\_parameters$. Further note that in order to maintain consistency with the other nonlinear solver, IMSL_NONLINREGRESS, the Jacobian values must be specified as the *negative* of the calculated derivatives.

LAGRANGE_MULT

Named variable into which a one-dimensional array of length *Num_Active* containing the Lagrange multiplier estimates of the final active constraints is stored.

MAX_SSE_EVALS

The maximum number of SSE evaluations allowed. Default: $Max\_Sse\_Eval = 400$

MEQ

Number of the *A_Matrix* constraints which are *equality* constraints; the remaining (*n_constraints* – *Meq*) constraints are *inequality* constraints. Default: *Meq* = 0.

NUM_ACTIVE

Named variable into which the final number of active constraints is stored.

PREDICTED

Named variable into which a one-dimensional array of length *n_observations* containing the predicted values at the approximate solution is stored.

RESIDUAL

Named variable into which a one-dimensional array of length *n_observations* containing the residuals at the approximate solution is stored.

SSE

Named variable into which the residual sum of squares is stored.

STOP_INFO

Named variable into which one of the following integer values to indicate the reason for leaving the routine is stored:

Stop_info	Reason for leaving routine
1	θ is feasible, and the condition that depends on <i>Acc</i> is satisfied.
2	θ is feasible, and rounding errors are preventing further progress.
3	θ is feasible, but <i>sse</i> fails to decrease although a decrease is predicted by the current gradient vector.
4	The calculation cannot begin because <i>A_Matrix</i> contains fewer than <i>n_constraints</i> constraints or because the lower bound on a variable is greater than the upper bound.

Table 14-9: Stop_Info Integer Values

Stop_info	Reason for leaving routine
5	The equality constraints are inconsistent. These constraints include any components of $\hat{\theta}$ that are frozen by setting $Xlb(i)$ equal to $Xub(i)$.
6	The equality constraints and the bound on the variables are found to be inconsistent.
7	There is no possible θ that satisfies all of the constraints.
8	Maximum number of sse evaluations (<i>Max_Sse_Eval</i>) is exceeded.
9	θ is determined by the equality constraints.

Table 14-9: Stop_Info Integer Values

THETA_GUESS

One-dimensional array with $n\_parameters$ components containing an initial guess.
Default: $Theta\_Guess(*) = 0$

WEIGHTS

One-dimensional array of length $n\_observations$ containing the weight for each observation. Default: $Weights(*) = 1$

XLB

One-dimensional array of length $n\_parameters$ containing the lower bounds on the parameters; choose a very large negative value if a component should be unbounded below or set $Xlb(i) = Xub(i)$ to freeze the i -th variable. Default: All parameters are bounded below by -10^6 .

XUB

One-dimensional array of length $n\_parameters$ containing the upper bounds on the parameters; choose a very large value if a component should be unbounded above or set $Xlb(i) = Xub(i)$ to freeze the i -th variable. Default: All parameters are bounded above by 10^6 .

Discussion

The IMSL_NONLINOPT function is based on M.J.D. Powell's TOLMIN, which solves linearly constrained optimization problems, i.e., problems of the form $\min f(q)$, $\theta \in \mathfrak{R}$, subject to:

$$A_1 \theta = b_1$$

$$A \leq b_2$$

$$\theta_l \leq \theta \leq \theta_u$$

given the vectors b_1 , b_2 , θ_l , and θ_u and the matrices A_1 and A_2 .

The algorithm starts by checking the equality constraints for inconsistency and redundancy. If the equality constraints are consistent, the method will revise θ^0 , the initial guess you provided, to satisfy:

$$A_1 \theta = b_1$$

Next, θ^0 is adjusted to satisfy the simple bounds and inequality constraints. This is done by solving a sequence of quadratic programming subproblems to minimize the sum of the constraint or bound violations.

Now, for each iteration with a feasible θ^k , let J_k be the set of indices of inequality constraints that have small residuals. Here, the simple bounds are treated as inequality constraints. Let I_k be the set of indices of active constraints. The following quadratic programming problem:

$$\min f(\theta^k) + d^T \nabla f(\theta^k) + \frac{1}{2} d^T B^k d$$

subject to:

$$a_j d = 0 \quad j \in I_k$$

$$a_j d \leq 0 \quad j \in J_k$$

is solved to get (d^k, λ^k) where a_j is a row vector representing either a constraint in A_1 or A_2 or a bound constraint on θ . In the latter case, the $a_j = e_i$ for the bound constraint $\theta_i \leq (\theta_u)_i$ and $a_j = -e_i$ for the constraint $\theta_i \geq (\theta_l)_i$. Here, e_i is a vector with a 1 as the i -th component, and zeroes elsewhere. λ^k are the Lagrange multipliers, and B^k is a positive definite approximation to the second derivative $\nabla^2 f(\theta^k)$.

After the search direction d^k is obtained, a line search is performed to locate a better point. The new point $\theta^{k+1} = \theta^k + \alpha^k d^k$ has to satisfy the conditions:

$$f(\theta^k + \alpha^k d^k) \leq f(\theta^k) + 0.1 \alpha^k (d^k)^T \nabla f(\theta^k)$$

and:

$$(d^k)^T \nabla f(\theta^k + \alpha^k d^k) \geq 0.7 (d^k)^T \nabla f(\theta^k)$$

The main idea in forming the set J_k is that, if any of the inequality constraints restricts the step-length α^k , then its index is not in J_k . Therefore, small steps are likely to be avoided.

Finally, the second derivative approximation, B^k , is updated by the BFGS formula, if the condition:

$$(\mathbf{d}^k)^T \nabla f(\boldsymbol{\theta}^k + \alpha^k \mathbf{d}^k) - \nabla f(\boldsymbol{\theta}^k) > 0$$

holds. Let $\boldsymbol{\theta}^k \leftarrow \boldsymbol{\theta}^{k+1}$, and start another iteration.

The iteration repeats until the stopping criterion:

$$\|\nabla f(\boldsymbol{\theta}^k) - \mathbf{A}^k \boldsymbol{\lambda}^k\|_2 \leq \tau$$

is satisfied; here, τ is a user-supplied tolerance. For more details, see Powell (1988, 1989).

Since a finite-difference method is used to estimate the gradient, for some single precision calculations. An inaccurate estimate of the gradient may cause the algorithm to terminate at a noncritical point. In such cases, high precision arithmetic is recommended. Also, whenever the exact gradient can be easily provided, the gradient should be passed to IMSL_NONLINOPT using the optional keyword *Jacobian*.

Examples

Example 1

In this example, a data set is fitted to the nonlinear model function:

$$y_i = \sin(\theta_0 x_i) + \varepsilon_i$$

```
.RUN
FUNCTION fcn, x, theta
    res = SIN(theta(0)*x(0))
    RETURN, res
END

x = [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
y = [0.05, 0.21, 0.67, 0.72, 0.98, 0.94, 1.00, 0.73, 0.44, $
    0.36, 0.02]
n_parameters = 1
theta_hat = IMSL_NONLINOPT('fcn', n_parameters, x, y)
% IMSL_NONLINOPT: Note: STAT_NOTE_3
'theta' is feasible but the objective function fails to
decrease. Using double precision may help.
PRINT, 'Theta Hat = ', theta_hat
Theta Hat =      3.16143
```


Example 2

Draper and Smith (1981, p. 475) state a problem due to Smith and Dubey. [H. Smith and S. D. Dubey (1964), "Some reliability problems in the chemical industry." Industrial Quality Control, 21 (2), 1964, pp. 641-670] A certain product must have 50% available chlorine at the time of manufacture. When it reaches the customer 8 weeks later, the level of available chlorine has dropped to 49%. It was known that the level should stabilize at about 30%. To predict how long the chemical would last at the customer site, samples were analyzed at different times. It was postulated that the following nonlinear model should fit the data:

$$y_i = \theta_0 + (0.49 - \theta_0)e^{-\theta_1(x_i - 8)} + \varepsilon_i$$

Since the chlorine level will stabilize at about 30%, the initial guess for θ_0 is 0.30. Using the last data point ($x = 42$, $y = 0.39$) and $\theta_0 = 0.30$ and the above nonlinear equation, an estimate for θ_1 of 0.02 is obtained.

The constraints that $\theta_0 \geq 0$ and $\theta_1 \geq 0$ are also imposed. These are equivalent to requiring that the level of available chlorine always be positive and never increase with time.

The Jacobian of the nonlinear model equation is also used.

```
.RUN
FUNCTION fcn, x, theta
    res = theta(0) + (0.49-theta(0))* exp(-theta(1)*(x(0) - 8.0))
    RETURN, res
END

.RUN
FUNCTION jacobian, x, theta
    fjac = theta
    fjac(*) = 0
    fjac(0) = -1.0 + exp(-theta(1)*(x(0) - 8.0));
    fjac(1) = (0.49 - theta(0))*(x(0) - 8.0) * $
        exp(-theta(1)*(x(0) - 8.0));
    RETURN, fjac
END

x = [8.0, 8.0, 10.0, 10.0, 10.0, 10.0, 12.0, 12.0, 12.0, $
    12.0, 14.0, 14.0, 14.0, 16.0, 16.0, 16.0, 18.0, 18.0, $
    20.0, 20.0, 20.0, 22.0, 22.0, 22.0, 24.0, 24.0, 24.0, $
    26.0, 26.0, 26.0, 28.0, 28.0, 30.0, 30.0, 30.0, 32.0, $
    32.0, 34.0, 36.0, 36.0, 38.0, 38.0, 40.0, 42.0]
y = [0.49, 0.49, 0.48, 0.48, 0.47, 0.48, 0.47, 0.46, 0.46, 0.45, $
    0.43, 0.45, 0.43, 0.43, 0.44, 0.43, 0.43, 0.43, 0.46, 0.45, $
    0.42, 0.42, 0.43, 0.41, 0.41, 0.40, 0.42, 0.40, 0.40, $
    0.41, 0.40, 0.41, 0.41, 0.40, 0.40, 0.40, 0.38, 0.41, $
```

```
0.40, 0.40, 0.41, 0.38, 0.40, 0.40, 0.39, 0.39]
theta_guess = [0.3, 0.02]
xlb = [0.0, 0.0]
n_parameters = 2
theta_hat = IMSL_NONLINOPT('fcn', n_parameters, x, y, $
    Theta_Guess = theta_guess, Xlb = xlb, $
    Jacobian = 'jacobian', Sse = sse)
PRINT, 'Theta Hat =', theta_hat

Theta Hat =      0.390143      0.101631

PRINT, 'Residual Sum of Squares =', sse

Residual Sum of Squares =      0.00500168
```

Errors

Fatal Errors

- STAT_BAD_CONSTRAINTS_1—The equality constraints are inconsistent.
- STAT_BAD_CONSTRAINTS_2—The equality constraints and the bounds on the variables are found to be inconsistent.
- STAT_BAD_CONSTRAINTS_3—No vector “theta” satisfies all of the constraints. Specifically, the current active constraints prevent any change in “theta” that reduces the sum of constraint violations.
- STAT_BAD_CONSTRAINTS_4—The variables are determined by the equality constraints.
- STAT_TOO_MANY_ITERATIONS_1—Number of function evaluations exceeded “maxfcn” = #.

Version History

6.4	Introduced

IMSL_LNORMREGRESS

The IMSL_LNORMREGRESS function fits a multiple linear regression model using criteria other than least squares. Namely, IMSL_LNORMREGRESS allows you to choose Least Absolute Value (L_1), Least L_p norm (L_p), or Least Maximum Value (Minimax or L_{infinity}) method of multiple linear regression.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_LNORMREGRESS(x, y [, DF=variable] [, /DOUBLE]
    [, EPS=value] [, FREQUENCIES=array] [, ITERS=variable]
    [, /LAV | /LLP | /LMV] [, NMISSING=variable] [, /NO_INTERCEPT]
    [, P=value] [, RANK=variable] [, R_MATRIX=variable]
    [, RESID_MAX=variable] [, RESID_NORM=variable]
    [, RESIDUALS=variable] [, SCALE=variable] [, SEA=variable]
    [, TOLERANCE=value] [, WEIGHTS=array])
```

Return Value

One-dimensional array of length $n_{\text{independent}} + 1$ containing a least absolute value solution for the regression coefficients. The estimated intercept is the initial component of the array, where the i -th component contains the regression coefficients for the i -th dependent variable. If the keyword *No_Intercept* is used then the $(i-1)$ -st component contains the regression coefficients for the i -th dependent variable.

IMSL_LNORMREGRESS returns the L_p norm or least maximum value solution for the regression coefficients when appropriately specified in the input keyword list.

Arguments

x

Two-dimensional array of size n_{rows} by $n_{\text{independent}}$ containing the independent (explanatory) variables(s) where $n_{\text{rows}} = \text{N\_ELEMENTS}(x(*,0))$ and $n_{\text{independent}}$ is the number of independent (explanatory) variables. The i -th column of x contains the i -th independent variable.

y

One-dimensional array of size *n_rows* containing the dependent (response) variable.

Keywords**DF**

Named variable into which the sum of the frequencies minus *Rank* is stored. In least squares fit ($p=2$) *Df* is called the degrees of freedom of error. Keyword *Llp* is required when using keyword *Df*.

DOUBLE

If present and nonzero, double precision is used.

EPS

Convergence criterion. If the maximum relative difference in residuals from the *k*-th to (*k*+1)-st iterations is less than *Eps*, convergence is declared. Keyword *Llp* is required when using keyword *Eps*. Default: $Eps = 100 * (\text{machine epsilon})$.

FREQUENCIES

One-dimensional array of size *n_rows* containing the frequencies for the independent (explanatory) variable. Keyword *Llp* is required when using keyword *Frequencies*.

ITERS

Named variable into which the number of iterations performed is stored.

LAV

By default (or if *Lav* is used) the function fits a multiple linear regression model using the least absolute values criterion. Keywords *Lav*, *Llp*, and *Lmv* can not be used together.

LLP

If present and nonzero, IMSL_LNORMREGRESS fits a multiple linear regression model using the L_p norm criterion. *Llp* requires the keyword *P*, for $P \geq 1$. Keywords *Lav*, *Llp*, and *Lmv* can not be used together.

LMV

If present and nonzero, IMSL_LNORMREGRESS fits a multiple linear regression model using the minimax criterion. Keywords *Lav*, *Llp*, and *Lmv* can not be used together.

NMISSING

Named variable into which the number of rows of data containing NaN (not a number) for the dependent or independent variables is stored. If a row of data contains NaN for any of these variables, that row is excluded from the computations.

NO_INTERCEPT

If present and nonzero, the intercept term:

$$\hat{\beta}_0$$

is omitted from the model and the return value from regression is a one-dimensional array of length *n_independent*. By default the fitted value for observation *i* is:

$$\hat{\beta}_0 + \hat{\beta}_1 x_1 + \dots + \hat{\beta}_k x_k$$

where $k = n\_independent$.

P

The *p* in the L_p norm criterion (see the *Discussion* section for details). *P* must be greater than or equal to one. *P* and *Llp* must be used together.

RANK

Named variable into which the rank of the fitted model is stored.

R_MATRIX

Named variable into which the two-dimensional array containing the upper triangular matrix of dimension (number of coefficients by number of coefficients) containing the R matrix from a QR decomposition of the matrix of regressors is stored. Keyword *Llp* is required when using keyword *R_Matrix*.

RESID_MAX

Named variable into which the magnitude of the largest residual is stored. Keyword *Lmv* is required when using keyword *Resid_Max*.

RESID_NORM

Named variable into which the L_p norm of the residuals is stored. Keyword *Llp* is required when using keyword *Resid_Norm*.

RESIDUALS

Named variable into which the one-dimensional array (of length equal to the number of observations) containing the residuals is stored. Keyword *Llp* is required when using keyword *Residuals*.

SCALE

Named variable into which the square of the scale constant used in an L_p analysis is stored. An estimated asymptotic variance-covariance matrix of the regression coefficients is $Scale * (R^T R)^{-1}$. Keyword *Llp* is required when using keyword *Scale*.

SEA

Named variable into which the sum of the absolute value of the errors is stored. Keyword *Lav* is required when using keyword *Sea*.

TOLERANCE

Tolerance used in determining linear dependence. Keyword *Llp* is required when using keyword *Tolerance*. Default: $Tolerance = 100 * (\text{machine epsilon})$.

WEIGHTS

One-dimensional array of size n_rows containing the weights for the independent (explanatory) variable. Keyword *Llp* is required when using keyword *Weights*.

Discussion

Least Absolute Value Criterion

The `IMSL_LNORMREGRESS` function computes estimates of the regression coefficients in a multiple linear regression model. For keyword *Lav* (default), the criterion satisfied is the minimization of the sum of the absolute values of the deviations of the observed response y_i from the fitted response:

$$\hat{y}_i$$

for a set on n observations. Under this criterion, known as the L_1 or LAV (least absolute value) criterion, the regression coefficient estimates minimize:

$$\sum_{i=0}^{n-1} |y_i - \hat{y}_i|$$

The estimation problem can be posed as a linear programming problem. The special nature of the problem, however, allows for considerable gains in efficiency by the modification of the usual simplex algorithm for linear programming. These modifications are described in detail by Barrodale and Roberts (1973, 1974).

In many cases, the algorithm can be made faster by computing a least-squares solution prior to the use of keyword *Lav*. This is particularly useful when a least-squares solution has already been computed. The procedure is as follows:

1. Fit the model using least squares and compute the residuals from this fit.
2. Fit the residuals from Step 1 on the regressor variables in the model using keyword *Lav*.
3. Add the two estimated regression coefficient vectors from Steps 1 and 2. The result is an L_1 solution.

When multiple solutions exist for a given problem, option *Lav* may yield different estimates of the regression coefficients on different computers, however, the sum of the absolute values of the residuals should be the same (within rounding differences). The informational error indicating nonunique solutions may result from rounding accumulation. Conversely, because of rounding the error may fail to result even when the problem does have multiple solutions.

L_p Norm Criterion

Keyword *Llp* computes estimates of the regression coefficients in a multiple linear regression model $y = X\beta + \varepsilon$ under the criterion of minimizing the L_p norm of the deviations for $i = 0, \dots, n - 1$ of the observed response y_i from the fitted response:

$$\hat{y}_i$$

for a set on n observations and for $p \geq 1$. For the case when keywords *Weights* and *Frequencies* are not supplied, the estimated regression coefficient vector:

$$\hat{\beta}$$

(output in *Result*) minimizes the L_p norm:

$$\left(\sum_{i=0}^{n-1} |y_i - \hat{y}_i|^p \right)^{1/p}$$

The choice $p = 1$ yields the maximum likelihood estimate for β when the errors have a Laplace distribution. The choice $p = 2$ is best for errors that are normally distributed. Sposito (1989, pages 36–40) discusses other reasonable alternatives for p based on the sample kurtosis of the errors.

Weights are useful if errors in the model have known unequal variances:

$$\sigma_i^2$$

In this case, the weights should be taken as:

$$w_i = 1/\sigma_i^2$$

Frequencies are useful if there are repetitions of some observations in the data set. If a single row of data corresponds to n_i observations, set the frequency $f_i = n_i$. In general, keyword *Llp* minimizes the L_p norm:

$$\left(\sum_{i=0}^{n-1} f_i \left| \sqrt{w_i} (y_i - \hat{y}_i) \right|^p \right)^{1/p}$$

The asymptotic variance-covariance matrix of the estimated regression coefficients is given by:

$$\text{asy.var}(\hat{\beta}) = \lambda^2 (R^T R)^{-1}$$

where R is from the QR decomposition of the matrix of regressors (output in keyword *R_Matrix*) and where an estimate of λ^2 is output in keyword *Scale*.

In the discussion that follows, we will first present the algorithm with frequencies and weights all taken to be one. Later, we will present the modifications to handle frequencies and weights different from one.

Keyword *Llp* uses Newton's method with a line search for $p > 1.25$ and, for $p \leq 1.25$, uses a modification due to Ekblom (1973, 1987) in which a series of perturbed problems are solved in order to guarantee convergence and increase the convergence rate. The cutoff value of 1.25 as well as some of the other implementation details given in the remaining discussion were investigated by Sallas (1990) for their effect on CPU times.

For the first iteration in each case, a least-squares solution for regression coefficients is computed with [IMSL_MULTIREGRESS](#). If $p = 2$, the computations are finished. Otherwise, the residuals from the k -th iteration:

$$e_i^{(k)} = y_i - \hat{y}_i^{(k)}$$

are used to compute the gradient and Hessian for the Newton step for the $(k + 1)$ -st iteration for minimizing the p -th power of the L_p norm. (The exponent $1/p$ in the L_p norm can be omitted during the iterations.)

For subsequent iterations, we first discuss the $p > 1.25$ case. For $p > 1.25$, the gradient and Hessian at the $(k + 1)$ -st iteration depend upon:

$$z_i^{(k+1)} = |e_i^{(k)}|^{p-1} \text{sign}(e_i^{(k)})$$

and:

$$v_i^{(k+1)} = |e_i^{(k)}|^{p-2}$$

In the case $1.25 < p < 2$ and:

$$e_i^{(k)} = 0, v_i^{(k+1)}$$

and the Hessian are undefined; and we follow the recommendation of Merle and Spath (1974). Specifically, we modify the definition of:

$$v_i^{(k+1)}$$

to the following:

$$v_i^{(k+1)} = \begin{cases} \tau^{p-2} & \text{if } p < 2 \text{ and } |e_i^{(k)}| < \tau \\ |e_i^{(k)}|^{p-2} & \text{otherwise} \end{cases}$$

where τ equals $100 * \text{machine epsilon}$ times the square root of the residual mean square from the least-squares fit.

Let $V^{(k+1)}$ be a diagonal matrix with diagonal entries:

$$v_i^{(k+1)}$$

and let $z^{(k+1)}$ be a vector with elements:

$$z_i^{(k+1)}$$

In order to compute the step on the $(k + 1)$ -st iteration, the R from the QR decomposition of:

$$[V^{(k+1)}]^{1/2} X$$

is computed using fast Givens transformations. Let:

$$R^{(k+1)}$$

denote the upper triangular matrix from the QR decomposition. The linear system:

$$[R^{(k+1)}]^T R^{(k+1)} d^{(k+1)} = X^T z^{(k+1)}$$

is solved for:

$$d^{(k+1)}$$

where $R^{(k+1)}$ is from the QR decomposition of $[V^{(k+1)}]^{1/2}X$. The step taken on the $(k+1)$ -st iteration is:

$$\hat{\beta}^{(k+1)} = \hat{\beta}^{(k)} + \alpha^{(k+1)} \frac{1}{p-1} d^{(k+1)}$$

The first attempted step on the $(k+1)$ -st iteration is with $\alpha^{(k+1)} = 1$. If all of the:

$$e_i^{(k)}$$

are nonzero, this is exactly the Newton step. See Kennedy and Gentle (1980, pages 528–529) for further discussion.

If the first attempted step does not lead to a decrease of at least one-tenth of the predicted decrease in the p -th power of the L_p norm of the residuals, a backtracking linesearch procedure is used. The backtracking procedure uses a one-dimensional quadratic model to estimate the backtrack constant p . The value of p is constrained to be no less than 0.1. An approximate upper bound for p is 0.5. If after 10 successive backtrack attempts, $\alpha^{(k)} = p_1 p_2 \dots p_{10}$ does not produce a step with a sufficient decrease, then IMSL_LNORMREGRESS issues a message with error code 5. For further details on the backtrack line-search procedure, see Dennis and Schnabel (1983, pages 126–127).

Convergence is declared when the maximum relative change in the residuals from one iteration to the next is less than or equal to Eps . The relative change:

$$\sigma_i^{(k+1)}$$

in the i -th residual from iteration k to iteration $k+1$ is computed as follows:

$$\delta_i^{(k+1)} = \begin{cases} 0 & \text{if } e_i^{(k+1)} = e_i^{(k)} = 0 \\ |e_i^{(k+1)} - e_i^{(k)}| / \max(|e_i^{(k)}|, |e_i^{(k+1)}|, s) & \text{otherwise} \end{cases}$$

where s is the square root of the residual mean square from the least-squares fit on the first iteration.

For the case $1 \leq p \leq 1.25$, we describe the modifications to the previous procedure that incorporate Ekblom's (1973) results. A sequence of perturbed problems are solved with a successively smaller perturbation constant c . On the first iteration, the least-

squares problem is solved. This corresponds to an infinite c . For the second problem, c is taken equal to s , the square root of the residual mean square from the least-squares fit. Then, for the $(j + 1)$ -st problem, the value of c is computed from the previous value of c according to:

$$c_{j+1} = c_j / 10^{5p-4}$$

Each problem is stated as:

$$\text{Minimize } \sum_{i=0}^{n-1} (e_i^2 + c^2)^{p/2}$$

For each problem, the gradient and Hessian on the $(k + 1)$ -st iteration depend upon:

$$z_i^{(k+1)} = e_i^{(k)} r_i^{(k)}$$

and:

$$v_i^{(k+1)} = \left[1 + \frac{(p-2)(e_i^{(k)})^2}{(e_i^{(k)})^2 + c^2} \right] r_i^{(k)}$$

where:

$$r_i^{(k)} = [(e_i^{(k)})^2 + c^2]^{(p-2)/2}$$

The linear system $[R^{(k+1)}]^T R^{(k+1)} d^{(k+1)} = X^T z^{(k+1)}$ is solved for $d^{(k+1)}$ where $R^{(k+1)}$ is from the QR decomposition of $[V^{(k+1)}]^{1/2} X$. The step taken on the $(k + 1)$ -st iteration is:

$$\hat{\beta}^{(k+1)} = \hat{\beta}^{(k)} + \alpha^{(k+1)} d^{(k+1)}$$

where the first attempted step is with $\alpha^{(k+1)} = 1$. If necessary, the backtracking line-search procedure discussed earlier is used.

Convergence for each problem is relaxed somewhat by using a convergence epsilon equal to $\max(Eps, 10^{-j})$ where $j = 1, 2, 3, \dots$ indexes the problems ($j = 0$ corresponds to the least-squares problem).

After the convergence of a problem for a particular c , Ekblom's (1987) extrapolation technique is used to compute the initial estimate of β for the new problem. Let $R^{(k)}$:

$$v_i^{(k)}, e_i^{(k)}$$

and c be from the last iteration of the last problem. Let:

$$t_i = \frac{(p-2)v_i^{(k)}}{(e_i^{(k)})^2 + c^2}$$

and let t be the vector with elements t_i . The initial estimate of β for the new problem with perturbation constant $0.01c$ is:

$$\hat{\beta}^{(0)} = \hat{\beta}^{(k)} + \Delta c d$$

where $\Delta c = (0.01c - c) = -0.99c$, and where d is the solution of the linear system $[R^{(k)}]^T R^{(k)} d = X^T t$.

Convergence of the sequence of problems is declared when the maximum relative difference in residuals from the solution of successive problems is less than Eps .

The preceding discussion was limited to the case for which $Weights(*) = 1$ and $Frequencies(*) = 1$, i.e., the weights and frequencies are all taken equal to one. The necessary modifications to the preceding algorithm to handle weights and frequencies not all equal to one are as follows:

1. Replace:

$$e_i^{(k)} \text{ by } \sqrt{w_i} e_i^{(k)}$$

in the definitions of:

$$z_i^{(k+1)}, v_i^{(k+1)}, \delta_i^{(k+1)}$$

and t_i .

2. Replace:

$$z_i^{(k+1)} \text{ by } f_i \sqrt{w_i} z_i^{(k+1)}, v_i^{(k+1)} \text{ by } f_i w_i v_i^{(k+1)}, \text{ and } t_i^{(k+1)} \text{ by } f_i \sqrt{w_i} t_i^{(k+1)}$$

These replacements have the same effect as multiplying the i -th row of X and y by:

$$\sqrt{w_i}$$

and repeating the row f_i times except for the fact that the residuals returned by `IMSL_LNORMREGRESS` are in terms of the original y and X .

Finally, R and an estimate of λ^2 are computed. Actually, R is recomputed because on output it corresponds to the R from the initial QR decomposition for least squares. The formula for the estimate of λ^2 depends on p .

For $p = 1$, the estimator for λ^2 is given by (McKean and Schrader 1987):

$$\hat{\lambda}^2 = \left[\frac{\sqrt{\text{DFE}}(\tilde{\mathbf{e}}_{(\text{DFE}-k+1)} - \tilde{\mathbf{e}}_{(k)})}{2z_{0.975}} \right]^2$$

with:

$$k = \frac{\text{DFE} + k}{2} - z_{0.975} \sqrt{\frac{\text{DFE}}{4}}$$

where $z_{0.975}$ is the 97.5 percentile of standard normal distribution, and:

$$\tilde{\mathbf{e}}_{(m)} (m = 1, 2, \dots, \text{DFE})$$

are ordered residuals where *Rank* zero residuals are excluded. Note that:

$$\text{DFE} = \sum_{i=0}^{n-1} f_i - \text{irank}$$

For $p = 2$, the estimator of λ^2 is the customary least-squares estimator given by:

$$s^2 = \frac{\text{SSE}}{\text{DFE}} = \frac{\sum_{i=0}^{n-1} f_i w_i (y_i - \hat{y}_i)^2}{\sum_{i=0}^{n-1} f_i - \text{irank}}$$

For $1 < p < 2$ and for $p > 2$, the estimator for λ^2 is given by (Gonin and Money 1989):

$$\hat{\omega}_p^2 = \frac{m_{2p-2}}{[(p-1)m_{p-2}]^2}$$

with:

$$m_r = \frac{\sum_{i=0}^{n-1} f_i \left| \sqrt{w_i} (y_i - \hat{y}_i) \right|^r}{\sum_{i=0}^{n-1} f_i}$$

Least Minimum Value Criterion (minimax)

Keyword *Lmv* computes estimates of the regression coefficients in a multiple linear regression model. The criterion satisfied is the minimization of the maximum deviation of the observed response y_i from the fitted response:

$$\hat{y}_i$$

for a set on n observations. Under this criterion, known as the minimax or LMV (least maximum value) criterion, the regression coefficient estimates minimize:

$$\max_{0 \leq i \leq n-1} |y_i - \hat{y}_i|$$

The estimation problem can be posed as a linear programming problem. A dual simplex algorithm is appropriate, however, the special nature of the problem allows for considerable gains in efficiency by modification of the dual simplex iterations so as to move more rapidly toward the optimal solution. The modifications are described in detail by Barrodale and Phillips (1975).

When multiple solutions exist for a given problem, *Lmv* may yield different estimates of the regression coefficients on different computers, however, the largest residual in absolute value should have the same absolute value (within rounding differences). The informational error indicating nonunique solutions may result from rounding accumulation. Conversely, because of rounding, the error may fail to result even when the problem does have multiple solutions.

Examples

Example 1

A straight line fit to a data set is computed under the LAV criterion.

```
PRO print_results, coefs, rank, sea, iters, nmissing
  PRINT, 'B = ', coefs(0), coefs(1), $
    FORMAT = '(A6, F5.2, 5X, F5.2)'
  PRINT
  PRINT, 'Rank of Regressors Matrix      = ', rank, $
```

```

        FORMAT = '(A32, I3)'
PRINT, 'Sum Absolute Value of Error    = ', sea, $
        FORMAT = '(A32, F7.4)'
PRINT, 'Number of Iterations          = ', iters, $
        FORMAT = '(A32, I3)'
PRINT, 'Number of Rows Missing        = ', nmissing, $
        FORMAT = '(A32, I3)'
END

x = [1.0, 4.0, 2.0, 2.0, 3.0, 3.0, 4.0, 5.0]
y = [1.0, 5.0, 0.0, 2.0, 1.5, 2.5, 2.0, 3.0]
coefs = IMSL_LNORMREGRESS(x, y, Nmissing = nmissing, $
        Rank = rank, Iters = iters, Sea = sea)
print_results, coefs, rank, sea, iters, nmissing
B =      0.50      0.50
Rank of Regressors Matrix    =      2
Sum Absolute Value of Error  =    6.0000
Number of Iterations         =      2
Number of Rows Missing       =      0

```

Example 2

Different straight line fits to a data set are computed under the criterion of minimizing the L_p norm by using p equal to 1, 1.5, 2.0 and 2.5.

```

.RUN
PRO print_results, coefs, residuals, p, resid_norm, rank, df, $
    iters, nmissing, scale, rm
PRINT, 'Coefficients ', coefs, FORMAT = '(A13, 2F7.2)'
PRINT, 'Residuals ', residuals, FORMAT = '(A10, 8F6.2)'
PRINT
PRINT, 'p', p, $
        FORMAT = '(A33, F6.3)'
PRINT, 'Lp norm of the residuals', resid_norm, $
        FORMAT = '(A33, F6.3)'
PRINT, 'Rank of the matrix of regressors ', rank, $
        FORMAT = '(A33, I6)'
PRINT, 'Degrees of freedom error', df, $
        FORMAT = '(A33, F6.3)'
PRINT, 'Number of iterations', iters, $
        FORMAT = '(A33, I6)'
PRINT, 'Number of missing values', nmissing, $
        FORMAT = '(A33, I6)'
PRINT, 'Square of the scale constant', scale, $
        FORMAT = '(A33, F6.3)'
PRINT
PM, rm, FORMAT = '(2F8.3)', Title = '      R matrix'
PRINT
PRINT, '-----'

```

```

PRINT
END

.RUN
x = [1.0, 4.0, 2.0, 2.0, 3.0, 3.0, 4.0, 5.0]
y = [1.0, 5.0, 0.0, 2.0, 1.5, 2.5, 2.0, 3.0]
eps = 0.001
FOR i = 0, 3 DO BEGIN
  p = 1.0 + i*0.5
  coefs = IMSL_LNORMREGRESS(x, y, /Llp, P = p, Eps = eps, $
    Nmissing = nmissing, Rank = rank, $
    Iters = iters, Scale = scale, $
    Df = df, R_Matrix = rm, Residuals = residuals, $
    Resid_Norm = resid_norm)
  print_results, coefs, residuals, p, resid_norm, rank, df, $
    iters, nmissing, scale, rm
ENDFOR
END

```

```

Coefficients      0.50      0.50
Residuals  -0.00   2.50  -1.50   0.50  -0.50   0.50  -0.50   0.00
p                                                    1.000
Lp norm of the residuals                            6.002
Rank of the matrix of regressors                    2
Degrees of freedom error                            6.000
Number of iterations                                8
Number of missing values                            0
Square of the scale constant                        6.248
R matrix
  2.828   8.485
  0.000   3.464

```

```

-----
Coefficients      0.39      0.56
Residuals   0.06   2.39  -1.50   0.50  -0.55   0.45  -0.61  -0.16

p                                                    1.500
Lp norm of the residuals                            3.712
Rank of the matrix of regressors                    2
Degrees of freedom error                            6.000
Number of iterations                                6
Number of missing values                            0
Square of the scale constant                        1.059
R matrix
  2.828   8.485
  0.000   3.464

```

```

-----
Coefficients     -0.12      0.75
Residuals   0.38   2.12  -1.38   0.62  -0.62   0.38  -0.88  -0.62
p                                                    2.000

```



```

Lp norm of the residuals          2.937
Rank of the matrix of regressors    2
Degrees of freedom error          6.000
Number of iterations              1
Number of missing values          0
Square of the scale constant      1.438
R matrix
    2.828    8.485
    0.000    3.464
-----
Coefficients   -0.44    0.87
Residuals      0.57    1.96 -1.30    0.70 -0.67    0.33 -1.04 -0.91
p
Lp norm of the residuals          2.540
Rank of the matrix of regressors    2
Degrees of freedom error          6.000
Number of iterations              4
Number of missing values          0
Square of the scale constant      0.789
R matrix
    2.828    8.485
    0.000    3.464

```

Example 3

A straight line fit to a data set is computed under the LMV criterion.

```

.RUN
PRO print_results, coefs, rank, rm, iters, nmissing
  PRINT, 'B = ', coefs(0), coefs(1), $
    FORMAT = '(A6, F5.2, 5X, F5.2)'
  PRINT
  PRINT, 'Rank of Regressors Matrix      = ', rank, $
    FORMAT = '(A34, I3)'
  PRINT, 'Magnitude of Largest Residual  = ', rm, $
    FORMAT = '(A34, F7.4)'
  PRINT, 'Number of Iterations           = ', iters, $
    FORMAT = '(A34, I3)'
  PRINT, 'Number of Rows Missing         = ', nmissing, $
    FORMAT = '(A34, I3)'
END
x = [0.0, 1.0, 2.0, 3.0, 4.0, 4.0, 5.0]
y = [0.0, 2.5, 2.5, 4.5, 4.5, 6.0, 5.0]
coefs = IMSL_LNORMREGRESS(x, y, /Lmv, Nmissing = nmissing, $
  Rank = rank, Iters = iters, Resid_Max = rm)
print_results, coefs, rank, rm, iters, nmissing

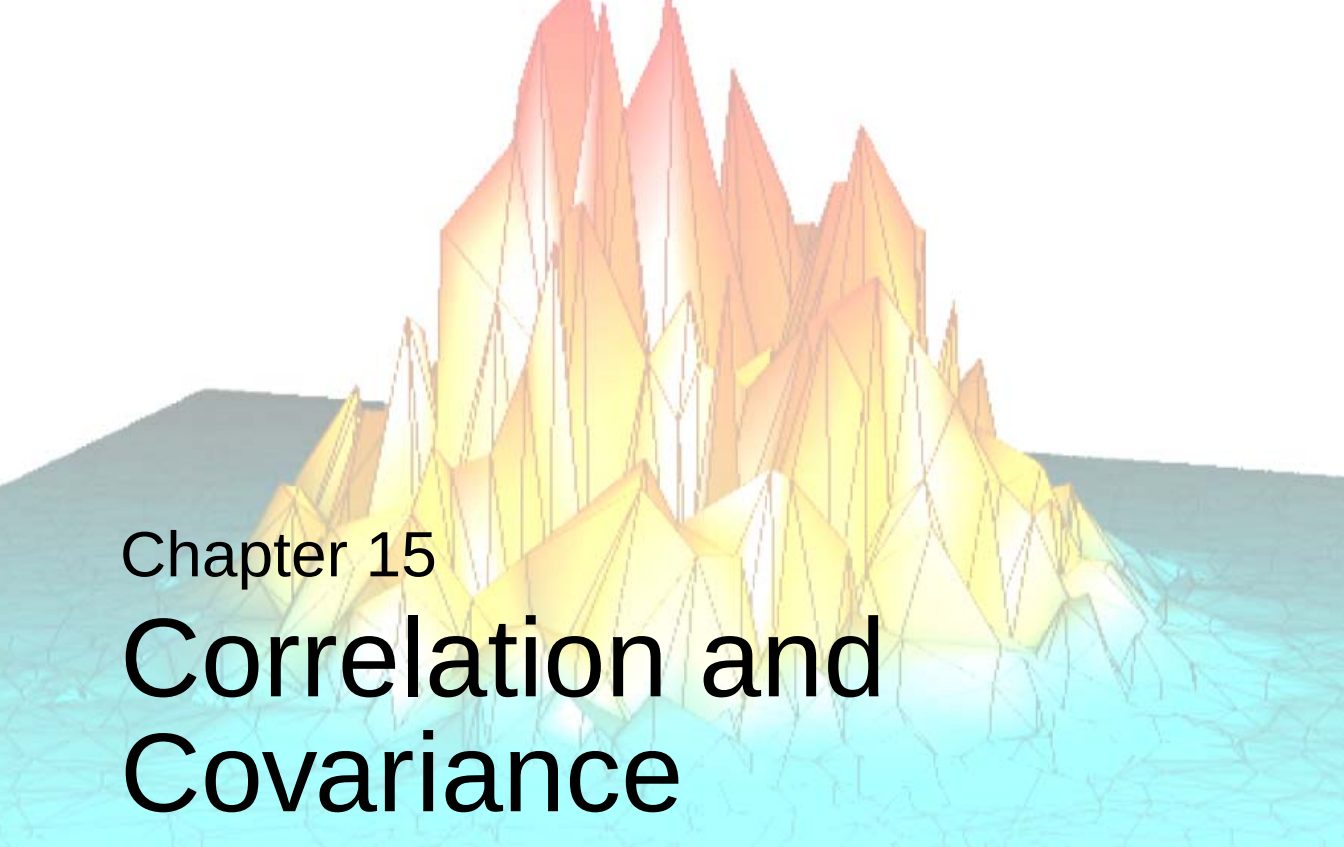
B =      1.00      1.00

```

Rank of Regressors Matrix	=	2
Magnitude of Largest Residual	=	1.0000
Number of Iterations	=	3
Number of Rows Missing	=	0

Version History

6.4	Introduced



Chapter 15

Correlation and Covariance

This section contains the following topics:

Overview: Correlation and Covariance . . .	720	Correlation and Covariance Routines	721

Overview: Correlation and Covariance

This chapter discusses measures of correlation for bivariate data. Topics covered include:

- The usual multivariate measures of correlation and covariance for continuous random variables (produced by `IMSL_COVARIANCES`).
- Data grouped by some auxiliary variable (`IMSL_POOLED_COV` can be used to compute the pooled covariance matrix along with the means for each group).
- Partial correlations or covariances computed using `IMSL_PARTIAL_COV`.
- Use of the `IMSL_ROBUST_COV` function to compute robust M-estimates of the mean and covariance matrix from a matrix of observations.

Correlation and Covariance Routines

[IMSL_COVARIANCES](#)—Variance-covariance or correlation matrix.

[IMSL_PARTIAL_COV](#)—Partial correlations and covariances.

[IMSL_POOLED_COV](#)—Pooled covariance matrix.

[IMSL_ROBUST_COV](#)—Robust estimate of covariance matrix.

IMSL_COVARIANCES

The IMSL_COVARIANCES function computes the sample variance-covariance or correlation matrix.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_COVARIANCES(x [, /DOUBLE] [, VAR_COVAR=value]
    [, CORRECTED_SSCP=value] [, CORRELATION=value]
    [, STDEV_CORRELATION=value] [, FREQUENCIES=array]
    [, INCIDENCE_MAT=variable] [, MISSING_VAL=value] [, MEANS=variable]
    [, NMISSING=variable] [, NOBS=variable] [, SUM_WEIGHTS=variable]
    [, WEIGHTS=array])
```

Return Value

If no keywords are used, IMSL_COVARIANCES returns a two-dimensional matrix containing the sample variance-covariance matrix of the observations in which value in element (i, j) corresponds to the sample covariance between the i -th and j -th variable.

Arguments

x

Two-dimensional matrix containing the data. The data value for the i -th observation of the j -th variable should be in $x(i, j)$.

Keywords

DOUBLE

If present and nonzero, double precision is used.

VAR_COVAR

Variance-covariance matrix (default).

Note

Exactly one of these keywords — VAR_COVAR, CORRECTED_SSCP, CORRELATION, STDEV_CORRELATION — is used to specify the type of matrix to be computed.

CORRECTED_SSCP

Corrected sum-of-squares and crossproducts matrix.

Note

Exactly one of these keywords — VAR_COVAR, CORRECTED_SSCP, CORRELATION, STDEV_CORRELATION — is used to specify the type of matrix to be computed.

CORRELATION

Correlation matrix.

Note

Exactly one of these keywords — VAR_COVAR, CORRECTED_SSCP, CORRELATION, STDEV_CORRELATION — is used to specify the type of matrix to be computed.

STDEV_CORRELATION

Correlation matrix, except for diagonal elements which are standard deviations.

Note

Exactly one of these keywords — VAR_COVAR, CORRECTED_SSCP, CORRELATION, STDEV_CORRELATION — is used to specify the type of matrix to be computed.

FREQUENCIES

Array containing the vector of frequencies for the observation. Default: all observations have a frequency of 1.

INCIDENCE_MAT

Named variable into which the incidence matrix is stored. If *Missing_Val* is 0, the number of valid observations is returned through this keyword; otherwise, the *nvar* x *nvar* matrix, where *nvar* is the number of variables in *x*, contains the number of pairs of valid observations used in calculating the crossproducts for covariance.

MISSING_VAL

Scalar integer which defines the method used to exclude missing values in *x* from the computations, where NaN is interpreted as the missing value code. The methods are as follows:

- 0—The exclusion is listwise. (The entire row of *x* is excluded if any of the values of the row is equal to the missing value code.)
- 1—Raw crossproducts are computed from all valid pairs and means, and variances are computed from all valid data on the individual variables. Corrected crossproducts, covariances, and correlations are computed using these quantities.
- 2—Raw crossproducts, means, and variances are computed as in the case of *Missing_Val* = 1. However, corrected crossproducts and covariances are computed only from the valid pairs of data. Correlations are computed using these covariances and the variances from all valid data.
- 3—Raw crossproducts, means, variances, and covariances are computed as in the case of *Missing_Val* = 2. Correlations are computed using these covariances, but the variances used are computed from the valid pairs of data.

MEANS

Named variable into which array containing the means of variables in *x* is stored. The *i*-th components of the array correspond to $x(*, i)$.

NMISSING

Specifies a variable into which the total number of observations that contain any missing values (NaN) is stored.

NOBS

Named variable into which the sum of the frequencies is stored. If *Missing_Val* is 0, observations with missing values are not included in *Nobs*; otherwise, all observations are included except for observations with missing values for the weight or the frequency.

SUM_WEIGHTS

Specifies a variable into which the sum of the weights of all observations is stored. If keyword *Missing_val* is equal to 0, observations with missing values are not included in *Sum_weights*. Otherwise, all observations are included except for observations with missing values for the weight or the frequency.

WEIGHTS

Array containing the vector of weights for the observation. Default: all observations have equal weights of 1.

Discussion

The IMSL_COVARIANCES function computes estimates of correlations, covariances, or sum of squares and crossproducts for a data matrix x . The means, (corrected) sum of squares, and (corrected) sums of crossproducts are computed using the method of provisional means.

Let:

$$\bar{x}_{ki}$$

denote the mean based on i observations for the k -th variable, f_i and w_i denote the frequency and weight of the i -th observation, respectively, and let c_{jki} denote the sum of crossproducts (or sum of squares if $j = k$) based on i observations. Then, the method of provisional means finds new means and sums of crossproducts shown in the example below.

The means and crossproducts are initialized as:

$$\bar{x}_{k0} = 0.0 \quad k = 0, \dots, p - 1$$

$$c_{jk0} = 0.0 \quad j, k = 0, \dots, p - 1$$

where p denotes the number of variables. Letting $x_{k, i+1}$ denote the k -th variable on observation $i + 1$, each new observation leads to the following updates for:

$$\bar{x}_{ki}$$

and c_{jki} using update constant r_{i+1} :

$$r_{i+1} = \frac{f_{i+1} w_{i+1}}{\sum_{j=0}^{i+1} f_j w_j}$$

$$\bar{x}_{k,i+1} = \bar{x}_{ki} + (x_{k,i+1} - \bar{x}_{ki})r_{i+1}$$

$$c_{jk,i+1} = c_{jki} + f_{i+1} w_{i+1} (x_{j,i+1} - \bar{x}_{ji})(x_{k,i+1} - \bar{x}_{ki})(1 - r_{i+1})$$

Syntax Notes

The IMSL_COVARIANCES function uses the following definition of a sample mean:

$$\bar{x}_k = \frac{\sum_{i=1}^{n_r} f_i w_i x_{ki}}{\sum_{i=1}^{n_r} f_i w_i}$$

where n_r is the number of cases. The formula below defines the sample covariance, s_{jk} , between variables j and k .

$$s_{jk} = \frac{\sum_{i=1}^n f_i w_i (x_{ji} - \bar{x}_j)(x_{ki} - \bar{x}_k)}{(\sum_{i=1}^n f_i) - 1}$$

The sample correlation between variables j and k , r_{jk} , is defined below:

$$r_{jk} = \frac{s_{jk}}{\sqrt{s_{jj}s_{kk}}}$$

Example

This example illustrates the use of IMSL_COVARIANCES for the first 50 observations in the Fisher iris data (Fisher 1936). Note that the first variable is constant over the first 50 observations.

```
x = IMSL_STATDATA(3)
x = x(0:49, *)
cov = IMSL_COVARIANCES(x)
; Call IMSL_COVARIANCES.
PM, cov
; Output the results.
```

0.00000	0.00000	0.00000	0.00000	0.00000
0.00000	0.124249	0.0992163	0.0163551	0.0103306
0.00000	0.0992163	0.143690	0.0116980	0.00929796
0.00000	0.0163551	0.0116980	0.0301592	0.00606939
0.00000	0.0103306	0.00929796	0.00606939	0.0111061

Errors

Warning Errors

STAT_CONSTANT_VARIABLE—Correlations are requested, but the observations on one or more variables are constant. The corresponding correlations are set to NaN.

Version History

6.4	Introduced

IMSL_PARTIAL_COV

The IMSL_PARTIAL_COV function computes partial covariances or partial correlations from the covariance or correlation matrix.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_PARTIAL_COV(n_independent, n_dependent, x [, /DOUBLE]  
[, /CORR] [, /COV] [, DF=integer] [, INDICES=array] [, /PVALS=variable])
```

Return Value

Array of size *n_dependent* by *n_dependent* containing the partial covariances (the default) or partial correlations (set keyword *Corr*).

Arguments

n_dependent

Number of variables for which partial covariances/correlations are desired (the number of “dependent” variables).

n_independent

Number of “independent” variables to be used in the partial covariances/correlations. The partial covariances/correlations are the covariances/correlations between the dependent variables after removing the linear effect of the independent variables.

x

The *n* by *n* covariance or correlation matrix, where $n = n\_independent + n\_dependent$. The rows/columns must be ordered such that the first *n_independent* rows/columns contain the independent variables, and the last *n_dependent* rows/columns contain the dependent variables. Array *x* must always be square symmetric.

Keywords

DOUBLE

If present and nonzero, double precision is used.

CORR

If present and nonzero, then partial correlations are calculated. Keywords *Cov* and *Corr* can not be used together.

COV

If present and nonzero, then partial covariances are calculated. (Default) Keywords *Cov* and *Corr* can not be used together.

DF

On input, an integer indicating the number of degrees of freedom associated with input array *x*. If the number of degrees of freedom in *x* varies from element to element, then a conservative choice for *Df* is the minimum degrees of freedom for all elements in *x*.

Upon output, named variable into which the number of degrees of freedom in the test that the partial covariances/correlations are zero is stored. This value will usually be *Df* – *n_independent*, but will be greater than this value if the independent variables are computationally linearly related. Keywords *Df* and *Pvals* must be used together.

INDICES

An array containing values indicating the status of the variable as in [Figure 15-1](#):

Indices(i)	Variable is...
–1	not used in analysis
0	dependent variable
1	independent variable

Table 15-1: Indices

Default: The first *n_independent* elements of *Indices* are equal to 1, and the last *n_dependent* elements are equal to 0.

PVALS

Named variable into which an array of size $n\_dependent$ by $n\_dependent$ containing the p -values for testing the null hypothesis that the associated partial covariance/correlation is zero is stored. It is assumed that the observations from which x was computed flows a multivariate normal distribution and that each element in x has Df degrees of freedom. Keywords Df and $Pvals$ must be used together.

Discussion

The `IMSL_PARTIAL_COV` function computed partial covariances or partial correlations from an input covariance or correlation matrix. If the “independent” variables (the linear “effect” of the independent variables is removed in computing the partial covariances/correlations) are linearly related to one another, `IMSL_PARTIAL_COV` detects the linearity and eliminates one or more of the independent variables from the list of independent variables. The number of variables eliminated, if any, can be determined from keyword Df .

Given a covariance or correlation matrix Σ partitioned as:

$$\begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix}$$

`IMSL_PARTIAL_COV` computed the partial covariances (of the standardized variables if Σ is a correlation matrix) as:

$$\Sigma_{22|1} = \Sigma_{22} - \Sigma_{21} \Sigma_{11}^{-1} \Sigma_{12}$$

If partial correlations are desired, these are computed as:

$$P_{22|1} = [\text{diag}(\Sigma_{22|1})]^{-1/2} \Sigma_{22|1} [\text{diag}(\Sigma_{22|1})]^{-1/2}$$

where *diag* denotes the matrix containing the diagonal of its argument along its diagonal with zeros off the diagonal. If Σ_{11} is singular, then as many variables as required are deleted from Σ_{11} (and Σ_{12}) in order to eliminate the linear dependencies. The computations then proceed as above.

The p -value for a partial covariance tests the null hypothesis $H_0: \sigma_{ij|1} = 0$, where $\sigma_{ij|1}$ is the (i, j) element in matrix $\Sigma_{22|1}$. The p -value for a partial correlation tests the null hypothesis $H_0: \rho_{ij|1} = 0$, where $\rho_{ij|1}$ is the (i, j) element in matrix $P_{22|1}$. The p -values are returned in $Pvals$. If the degrees of freedom for x , Df , is not known, the resulting p -values may be useful for comparison, but they should not be used as an approximation to the actual probabilities.

Examples

Example 1

The following example computes partial covariances, scaled from a nine-variable correlation matrix originally given by Emmett (1949). The first three rows and columns contain the independent variables and the final six rows and columns contain the dependent variables.

```
x = TRANSPOSE([ $
  [6.300, 3.050, 1.933, 3.365, 1.317, 2.293, 2.586, 1.242, $
  4.363], [3.050, 5.400, 2.170, 3.346, 1.473, 2.303, 2.274, $
  0.750, 4.077], [1.933, 2.170, 3.800, 1.970, 0.798, 1.062, $
  1.576, 0.487, 2.673], [3.365, 3.346, 1.970, 8.100, 2.983, $
  4.828, 2.255, 0.925, 3.910], [1.317, 1.473, 0.798, 2.983, $
  2.300, 2.209, 1.039, 0.258, 1.687], [2.293, 2.303, 1.062, $
  4.828, 2.209, 4.600, 1.427, 0.768, 2.754], [2.586, 2.274, $
  1.576, 2.255, 1.039, 1.427, 3.200, 0.785, 3.309], [1.242, $
  0.750, 0.487, 0.925, 0.258, 0.768, 0.785, 1.300, 1.458], $
  [4.363, 4.077, 2.673, 3.910, 1.687, 2.754, 3.309, 1.458, $
  7.400]])
pcov = IMSL_PARTIAL_COV(3, 6, x)
PM, pcov, FORMAT = '(6F10.3)', Title = 'Partial Covariances'
```

Partial Covariances

0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	5.495	1.895	3.084
0.000	0.000	0.000	1.895	1.841	1.476
0.000	0.000	0.000	3.084	1.476	3.403

Example 2

The following example computes partial correlations from a 9 variable correlation matrix originally given by Emmett (1949). The partial correlations between the remaining variables, after adjusting for variables 1, 3 and 9, are computed.

```
x = TRANSPOSE([ $
  [1.0, 0.523, 0.395, 0.471, 0.346, 0.426, 0.576, 0.434, $
  0.639], [0.523, 1.0, 0.479, 0.506, 0.418, 0.462, 0.547, $
  0.283, 0.645], [0.395, 0.479, 1.0, 0.355, 0.27, 0.254, $
  0.452, 0.219, 0.504], [0.471, 0.506, 0.355, 1.0, 0.691, $
  0.791, 0.443, 0.285, 0.505], [0.346, 0.418, 0.27, 0.691, $
  1.0, 0.679, 0.383, 0.149, 0.409], [0.426, 0.462, 0.254, $
  0.791, 0.679, 1.0, 0.372, 0.314, 0.472], [0.576, 0.547, $
  0.452, 0.443, 0.383, 0.372, 1.0, 0.385, 0.68], [0.434, $
  0.283, 0.219, 0.285, 0.149, 0.314, 0.385, 1.0, 0.47], $
```

```

    [0.639, 0.645, 0.504, 0.505, 0.409, 0.472, 0.68, 0.47, 1.0]])
df = 30
indices = [1, 0, 1, 0, 0, 0, 0, 0, 1]
pcov = IMSL_PARTIAL_COV(3, 6, x, Indices = indices, Df = df, $
    Pvals = pvals, /Corr)
PRINT, 'Degrees Of Freedom: ', df
PM, pcov, FORMAT = '(6F10.3)', Title = 'Partial Correlations'
PM, pvals, FORMAT = '(6F10.4)', Title = 'P values'

```

IDL Prints:

```

Degrees Of Freedom:                27
Partial Correlations
  1.000    0.224    0.194    0.211    0.125   -0.061
  0.224    1.000    0.605    0.720    0.092    0.025
  0.194    0.605    1.000    0.598    0.123   -0.077
  0.211    0.720    0.598    1.000    0.035    0.086
  0.125    0.092    0.123    0.035    1.000    0.062
 -0.061    0.025   -0.077    0.086    0.062    1.000
P values
  0.0000    0.2525    0.3232    0.2801    0.5249    0.7576
  0.2525    0.0000    0.0006    0.0000    0.6417    0.9000
  0.3232    0.0006    0.0000    0.0007    0.5328    0.6982
  0.2801    0.0000    0.0007    0.0000    0.8602    0.6650
  0.5249    0.6417    0.5328    0.8602    0.0000    0.7532
  0.7576    0.9000    0.6982    0.6650    0.7532    0.0000

```

Errors

Warning Errors

STAT_NO_HYP_TESTS—The input matrix “x” has # degrees of freedom, and the rank of the dependent variables is #. There are not enough degrees of freedom for hypothesis testing. The elements of “Pvals” are set to NaN (not a number).

Fatal Errors

STAT_INVALID_MATRIX_1—The input matrix “x” is incorrectly specified. A computed correlation is greater than 1 for variables # and #.

STAT_INVALID_PARTIAL—A computed partial correlation for variables # and # is greater than 1. The input matrix “x” is not positive semi-definite.

Version History

6.4	Introduced

IMSL_POOLED_COV

The IMSL_POOLED_COV function computes a pooled variance-covariance from the observations.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_POOLED_COV(x, ngroups [, /DOUBLE] [, GCOUNTS=variable]
    [, IDX_COLS=array] [, IDX_VARS=array] [, MEANS=variable]
    [, NMISSING=variable] [, SUM_WEIGHTS=variable] [, U=variable])
```

Return Value

Two-dimensional array containing the matrix of covariances.

Arguments

ngroups

Number of groups in the data.

x

Two-dimensional array containing the data. The first $n\_variables = (N\_ELEMENTS(x(0,*)) - 1)$ columns correspond to the variables, and the last column must contain the group numbers.

Keywords

DOUBLE

If present and nonzero, double precision is used.

GCOUNTS

Named variable into which the array of length n_groups containing the number of observations in each group is stored.

IDX_COLS

One-dimensional array containing the indices of the variables to be used in the analysis.

IDX_VARS

Three element array indicating the column numbers of x in which particular types of data are stored. Columns are numbered 0 ... $N\_ELEMENTS(IdX\_Cols) - 1$.

- $Idx\_Vars(0)$ contains the index for the column of x in which the group numbers are stored.
- $Idx\_Vars(1)$ and $Idx\_Vars(2)$ contain column numbers of x in which the frequencies and weights, respectively, are stored. Set $Idx\_Vars(1) = -1$ if there will be no column for frequencies. Set $Idx\_Vars(2) = -1$ if there will be no column for weights. Weights are rounded to the nearest integer. Negative weights are not allowed.
- Defaults: $Idx\_Cols = 0, 1, \dots, n\_variables - 1$,
 $Idx\_Vars(0) = n\_variables$,
 $Idx\_Vars(1) = -1$, and
 $Idx\_Vars(2) = -1$

MEANS

Named variable into which the array of size n_groups by $n\_variables$ in which the i -th row of *Means* contains the group i variable means is stored.

NMISSING

Named variable into which the number of rows of data containing missing values (NaN) for any of the variables used is stored.

SUM_WEIGHTS

Named variable into which the array of length n_groups containing the sum of the weights times the frequencies in the groups is stored.

U

Named variable into which the array of size $n\_variables$ by $n\_variables$ containing the lower matrix U , the lower triangular for the pooled sample cross-products matrix is stored. U is computed from the pooled sample covariance matrix, S (See the *Discussion* section), as $S = U^T U$.

Discussion

The IMSL_POOLED_COV function computes the pooled variance-covariance matrix from a matrix of observations. The within-groups means are also computed. Listwise deletion of missing values is assumed so that all observations used are complete; in any row of x , if any element of the observation is missing, the row is not used. The IMSL_POOLED_COV function should be used whenever you suspect the data has been sampled from populations with different means but identical variance-covariance matrices. If these assumptions cannot be made, a different variance-covariance matrix should be estimated within each group.

If $N\_ELEMENTS(x(*,0))$ (0, the group observation totals, T_i , for $i = 1, \dots, g$, where g is the number of groups, are updated for the $N\_ELEMENTS(x(*,0))$ observations in x . The group totals are computed as:

$$T_i = \sum_j w_{ij} f_{ij} x_{ij}$$

where w_{ij} is the observation weight, x_{ij} is the j -th observation in the i -th group, and f_{ij} is the observation frequency.

Modified Givens rotations are used in computed the Cholesky decomposition of the pooled sums of squares and crossproducts matrix. (Golub and Van Loan 1983).

The group means and the pooled sample covariance matrix S are computed from the intermediate results. These quantities are defined by:

$$\bar{x}_{i\bullet} = \frac{T_i}{\sum_j w_{ij} f_{ij}}$$

$$S = \frac{1}{\sum_{ij} f_{ij} - g} \sum_{ij} w_{ij} f_{ij} (x_{ij} - \bar{x}_{i\bullet})(x_{ij} - \bar{x}_{i\bullet})^T$$

Example

The following example computes a pooled variance-covariance matrix. The last column of the data set is the group indicator.

```
ngroups = 2
x = TRANSPOSE([[2.2, 5.6, 1], [3.4, 2.3, 1], [1.2, 7.8, 1], $
               [3.2, 2.1, 2], [4.1, 1.6, 2], [3.7, 2.2, 2]])
cov = IMSL_POOLED_COV(x, ngroups)
PM, cov, FORMAT = '(2F10.3)', Title = 'Pooled Covariance Matrix'
```

```
Pooled Covariance Matrix
 0.708      -1.575
-1.575       3.883
```

Errors

Warning Errors

STAT_OBSERVATION_IGNORED—In call #, row # of the matrix “x” has group number = #. The group number must be between 1 and #, the number of groups. This observation will be ignored.

Version History

6.4	Introduced

IMSL_ROBUST_COV

The IMSL_ROBUST_COV function computes a robust estimate of a covariance matrix and mean vector.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_ROBUST_COV(x, n_groups [, BETA=variable]
    [, COV_EST=array] [, /DOUBLE] [, GROUP_COUNTS=variable] [, /HUBER]
    [, IDX_COLS=array] [, IDX_VARS=array] [, /INIT_EST_MEAN]
    [, /INIT_EST_MEDIAN] [, ITMAX=value] [, MEAN_EST=array]
    [, MEANS=variable] [, MINIMAX_WEIGHTS=variable]
    [, NMISSING=variable] [, PERCENTAGE=value] [, /STAHEL]
    [, SUM_WEIGHTS=variable] [, TOLERANCE=value] [, U=variable])
```

Return Value

Two-dimensional array containing the matrix of covariances.

Arguments

n_groups

Number of groups in the data.

x

Two-dimensional array of size *nrows* by (*n_variables* + 1) containing the data where *nrows* = N_ELEMENTS(x(*,0)) and *n_variables* = (N_ELEMENTS(x(0,*)) - 1). The first *n_variables* columns correspond to the variables, and the last column must contain the group numbers.

Keywords

BETA

Named variable into which the constant used to ensure that the estimated covariance matrix has unbiased expectation (for a given mean vector) for a multivariate normal density is stored.

COV_EST

Two-dimensional array of size $n\_variables$ by $n\_variables$ containing the estimate of the covariance matrix. Keywords *Mean_Est* and *Cov_Est* must be used together.

DOUBLE

If present and nonzero, double precision is used.

GROUP_COUNTS

Named variable into which the one-dimensional array of length n_groups containing the number of observations in each group is stored.

HUBER

If present and nonzero, Huber's conjugate-gradient algorithm is used. Keywords *Stahel* and *Huber* can not be used together.

IDX_COLS

One-dimensional array containing the indices of the variables to be used in the analysis.

IDX_VARS

Three element array indicating the column numbers of x in which particular types of data are stored. Columns are numbered 0 ... $N\_ELEMENTS(Idx\_Cols) - 1$.

- *Idx_Vars*(0) contains the index for the column of x in which the group numbers are stored.
- *Idx_Vars*(1) and *Idx_Vars*(2) contain column numbers of x in which the frequencies and weights, respectively, are stored. Set *Idx_Vars*(1) = -1 if there will be no column for frequencies. Set *Idx_Vars*(2) = -1 if there will be no column for weights. Weights are rounded to the nearest integer. Negative weights are not allowed.

- Defaults: $Idx\_Cols = 0, 1, \dots, n\_variables - 1$,
 $Idx\_Vars(0) = n\_variables$,
 $Idx\_Vars(1) = -1$, and
 $Idx\_Vars(2) = -1$

INIT_EST_MEAN

If present and nonzero, initial estimates are obtained as the usual estimate of a mean vector and of a covariance matrix. Keywords *Init_Est_Mean*, *Init_Est_Median*, and *Mean_Est* can not be used together.

INIT_EST_MEDIAN

If present and nonzero, initial estimates based upon the median and interquartile range must be used. Keywords *Init_Est_Mean*, *Init_Est_Median*, and *Mean_Est* can not be used together.

ITMAX

Maximum number of iterations. Default: *Itmax* = 30

MEAN_EST

Two-dimensional array of size n_groups by $n\_variables$ containing initial estimates for the mean. Keywords *Mean_Est* and *Cov_Est* must be used together. Keywords *Init_Est_Mean*, *Init_Est_Median*, and *Mean_Est* can not be used together.

MEANS

Named variable into which the array of size n_groups by $n\_variables$ is stored. The i -th row of *Means* contains the group i variable means.

MINIMAX_WEIGHTS

Named variable into which the one-dimensional array containing the values for the parameters of the weighting function is stored. See the *Discussion* section for details.

NMISSING

Named variable into which the number of rows of data containing missing values (NaN) for any of the variables used is stored.

PERCENTAGE

Percentage of gross errors expected in the data. Keyword *Percentage* must be in the range 0.0 to 100.0 and contains the percentage of outliers expected in the data. If the percentage of gross errors expected in the data is not known, a reasonable strategy is to choose a value of *Percentage* that is such that larger values do not result in significant changes in the estimates. Default: *Percentage* = 5.0

STAHEL

If present and nonzero, the Stahel's algorithm is used. Keywords *Stahel* and *Huber* cannot be used together.

SUM_WEIGHTS

Named variable into which the one-dimensional array of length *n_groups* containing the sum of the weights times the frequencies in the groups is stored.

TOLERANCE

Convergence criterion. When the maximum absolute change in a location or covariance estimate is less than *Tolerance*, convergence is assumed. Default: *Tolerance* = 10^{-4}

U

Named variable into which an array of size *n_variables* by *n_variables* containing the lower matrix *U*, the lower triangular for the robust sample cross-products matrix is stored. *U* is computed from the robust sample covariance matrix, *S* (See the *Discussion* section), as $S = U^T U$.

Discussion

The IMSL_ROBUST_COV function computes robust M-estimates of the mean and covariance matrix from a matrix of observations. A pooled estimate of the covariance matrix is computed when multiple groups are present in the input data. M-estimate weights are obtained using the “minimax” weights of Huber (1981, pp. 231-235), with *Percentage* expected gross errors. Huber's (1981) weighting equations are given by:

User specified observation weights and frequencies may be given for each row in *x*. Listwise deletion of missing values is assumed so that all observations used are “complete”.

$$u(r) = \begin{cases} \frac{a^2}{r^2} & r < a \\ 1 & a \leq r \leq b \\ \frac{b^2}{r^2} & r > b \end{cases}$$

$$w(r) = \min\left(1, \frac{c}{r}\right)$$

Let $f(x; \mu_i, \Sigma)$ denote the density of an observation p -vector x in population (group) i with mean vector μ_i , for $i = 1, \dots, \tau$. Let the covariance matrix Σ be such that $\Sigma = R^T R$. If:

$$y = R^{-T} (x - \mu_i)$$

then:

$$g(y) = |\Sigma|^{1/2} f(R^T y + \mu_i; \mu_i, \Sigma)$$

It is assumed that $g(y)$ is a spherically symmetric density in p -dimensions.

In IMSL_ROBUST_COV, Σ and μ_i are estimated as the solutions:

$$(\hat{\Sigma}, \hat{\mu}_i)$$

of the estimation equations:

$$\frac{1}{n} \sum_{j=1}^{n_i} f_{ij} w_{ij} w(r_{ij}) y_{ij} = 0$$

and:

$$\frac{1}{n} \sum_{i=1}^{\tau} \sum_{j=1}^{n_i} f_{ij} w_{ij} [u(r_{ij}) y_{ij} y_{ij}^T - \beta I_p] = 0$$

where i indexes the τ groups, n_i is the number of observations in group i , f_{ij} is the frequency for the j -th observation in group i , w_{ij} is the observation weight specified in column $Idx\_Vars(2)$ of x , I_p is a p by p identity matrix,

$$r_{ij} = \sqrt{y_{ij}^T y_{ij}}$$

$w(r)$ and $u(r)$ are the weighting functions, and where β is a constant computed by the program to make the expected weighted Mahalanobis distance ($y^T y$) equal the expected Mahalanobis distance from a multivariate normal distribution (see Marazzi 1985). The constant β is described more fully below.

The IMSL_ROBUST_COV function uses one of two algorithms for solving the estimation equations. The first algorithm is discussed in detail in Huber (1981) and is a variant of the conjugate gradient method. The second algorithm is due to Stahel (1981) and is discussed in detail by Marazzi (1985). In both algorithms, correction vectors T_{ki} for the group i means and correction matrix $W_k = I_p + U_k$ for the Cholesky factorization of S are found such that the updated mean vectors are given by:

$$\hat{u}_{i,k+1} = \hat{u}_{ik} + T_{ki}$$

and the updated matrix R is given as:

$$\hat{R}_{k+1} = W_k \hat{R}_k$$

where k is the iteration number and:

$$\hat{\Sigma}_k = R_k^T R_k$$

When all elements of U_k and T_{ki} are less than $\epsilon = \textit{Tolerance}$, convergence is assumed.

Three methods for obtaining estimates are allowed. In the first method, the sample weighted estimate of Σ is computed. In the second method, estimates based upon the median and the interquartile range are used. Finally, in the last method, you input initial estimates.

The IMSL_ROBUST_COV function computes estimates based on the “minimax” weights discussed above. The constant β is chosen such that $E(u(r)r_2) = \rho\beta$ where the expectation is with respect to a standard p -variate multivariate normal distribution. This yields estimates with the correct expectation for the multivariate normal distribution (for given mean vector). The expectation is computed via integration of estimated spline function. 200 knots are used on an equally spaced grid from 0.0 to the 99.999 percentile of:

$$\chi_p^2$$

distribution. An error estimate is computed based upon 100 of these knots. If the estimated relative error is greater than 0.0001, a warning message is issued. If β is not computed accurately (i.e., if warning message is issued), the computed estimates are still optimal, but the scale of the estimated covariance matrix may need to be multiplied by a constant in order for:

$$\hat{\Sigma}$$

to have the correct multivariate normal covariance expectation.

Examples

Example 1

The following example computes a robust variance-covariance matrix. The last column of the data set is the group indicator.

```
n_groups = 2
x = TRANSPOSE([ [2.2, 5.6, 1.0], [3.4, 2.3, 1.0], $
  [1.2, 7.8, 1.0], [3.2, 2.1, 2.0], [4.1, 1.6, 2.0], $
  [3.7, 2.2, 2.0] ])
cov = IMSL_ROBUST_COV(x, n_groups)
PM, cov, Title = 'Robust Covariance Matrix'
```

```
Robust Covariance Matrix
0.522022      -1.16027
-1.16027      2.86203
```

Example 2

The following example computes estimates of the pooled covariance matrix for the Fisher's iris data. For comparison, the estimates are first computed via `IMSL_POOLED_COV`. The `IMSL_ROBUST_COV` function with *Percentage* = 2.0 is then used to compute the robust estimates. As can be seen from the output, the resulting estimates are quite similar.

Next, three observations are made into outliers, and again, estimates are computed using functions `IMSL_POOLED_COV` and `IMSL_ROBUST_COV`. When outliers are present, the estimates of `IMSL_POOLED_COV` are adversely affected, while the estimates produced by `IMSL_ROBUST_COV` are close to the estimates produced when no outliers are present.

```
n_groups = 3
idxv = [1, 2, 3, 4]
idxc = [0, -1, -1]
percentage = 2.0
x = IMSL_STATDATA(3)
p_cov = IMSL_POOLED_COV(x, n_groups, Idx_Vars = idxv, $
  Idx_Cols = idxc)
PM, p_cov, Title = 'Pooled Covariance with No Outliners'
r_cov = IMSL_ROBUST_COV(x, n_groups, Idx_Vars = idxv, $
  Idx_Cols = idxc, Percentage = percentage)
PM, r_cov, Title = 'Robust Covariance with No Outliners'
```

IDL Prints:

Pooled Covariance with No Outliners

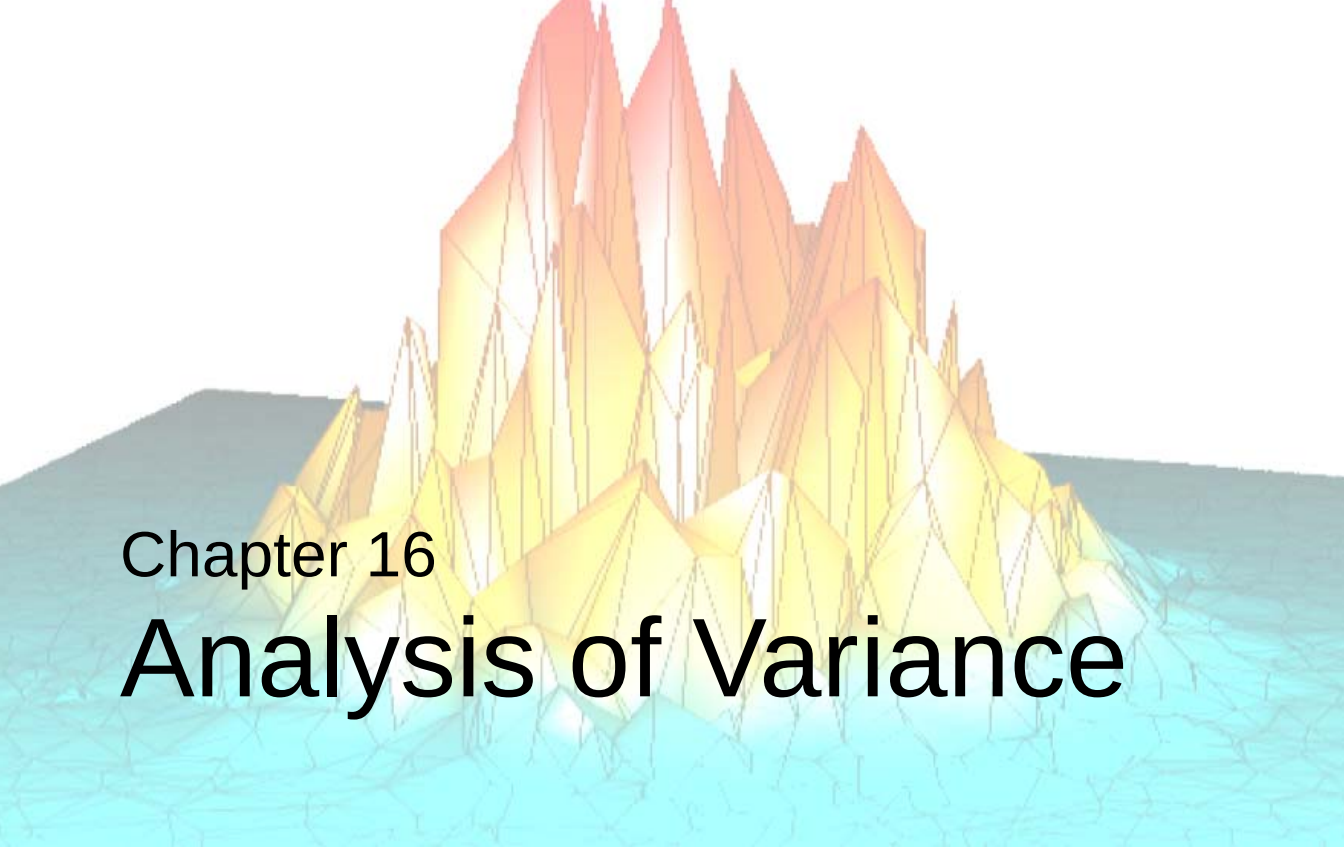
0.265008	0.0927211	0.167514	0.0384014
0.0927211	0.115388	0.0552436	0.0327102
0.167514	0.0552436	0.185188	0.0426653
0.0384014	0.0327102	0.0426653	0.0418816

Robust Covariance with No Outliners

0.247410	0.0872090	0.153530	0.0359695
0.0872090	0.107336	0.0538220	0.0321557
0.153530	0.0538220	0.170550	0.0411720
0.0359695	0.0321557	0.0411720	0.0401394

Version History

6.4	Introduced



Chapter 16

Analysis of Variance

This section describes functions for analysis of variance models and for multiple comparison methods for means.

Overview: Analysis of Variance	748	Analysis of Variance Routines	749

Overview: Analysis of Variance

The functions described in this chapter are for commonly-used experimental designs. Typically, responses are stored in the input vector y in a pattern that takes advantage of the balanced design structure. Consequently, the full set of model subscripts is not needed to identify each response. The functions assume the usual pattern, which requires that the last model subscript change most rapidly, followed by the model subscript next in line, and so forth, with the first subscript changing at the slowest rate. This pattern is referred to as *lexicographical ordering*.

The `IMSL_ANOVA1` function allows missing responses if confidence interval information is not requested. NaN (Not a Number) is the missing value code used by these functions. Use `IMSL_MACHINE` to retrieve NaN. Any element of y that is missing must be set to NaN. Other functions described in this chapter do not allow missing responses because the functions generally deal with balanced designs.

As a diagnostic tool for determination of the validity of a model, functions in this chapter typically perform a test for lack of fit when n ($n > 1$) responses are available in each cell of the experimental design. Functions in [Chapter 14](#), “Regression” are used for analysis of generalizations of the models treated in this chapter. In particular, [Chapter 2: Regression](#), also provides functions for the general linear model.

Analysis of Variance Routines

[IMSL_ANOVA1](#)—Analyzes a one-way classification model.

[IMSL_ANOVAFACT](#)—Analyzes a balanced factorial design with fixed effects.

[IMSL_MULTICOMP](#)—Performs Student-Newman-Keuls multiple comparisons test.

[IMSL_ANOVANESTED](#)—Nested random model.

[IMSL_ANOVABALANCED](#)—Balanced fixed, random, or mixed model.

IMSL_ANOVA1

The IMSL_ANOVA1 function analyzes a one-way classification model.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_ANOVA1(n, y [, ANOVA_TABLE=variable]
    [, BONFERRONI=variable] [, CONFIDENCE=value] [, /DOUBLE]
    [, DUNN_SIDAK=variable] [, GROUP_COUNTS=variable]
    [, GROUP_MEANS=variable] [, GROUP_STD_DEV=variable]
    [, ONE_AT_A_TIME=variable] [, SCHEFFE=variable] [, TUKEY=variable])
```

Return Value

The p -value for the F -statistic.

Arguments

n

One-dimensional array containing the number of responses for each group.

y

One-dimensional array of length:

$$n(0) + n(1) + \dots + n(N\_ELEMENTS(n) - 1)$$

containing the responses for each group.

Keywords

ANOVA_TABLE

Named variable into which the analysis of variance table is stored. The analysis of variance statistics are as follows:

- 0—degrees of freedom for the model
- 1—degrees of freedom for error
- 2—total (corrected) degrees of freedom
- 3—sum of squares for the model
- 4—sum of squares for error
- 5—total (corrected) sum of squares
- 6—model mean square
- 7—error mean square
- 8—overall F -statistic
- 9— p -value
- 10— R^2 (in percent)
- 11—Adjusted R^2 (in percent)
- 12—estimate of the standard deviation
- 13—overall mean of y
- 14—coefficient of variation (in percent)

BONFERRONI

Named variable into which the array containing the statistics relating to the difference of means is stored. On return, the named variable contains an array of size:

$$\binom{ngroups}{2} \times 5$$

where $ngroups = N\_ELEMENTS(n)$.

- 0—group number for the i -th mean
- 1—group number for the j -th mean
- 2—difference of means (i -th mean) – (j -th mean)
- 3—lower confidence limit for the difference
- 4—upper confidence limit for the difference

The `IMSL_ANOVA1` function computes confidence intervals on all pairwise differences of means using one of six methods: Tukey, Tukey-Kramer, Dunn-Sidak, Bonferroni, Scheffé, or Fisher's LSD (One-at-a-Time). If *Tukey* is specified, Tukey confidence intervals are calculated if the group sizes are equal; otherwise, the Tukey-Kramer confidence intervals are calculated.

CONFIDENCE

Confidence level for the simultaneous interval estimation. If *Tukey* is specified, *Confidence* must be in the range [90.0, 99.0); otherwise, *Confidence* is in the range [0.0, 100.0). Default: *Confidence* = 95.0

DOUBLE

If present and nonzero, then double precision is used.

DUNN_SIDAK

Named variable into which the array containing the statistics relating to the difference of means is stored. On return, the named variable contains an array of size:

$$\binom{ngroups}{2} \times 5$$

where *ngroups* = N_ELEMENTS(*n*).

- 0—group number for the *i*-th mean
- 1—group number for the *j*-th mean
- 2—difference of means (*i*-th mean) – (*j*-th mean)
- 3—lower confidence limit for the difference
- 4—upper confidence limit for the difference

The IMSL_ANOVA1 function computes confidence intervals on all pairwise differences of means using one of six methods: Tukey, Tukey-Kramer, Dunn-Sidak, Bonferroni, Scheffé, or Fisher's LSD (One-at-a-Time). If *Tukey* is specified, Tukey confidence intervals are calculated if the group sizes are equal; otherwise, the Tukey-Kramer confidence intervals are calculated.

GROUP_COUNTS

Named variable into which the array containing the number of nonmissing observations for the groups is stored.

GROUP_MEANS

Named variable into which the array containing the group means is stored.

GROUP_STD_DEV

Named variable into which the array containing the group standard deviations is stored.

ONE_AT_A_TIME

Named variable into which the array containing the statistics relating to the difference of means is stored. On return, the named variable contains an array of size:

$$\binom{ngroups}{2} \times 5$$

where $ngroups = N\_ELEMENTS(n)$.

- 0—group number for the i -th mean
- 1—group number for the j -th mean
- 2—difference of means (i -th mean) – (j -th mean)
- 3—lower confidence limit for the difference
- 4—upper confidence limit for the difference

The `IMSL_ANOVA1` function computes confidence intervals on all pairwise differences of means using one of six methods: Tukey, Tukey-Kramer, Dunn-Sidak, Bonferroni, Scheffé, or Fisher's LSD (One-at-a-Time). If *Tukey* is specified, Tukey confidence intervals are calculated if the group sizes are equal; otherwise, the Tukey-Kramer confidence intervals are calculated.

SCHEFFE

Named variable into which the array containing the statistics relating to the difference of means is stored. On return, the named variable contains an array of size:

$$\binom{ngroups}{2} \times 5$$

where $ngroups = N\_ELEMENTS(n)$.

- 0—group number for the i -th mean
- 1—group number for the j -th mean
- 2—difference of means (i -th mean) – (j -th mean)
- 3—lower confidence limit for the difference
- 4—upper confidence limit for the difference

The `IMSL_ANOVA1` function computes confidence intervals on all pairwise differences of means using one of six methods: Tukey, Tukey-Kramer, Dunn-Sidak, Bonferroni, Scheffé, or Fisher's LSD (One-at-a-Time). If *Tukey* is specified, Tukey confidence intervals are calculated if the group sizes are equal; otherwise, the Tukey-Kramer confidence intervals are calculated.

TUKEY

Named variable into which the array containing the statistics relating to the difference of means is stored. On return, the named variable contains an array of size:

$$\binom{ngroups}{2} \times 5$$

where $ngroups = N\_ELEMENTS(n)$.

- 0—group number for the i -th mean
- 1—group number for the j -th mean
- 2—difference of means (i -th mean) – (j -th mean)
- 3—lower confidence limit for the difference
- 4—upper confidence limit for the difference

The `IMSL_ANOVA1` function computes confidence intervals on all pairwise differences of means using one of six methods: Tukey, Tukey-Kramer, Dunn-Sidak, Bonferroni, Scheffé, or Fisher's LSD (One-at-a-Time). If *Tukey* is specified, Tukey confidence intervals are calculated if the group sizes are equal; otherwise, the Tukey-Kramer confidence intervals are calculated.

Discussion

The `IMSL_ANOVA1` function performs an analysis of variance of responses from a one-way classification design. The model is:

$$y_{ij} = \mu_i + \epsilon_{ij} \quad i = 1, 2, \dots, k; \quad j = 1, 2, \dots, n_i$$

where the observed value y_{ij} constitutes the j -th response in the i -th group, μ_i denotes the population mean for the i -th group, and the ϵ_{ij} arguments are errors that are identically and independently distributed normal with mean 0 and variance σ^2 . The `IMSL_ANOVA1` function requires the y_{ij} observed responses as input into a single vector y with responses in each group occupying contiguous locations. The analysis of variance table is computed along with the group sample means and standard deviations. A discussion of formulas and interpretations for the one-way analysis of variance problem appears in most statistics texts, e.g., Snedecor and Cochran (1967, Chapter 10).

The `IMSL_ANOVA1` function computes simultaneous confidence intervals on all:

$$k' = \frac{k(k-1)}{2}$$

pairwise comparisons of k means $\mu_1, \mu_2, \dots, \mu_k$ in the one-way analysis of variance model. Any of several methods can be chosen. A good review of these methods is

given by Stoline (1981). The methods also are discussed in many statistics texts, e.g., Kirk (1982, pp. 114–127).

Let s^2 be the estimated variance of a single observation. Let v be the degrees of freedom associated with s^2 . Let:

$$\alpha = 1 - \frac{\text{Confidence}}{100.0}$$

The methods are summarized as follows:

Tukey method: The Tukey method gives the narrowest simultaneous confidence intervals for all pairwise differences of means $\mu_i - \mu_j$ in balanced ($n_1 = n_2 = \dots n_k = n$) one-way designs. The method is exact and uses the Studentized range distribution. The formula for the difference $\mu_i - \mu_j$ is given by the following:

$$\bar{y}_i - \bar{y}_j \pm q_{1-\alpha; k, v} \sqrt{\frac{s^2}{n}}$$

where $q_{1-\alpha; k, v}$ is the $(1 - \alpha)$ 100 percentage point of the Studentized range distribution with parameters k and v .

Tukey-Kramer method: The Tukey-Kramer method is an approximate extension of the Tukey method for the unbalanced case. (The method simplifies to the Tukey method for the balanced case.) The method always produces confidence intervals narrower than the Dunn-Sidak and Bonferroni methods. Hayter (1984) proved that the method is conservative, i.e., the method guarantees a confidence coverage of at least $(1 - \alpha)$ 100. Hayter's proof gave further support to earlier recommendations for its use (Stoline 1981). (Methods that are currently better are restricted to special cases and only offer improvement in severely unbalanced cases; see, for example, Spurrier and Isham 1985.) The formula for the difference $\mu_i - \mu_j$ is given by the following:

$$\bar{y}_i - \bar{y}_j \pm q_{1-\alpha; k, v} \sqrt{\frac{s^2}{2n_i} + \frac{s^2}{2n_j}}$$

Dunn-Sidak method: The Dunn-Sidak method is a conservative method. The method gives wider intervals than the Tukey-Kramer method. (For large v and small α and k , the difference is only slight.) The method is slightly better than the Bonferroni method and is based on an improved Bonferroni (multiplicative) inequality (Miller 1980, pp. 101, 254–255). The method uses the t distribution (see IMSL_TCDF). The formula for the difference $\mu_i - \mu_j$ is given by the following:

$$\bar{y}_i - \bar{y}_j \pm t_{\frac{1}{2} + \frac{1}{2}(1-\alpha)^{1/k'}, v} \sqrt{\frac{s^2}{n_i} + \frac{s^2}{n_j}}$$

where $t_{f;v}$ is the 100f percentage point of the t distribution with v degrees of freedom.

Bonferroni method: The Bonferroni method is a conservative method based on the Bonferroni (additive) inequality (Miller, p. 8). The method uses the t distribution. The formula for the difference $\mu_i - \mu_j$ is given by the following:

$$\bar{y}_i - \bar{y}_j \pm t_{1 - \frac{\alpha}{2k'}, v} \sqrt{\frac{s^2}{n_i} + \frac{s^2}{n_j}}$$

Scheffé method: The Scheffé method is an overly conservative method for simultaneous confidence intervals on pairwise difference of means. The method is applicable for simultaneous confidence intervals on all contrasts, i.e., all linear combinations:

$$\sum_{i=1}^k c_i \mu_i$$

where the following is true:

$$\sum_{i=1}^k c_i = 0$$

This method can be recommended here only if a large number of confidence intervals on contrasts, in addition to the pairwise differences of means, are to be constructed. The method uses the F distribution (see IMSL_FCDF). The formula for the difference $\mu_i - \mu_j$ is given by the following:

$$\bar{y}_i - \bar{y}_j \pm \sqrt{(k-1)F_{1-\alpha; k-1, v} \left(\frac{s^2}{n_i} + \frac{s^2}{n_j} \right)}$$

where:

$$F_{1-\alpha; k-1, v}$$

is the $(1 - \alpha)$ 100 percentage point of the F distribution with $k - 1$ and v degrees of freedom.

One-at-a-Time t method (Fisher's LSD): The One-at-a-Time t method is appropriate for constructing a single confidence interval. The confidence percentage input is appropriate for one interval at a time. The method has been used widely in conjunction with the overall test of the null hypothesis $\mu_1 = \mu_2 = \dots = \mu_k$ by the use of the F statistic. Fisher's LSD (least significant difference) test is a two-stage test that proceeds to make pairwise comparisons of means only if the overall F test is significant. Milliken and Johnson (1984, p. 31) recommend LSD comparisons after a significant F only if the number of comparisons is small and the comparisons were planned prior to the analysis. If many unplanned comparisons are made, they recommend Scheffé's method. If the F test is insignificant, a few planned comparisons for differences in means can still be performed by using either Tukey, Tukey-Kramer, Dunn-Sidak or Bonferroni methods. Because the F test is insignificant, Scheffé's method does not yield any significant differences. The formula for the difference $\mu_i - \mu_j$ is given by the following:

$$\bar{y}_i - \bar{y}_j \pm t_{1 - \frac{\alpha}{2}, v} \sqrt{\frac{s^2}{n_i} + \frac{s^2}{n_j}}$$

Examples

Example 1

This example computes a one-way analysis of variance for data discussed by Searle (1971, Table 5.1, pp. 165–179). The responses are plant weights for six plants of three different types shown in Table 16-1—three normal, two off-types, and one aberrant.

Normal	Off-Type	Aberrant
101	84	32
105	88	
94		

Table 16-1: Plant Types

```
n = [3,2,1]
y = [101.0, 105.0, 94.0, 84.0, 88.0, 32.0]
PRINT, 'p-value = ', IMSL_ANOVA1(n, y)

p-value = 0.00276887
```

Example 2: Multiple Comparisons

Simultaneous confidence intervals are generated for the measurements of cold-cranking power for five models of automobile batteries shown in [Table 16-2](#). Nelson (1989, pp. 232–241) provided the data and approach.

Model 1	Model 2	Model 3	Model 4	Model 5
41	42	27	48	28
43	43	26	45	32
42	46	28	51	37
46	38	27	46	25

Table 16-2: Cold-Cranking Power for Batteries

The Tukey method is chosen for the analysis of pairwise comparisons, with a confidence level of 99 percent. The means and their confidence limits are output. First, a procedure to print out the results is defined.

```
.RUN
PRO print_results, anova_table, diff_means
  anova_labels = ['df for among groups', $
    'df for within groups', 'total (corrected) df', $
    'ss for among groups', 'ss for within groups', $
    'total (corrected) ss', 'mean square among groups', $
    'mean square within groups', 'F-statistic', $
    'P-value', 'R-squared (in percent)', $
    'adjusted R-squared (in percent)', $
    'est. std of within group error', 'overall mean of y', $
    'coef. of variation (in percent)']
  PRINT, ' * *Analysis of Variance * *'
  FOR i = 0, 14 DO PM, anova_labels(i), $
    anova_table(i), FORMAT = '(a40,f20.2)'
  PRINT
  ; Print the analysis of variance table.
  PRINT, ' * *Differences of Means * *'
  PRINT, 'groups', 'difference', 'lower limit', 'upper limit'
  PM, diff_means, FORMAT = '(2i3, x, f9.2, 4x, f9.2, 5x, f9.2)'
  ; Print the differences of means.
END

n = [4, 4, 4, 4, 4]
y = [41, 43, 42, 46, 42, 43, 46, 38, 27, 26, 28, 27, $
    48, 45, 51, 46, 28, 32, 37, 25]
p_value = IMSL_ANOVA1(n, y, Confidence = 99.0, $
```

```

        Anova_Table = anova_table, Tukey = diff_means)
; Call IMSL_ANOVA1.
print_results, anova_table, diff_means

; Output the results.

* *Analysis of Variance * *
df for among groups                4.00
df for within groups               15.00
total (corrected) df               19.00
ss for among groups                1242.20
ss for within groups               150.75
total (corrected) ss               1392.95
mean square among groups           310.55
mean square within groups          10.05
F-statistic                        30.90
P-value                            0.00
R-squared (in percent)             89.18
adjusted R-squared (in percent)    86.29
est. std of within group error     3.17
overall mean of y                  38.05
coef. of variation (in percent)    8.33
* *Differences of Means * *
groups  difference  lower limit  upper limit
1 2      0.75      -8.05      9.55
1 3     16.00       7.20     24.80
1 4     -4.50    -13.30      4.30
1 5     12.50       3.70     21.30
2 3     15.25       6.45     24.05
2 4     -5.25    -14.05      3.55
2 5     11.75       2.95     20.55
3 4    -20.50    -29.30    -11.70
3 5     -3.50    -12.30      5.30
4 5     17.00       8.20     25.80

```

Version History

6.4	Introduced

IMSL_ANOVAFACT

The IMSL_ANOVAFACT function analyzes a balanced factorial design with fixed effects.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_ANOVAFACT(n_levels, y [, ANOVA_TABLE=variable]
    [, /DOUBLE] [, MEANS=variable] [, ORDER=value] [, /PURE_ERROR]
    [, /POOL_INTER] [, TEST_EFFECTS=variable])
```

Return Value

The p -value for the overall F -test.

Arguments

n_levels

One-dimensional array containing the number of levels for each of the factors and the number of replicates for each effect.

y

One-dimensional array of length:

$$n\_levels(0) * n\_levels(1) * \dots * ((N\_ELEMENTS(n\_levels) - 1))$$

containing the responses. Parameter *y* must not contain NaN for any of its elements, i.e., missing values are not allowed.

Keywords

ANOVA_TABLE

Named variable into which an array of size 15 containing the analysis of variance table is stored. The analysis of variance statistics are given as follows:

- 0—degrees of freedom for the model
- 1—degrees of freedom for error
- 2—total (corrected) degrees of freedom
- 3—sum of squares for the model
- 4—sum of squares for error
- 5—total (corrected) sum of squares
- 6—model mean square
- 7—error mean square
- 8—overall F -statistic
- 9— p -value
- 10— R^2 (in percent)
- 11—adjusted R^2 (in percent)
- 12—estimate of the standard deviation
- 13—overall mean of y
- 14—coefficient of variation (in percent)

DOUBLE

If present and nonzero, then double precision is used.

MEANS

Named variable into which an array of length $(n\_levels(0) + 1) \times (n\_levels(1) + 1) \times \dots \times (n\_levels(n-1) + 1)$ containing the subgroup means is stored.

See keyword *Test_Effects* for a definition of n . If the factors are A, B, C, and replicates, the ordering of the means is grand mean, A means, B means, C means, AB means, AC means, BC means, and ABC means.

ORDER

Number of factors included in the highest-way interaction in the model. *Order* must be in the interval $[1, N\_ELEMENTS(n\_levels) - 1]$. For example, an *Order* of 1 indicates that a main-effect model is analyzed, and an *Order* of 2 indicates that two-way interactions are included in the model. Default: *Order* = $N\_ELEMENTS(n\_levels) - 1$

PURE_ERROR

If present and nonzero, *Pure_Error* (the default option) indicates all the main effect and the interaction effects involving the replicates, the last element in *n_levels*, are pooled together to create the error term. The *Pool_Inter* option indicates (*Order* + 1)-way and higher-way interactions are pooled together to create the error. Keywords *Pure_Error* and *Pool_Inter* cannot be used together.

POOL_INTER

If present and nonzero, *Pure_Error* (the default option) indicates all the main effect and the interaction effects involving the replicates, the last element in *n_levels*, are pooled together to create the error term. The *Pool_Inter* option indicates (*Order* + 1)-way and higher-way interactions are pooled together to create the error. Keywords *Pure_Error* and *Pool_Inter* cannot be used together.

TEST_EFFECTS

Named variable into which an array of size *nef* × 4 containing statistics relating to the sums of squares for the effects in the model is stored. Here:

$$nef = \binom{n}{1} + \binom{n}{2} + \dots + \binom{n}{\min(n, |Order|)}$$

where *n* is given by *N_ELEMENTS(n_levels)* if *Pool_Inter* is specified; otherwise, *N_ELEMENTS(n_levels) - 1*.

Suppose the factors are A, B, C, and error. With *Order* = 3, rows 0 through *nef* - 1 correspond to A, B, C, AB, AC, BC, and ABC. The columns of *Test_Effects* are as follows:

- 0—degrees of freedom
- 1—sum of squares
- 2—*F*-statistic
- 3—*p*-value

Discussion

The *IMSL_ANOVAFACT* function performs an analysis for an *n*-way classification design with balanced data. For balanced data, there must be an equal number of responses in each cell of the *n*-way layout. The effects are assumed to be fixed effects. The model is an extension of the two-way model to include *n* factors. The interactions (two-way, three-way, up to *n*-way) can be included in the model, or some

of the higher-way interactions can be pooled into error. The keyword *Order* specifies the number of factors to be included in the highest-way interaction. For example, if three-way and higher-way interactions are to be pooled into error, set *Order* = 2.

By default, *Order* = *N_ELEMENTS* (*n_levels*) – 1 with the last subscript being the replicates subscript. Keyword *Pure_Error* indicates there are repeated responses within the *n*-way cell; *Pool_Inter* indicates otherwise.

The *IMSL_ANOVAFAC* function requires the responses as input into a single vector *y* in lexicographical order, so that the response subscript associated with the first factor varies least rapidly, followed by the subscript associated with the second factor, and so forth. Hemmerle (1967, Chapter 5) discusses the computational method.

Examples

Example 1

A two-way analysis of variance is performed with balanced data discussed by Snedecor and Cochran (1967, Table 12.5.1, p. 347). The responses are the weight gains (in grams) of rats that were fed diets varying in the source (A) and level (B) of protein.

The model is:

$$y_{ijk} = \mu + \alpha_i + \beta_j + \gamma_{ij} + \varepsilon_{ijk}$$

for $i = 0, 1$; $j = 0, 1, 2$; $k = 0, 1, \dots, 9$

where

$$\sum_{i=0}^1 \alpha_i = 0; \sum_{j=0}^2 \beta_j = 0; \sum_{i=0}^1 \gamma_{ij} = 0$$

for $j = 0, 1, 2$ and

$$\sum_{j=0}^2 \gamma_{ij} = 0$$

for $i = 0, 1$. The first responses in each cell in the two-way layout are given in [Table 16-3](#):

Protein Level (B)	Protein Source (A)		
	Beef	Cereal	Pork
High	73, 102, 118, 104, 81, 107, 100, 87, 117, 111	98, 74, 56, 111, 95, 88, 82, 77, 86, 92	94, 79, 96, 98, 102, 102, 108, 91, 120, 105
Low	90, 76, 90, 64, 86, 51, 72, 90, 95, 78	107, 95, 97, 80, 98, 74, 74, 67, 89, 58	49, 82, 73, 86, 81, 97, 106, 70, 61, 82

Table 16-3: Cell First Responses

```

n = [3, 2, 10]
y = [73.0, 102.0, 118.0, 104.0, 81.0, $
     107.0, 100.0, 87.0, 117.0, 111.0, $
     90.0, 76.0, 90.0, 64.0, 86.0, $
     51.0, 72.0, 90.0, 95.0, 78.0, $
     98.0, 74.0, 56.0, 111.0, 95.0, $
     88.0, 82.0, 77.0, 86.0, 92.0, $
     107.0, 95.0, 97.0, 80.0, 98.0, $
     74.0, 74.0, 67.0, 89.0, 58.0, $
     94.0, 79.0, 96.0, 98.0, 102.0, $
     102.0, 108.0, 91.0, 120.0, 105.0, $
     49.0, 82.0, 73.0, 86.0, 81.0, $
     97.0, 106.0, 70.0, 61.0, 82.0]
p_value = IMSL_ANOVAFAC(n, y, Anova_Table = anova_table)
PRINT, 'p-value = ', p_value

p-value =      0.00229943

```

Example 2: Two-way ANOVA

In this example, the same model and data are fit as in the initial example, but keywords are used for a more complete analysis. First, a procedure to output the results is defined.

```

.RUN
PRO print_results, anova_table, test_effects, means
  anova_labels = ['df for among groups', $
                 'df for within groups', 'total (corrected) df', $
                 'ss for among groups', 'ss for within groups', $
                 'total (corrected) ss', 'mean square among groups', $

```



```

    'mean square within groups', 'F-statistic', $
    'P-value', 'R-squared (in percent)', $
    'adjusted R-squared (in percent)', $
    'est. std of within group error', 'overall mean of y', $
    'coef. of variation (in percent)']
effects_labels = ['A ', 'B ', 'A*B']
means_labels = ['grand', 'A1', 'A2', $
    'A3', 'B1', 'B2', 'A1*B1', 'A1*B2', $
    'A2*B1', 'A2*B2', 'A3*B1', 'A3*B2']
PRINT, '          * *Analysis of Variance * *'
FOR i = 0, 14 DO PM, anova_labels(i), $
    anova_table(i), FORMAT = '(a40,f15.2)'
PRINT
; Print the analysis of variance table.
PRINT, '          * * Variation Due to the Model * *'
PRINT, 'Source      DF      SS      MS      P-value'
FOR i = 0, 2 DO PM, effects_labels(i), test_effects(i, *)
PRINT
PRINT, ' * * Subgroup Means * *'
FOR i = 0, 11 DO PM, means_labels(i), $
    means(i), FORMAT = '(a5,f15.2)'
END

n = [3, 2, 10]
y = [73.0, 102.0, 118.0, 104.0, 81.0, $
    107.0, 100.0, 87.0, 117.0, 111.0, $
    90.0, 76.0, 90.0, 64.0, 86.0, $
    51.0, 72.0, 90.0, 95.0, 78.0, $
    98.0, 74.0, 56.0, 111.0, 95.0, $
    88.0, 82.0, 77.0, 86.0, 92.0, $
    107.0, 95.0, 97.0, 80.0, 98.0, $
    74.0, 74.0, 67.0, 89.0, 58.0, $
    94.0, 79.0, 96.0, 98.0, 102.0, $
    102.0, 108.0, 91.0, 120.0, 105.0, $
    49.0, 82.0, 73.0, 86.0, 81.0, $
    97.0, 106.0, 70.0, 61.0, 82.0]
p_value = IMSL_ANOVAFAC(n, y, Anova_Table = anova_table, $
    Test_Effects = test_effects, Means = means)
print_results, anova_table, test_effects, means

* *Analysis of Variance * *
df for among groups          5.00
df for within groups        54.00
total (corrected) df        59.00
ss for among groups         4612.93
ss for within groups        11586.00
total (corrected) ss        16198.93
mean square among groups     922.59
mean square within groups    214.56

```

```

F-statistic                                4.30
P-value                                    0.00
R-squared (in percent)                    28.48
adjusted R-squared (in percent)           21.85
est. std of within group error            14.65
overall mean of y                         87.87
coef. of variation (in percent)           16.67
* * Variation Due to the Model * *
Source      DF      SS      MS      P-value
A           2.00000  266.533  0.621128  0.541132
B           1.00000  3168.27  14.7667   0.000322342
A*B         2.00000  1178.13  2.74552   0.0731880
* * Subgroup Means * *
grand              87.87
A1                 89.60
A2                 84.90
A3                 89.10
B1                 95.13
B2                 80.60
A1*B1              100.00
A1*B2              79.20
A2*B1              85.90
A2*B2              83.90
A3*B1              99.50
A3*B2              78.70

```

Example 3: Three-way ANOVA

This example performs a three-way analysis of variance using data discussed by John (1971, pp. 91–92). The responses are weights (in grams) of roots of carrots grown with varying amounts of applied nitrogen (A), potassium (B), and phosphorus (C). Each cell of the three-way layout has one response. Note that the ABC interactions sum of squares (186) is given incorrectly by John (1971, Table 5.2.)

The three-way layout is given in [Table 16-4](#):

	A_0			A_1			A_2		
	B_0	B_1	B_2	B_0	B_1	B_2	B_0	B_1	B_2
C_0	88.76	91.41	97.8	94.83	100.4	99.75	99.90	100.2	104.5
			5		9			3	1

Table 16-4: Three-way Layout

	A_0			A_1			A_2		
C_1	87.45	98.27	95.8 5	84.57	97.20	112.30	92.98	107.7 7	110.9 4
C_2	86.01	104.2 0	90.0 9	81.06	120.8 0	108.77	94.72	118.3 9	102.8 7

Table 16-4: Three-way Layout

```
.RUN
PRO print_results, anova_table, test_effects, means
  anova_labels = ['df for among groups', $
    'df for within groups', 'total (corrected) df', $
    'ss for among groups', 'ss for within groups', $
    'total (corrected) ss', 'mean square among groups', $
    'mean square within groups', 'F-statistic', $
    'P-value', 'R-squared (in percent)', $
    'adjusted R-squared (in percent)', $
    'est. std of within group error', $
    'overall mean of y', 'coef. of variation (in percent)']
  effects_labels = ['A ', 'B ', 'C ', 'A*B', 'A*B', 'A*C']
  PRINT, '          * *Analysis of Variance * *'
  FOR i = 0, 14 DO PM, anova_labels(i), $
    anova_table(i), FORMAT = '(a40,f15.2)'
  PRINT
  PRINT, '          * * Variation Due to the Model * *'
  PRINT, 'Source      DF      SS      MS      P-value'
  FOR i = 0,5 DO PM, effects_labels(i), test_effects(i, *)
END

n = [3, 3, 3]
y = [88.76, 87.45, 86.01, 91.41, 98.27, 104.20, 97.85, $
  95.85, 90.09, 94.83, 84.57, 81.06, 100.49, 97.20, $
  120.80, 99.75, 112.30, 108.77, 99.90, 92.98, 94.72, $
  100.23, 107.77, 118.39, 104.51, 110.94, 102.87]
p_value = IMSL_ANOVAFAC(n, y, Anova_Table = anova_table, $
  Test_Effects = test_effects, /Pool_Inter)
print_results, anova_table, test_effects

* *Analysis of Variance * *
df for among groups          18.00
df for within groups         8.00
total (corrected) df        26.00
ss for among groups         2395.73
ss for within groups         185.78
total (corrected) ss        2581.51
mean square among groups     133.10
```

mean square within groups	23.22			
F-statistic	5.73			
p-value	0.01			
R-squared (in percent)	92.80			
adjusted R-squared (in percent)	76.61			
est. std of within group error	4.82			
overall mean of y	98.96			
coef. of variation (in percent)	4.87			
* * Variation Due to the Model * *				
Source	DF	SS	MS	p-value
A	2.00000	488.368	10.5152	0.00576699
B	2.00000	1090.66	23.4832	0.000448704
C	2.00000	49.1484	1.05823	0.391063
A*B	4.00000	142.586	1.53502	0.280423
A*B	4.00000	32.3474	0.348241	0.838336
A*C	4.00000	592.624	6.37997	0.0131252

Version History

6.4	Introduced

IMSL_MULTICOMP

The IMSL_MULTICOMP function performs Student-Newman-Keuls multiple-comparisons test.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_MULTICOMP(means, df, std_error [, ALPHA=value]  
[, /DOUBLE])
```

Return Value

A one-dimensional array of length N_ELEMENTS(*means*) indicating the size of the groups of means declared to be equal. If the *i*-th element of the returned array is equal to *j*, then the *i*-th smallest mean and the next *j* – 1 larger means are declared equal. If the *i*-th element of the returned array is equal to 0, then no group of means starts with the *i*-th smallest mean.

Arguments

df

Degrees of freedom associated with *std_error*.

means

One-dimensional array containing the means.

std_error

Effective estimated standard error of a mean. In fixed effects models, *std_error* equals the estimated standard error of a mean.

For example, in a one-way model:

$$\text{std\_error} = \sqrt{\frac{s^2}{n}}$$

where s^2 is the estimate of σ^2 and n is the number of responses in a sample mean. In models with random components, use:

$$\text{std\_error} = \frac{\text{sedif}}{\sqrt{2}}$$

where *sedif* is the estimated standard error of the difference of two means.

Keywords

ALPHA

Significance level of test. Must be in the interval [0.01, 0.10]. Default: *Alpha* = 0.01

DOUBLE

If present and nonzero, then double precision is used.

Discussion

The IMSL_MULTICOMP function performs a multiple-comparison analysis of means using the Student-Newman-Keuls method. The null hypothesis is equality of all possible ordered subsets of a set of means. This null hypothesis is tested using the Studentized range of each of the corresponding subsets of sample means. The method is discussed in many elementary statistics texts, e.g., Kirk (1982, pp. 123–125).

Example

A multiple-comparisons analysis is performed using data discussed by Kirk. The results show that there are three groups of means with three separate sets of values: (36.7, 40.3, 43.4), (40.3, 43.4, 47.2), and (43.4, 47.2, 48.7).

```
df = 45
std_error = 1.6970563
means = [36.7, 48.7, 43.4, 47.2, 40.3]
equal_means = IMSL_MULTICOMP(means, df, std_error)
```

```
PM, equal_means, Title = 'Size of groups of means:'
```

```
Size of groups of means:
```

```
3
```

```
3
```

```
3
```

```
0
```

Version History

6.4	Introduced

IMSL_ANOVANESTED

The IMSL_ANOVANESTED function analyzes a completely nested random model with possibly unequal numbers in the subgroups.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_ANOVANESTED(n_factors, eq_option, n_levels, y  
[, ANOVA_TABLE=variable] [, CONFIDENCE=value] [, /DOUBLE]  
[, EMS=array] [, VAR_COMP=variable] [, Y_MEANS=array])
```

Return Value

The p -value for the F-statistic.

Arguments

eq_option

Equal numbers option.

- 0—Unequal numbers in the subgroups
- 1—Equal numbers in the subgroups

n_factors

Number of factors (number of subscripts) in the model, including error.

n_levels

One-dimensional array with the number of levels.

If $eq\_option = 1$, n_levels is of length $n\_factors$ and contains the number of levels for each of the factors. In this case, the additional variables listed in Table 16-5 are referred to in the description of IMSL_ANOVANESTED:

Variable	Description
LNL	$n\_levels(1) +$
	$\dots + n\_levels(0) * n\_levels(1) *$
	$\dots * n\_levels(n\_factors - 2)$
LNLNF	$n\_levels(0) * n\_levels(1) * \dots *$
	$n\_levels(n\_factors - 2)$
NOBS	The number of observations. NOBS equals
	$n\_levels(0) * n\_levels(1) * \dots *$
	$n\_levels(n\_factors-1)$

Table 16-5: Additional Variables

If $eq\_option = 0$, n_levels contains the number of levels of each factor at each level of the factor in which it is nested. In this case, the following additional variables are referred to in the description of IMSL_ANOVANESTED:

- *LNL*—Length of n_levels .
- *LNLNF*—Length of the subvector of n_levels for the last factor.
- *NOBS*—Number of observations. NOBS equals the sum of the last *LNLNF* elements of n_levels . $n\_levels(n\_factors-1)$.

For example, a random one-way model with two groups, five responses in the first group and ten in the second group, would have $LNL = 3$, $LNLNF = 2$, $NOBS = 15$, $n\_levels(0) = 2$, $n\_levels(1) = 5$, and $n\_levels(2) = 10$.

y

One-dimensional array of length NOBS containing the responses.

Keywords

ANOVA_TABLE

Named variable which stores the size 15 array containing the analysis of variance table. Analysis of variance statistics are as follows:

- 0—Degrees of freedom for the model
- 1—Degrees of freedom for error
- 2—Total (corrected) degrees of freedom
- 3—Sum of squares for the model
- 4—Sum of squares for error
- 5—Total (corrected) sum of squares
- 6—Model mean square
- 7—Error mean square
- 8—Overall F -statistic
- 9— p -value
- 10— R^2 (in percent)
- 11—Adjusted R^2 (in percent)
- 12—Estimate of the standard deviation
- 13—Overall mean of y
- 14—Coefficient of variation (in percent)

CONFIDENCE

Confidence level for two-sided interval estimates on the variance components, in percent. *Confidence* percent confidence intervals are computed, hence, *Confidence* must be in the interval [0.0, 100.0). *Confidence* often will be 90.0, 95.0, or 99.0. For one-sided intervals with confidence level ONECL, ONECL in the interval [50.0, 100.0), set *Confidence* = $100.0 - 2.0 * (100.0 - \text{ONECL})$. Default: *Confidence* = 95.0

DOUBLE

If present and nonzero, then double precision is used.

EMS

One-dimensional array of length $n\_factors * ((n\_factors + 1)/2)$ with expected mean square coefficients.

VAR_COMP

Named variable into which an array of size $n\_factors$ by 9 containing statistics relating to the particular variance components in the model is stored. Rows of *Var_Comp* correspond to the $n\_factors$ factors. Columns of *Var_Comp* are as follows:

- 1—Degrees of freedom
- 2—Sum of squares
- 3—Mean squares
- 4—F -statistic
- 5— p -value for F test
- 6—Variance component estimate
- 7—Percent of variance explained by variance component
- 8—Lower endpoint for confidence interval on the variance component
- 9—Upper endpoint for confidence interval on the variance component

If a test for error variance equal to zero cannot be performed, *Var_Comp*($n\_factors$, 4) and *Var_Comp*($n\_factors$, 5) are set to NaN.

Y_MEANS

One-dimensional array containing the subgroup means.

eq_option	Length of y means
0	$1 + n\_levels(0) + n\_levels(1) + \dots + n\_levels((LNL - LNLNF) - 1)$ (See description of argument <i>n_levels</i> for definitions of LNL and LNLNF.)
1	$1 + n\_levels(0) + n\_levels(0) * n\_levels(1) + \dots + n\_levels(0) * n\_levels(1) * \dots * n\_levels(n\_factors - 2)$

Table 16-6: *eq_option* for *Y_Means*

If the factors are labeled A , B , C , and error, the ordering of the means is grand mean, A means, AB means, and then ABC means.

Discussion

The `IMSL_ANOVANESTED` function analyzes a nested random model with equal or unequal numbers in the subgroups. The analysis includes an analysis of variance table and computation of subgroup means and variance component estimates. Anderson and Bancroft (1952, pages 325–330) discuss the methodology. The analysis of variance method is used for estimating the variance components. This method solves a linear system in which the mean squares are set to the expected mean squares. A problem that Hocking (1985, pages 324–330) discusses is that this method can yield negative variance component estimates. Hocking suggests a diagnostic procedure for locating the cause of a negative estimate. It may be necessary to reexamine the assumptions of the model.

Example

An analysis of a three-factor nested random model with equal numbers in the subgroups is performed using data discussed by Snedecor and Cochran (1967, Table 10.16.1, pages 285–288). The responses are calcium concentrations (in percent, dry basis) as measured in the leaves of turnip greens. Four plants are taken at random, then three leaves are randomly selected from each plant. Finally, from each selected leaf two samples are taken to determine calcium concentration. The model is:

$$y_{ijk} = \mu + \alpha_i + \beta_{ij} + e_{ijk} \quad i = 1, 2, 3, 4; j = 1, 2, 3; k = 1, 2$$

where y_{ijk} is the calcium concentration for the k -th sample of the j -th leaf of the i -th plant, the α_i 's are the plant effects and are taken to be independently distributed:

$$N(0, \sigma^2)$$

the β_{ij} 's are leaf effects each independently distributed:

$$N(0, \sigma_\beta^2)$$

and the ε_{ijk} 's are errors each independently distributed $N(0, \sigma^2)$. The effects are all assumed to be independently distributed. The data is given in [Table 16-7](#):

Plant	Leaf	Samples	
1	1	3.28	3.09
	2	3.52	3.48
	3	2.88	2.80
2	1	2.46	2.44
	2	1.87	1.92
	3	2.19	2.19
3	1	2.77	2.66
	2	3.74	3.44
	3	2.55	2.55
4	1	3.78	3.87
	2	4.07	4.12
	3	3.31	3.31

Table 16-7: Calcium Concentrations

```
.RUN
PRO print_results, p, at, ems, y_means, var_comp
  anova_labels = ['degrees of freedom for model', $
    'degrees of freedom for error', $
    'total (corrected) degrees of freedom', $
    'sum of squares for model', 'sum of squares for error', $
    'total (corrected) sum of squares', 'model mean square', $
    'error mean square', 'F-statistic', 'p-value', $
    'R-squared (in percent)', $
    'adjusted R-squared (in percent)', $
    'est. standard deviation of within error', $
    'overall mean of y', $
    'coefficient of variation (in percent)']
  ems_labels = ['Effect A and Error', 'Effect A and Effect B', $
    'Effect A and Effect A', 'Effect B and Error', $
    'Effect B and Effect B', 'Error and Error']
  components_labels = ['degrees of freedom for A', $
    'sum of squares for A', 'mean square of A', $
    'F-statistic for A', 'p-value for A', $
    'Estimate of A', 'Percent Variation Explained by A', $
    '95% Confidence Interval Lower Limit for A', $
```

```

'95% Confidence Interval Upper Limit for A', $
'degrees of freedom for B', 'sum of squares for B', $
'mean square of B', 'F-statistic for B', 'p-value for B', $
'Estimate of B', 'Percent Variation Explained by B', $
'95% Confidence Interval Lower Limit for B', $
'95% Confidence Interval Upper Limit for B', $
'degrees of freedom for Error', $
'sum of squares for Error', 'mean square of Error', $
'F-statistic for Error', 'p-value for Error', $
'Estimate of Error', 'Percent Explained by Error', $
'95% Confidence Interval Lower Limit for Error', $
'95% Confidence Interval Upper Limit for Error']
means_labels = ['Grand mean', $
  ' A means 1', $
  ' A means 2', $
  ' A means 3', $
  ' A means 4', $
  'AB means 1 1', $
  'AB means 1 2', $
  'AB means 1 3', $
  'AB means 2 1', $
  'AB means 2 2', $
  'AB means 2 3', $
  'AB means 3 1', $
  'AB means 3 2', $
  'AB means 3 3', $
  'AB means 4 1', $
  'AB means 4 2', $
  'AB means 4 3']
PRINT, 'p value of F statistic =', p
PRINT
PRINT, '          * * * Analysis of Variance * * *'
FOR i = 0, 14 DO $
  PM, anova_labels(i), at(i), FORMAT = '(A40, F20.5)'
PRINT
PRINT, '          * * * Expected Mean Square Coefficients * * *'
FOR i = 0, 5 DO $
  PM, ems_labels(i), ems(i), FORMAT = '(A40, F20.2)'
PRINT
PRINT, '          * * Analysis of Variance / Variance Components *
* '
k = 0
FOR i = 0, 2 DO BEGIN
  FOR j = 0, 8 DO BEGIN
    PM, components_labels(k), var_comp(i, j), $
    FORMAT = '(A45, F20.5)'
    k = k + 1
  ENDFOR
ENDFOR
ENDFOR

```

```

PRINT
PRINT, 'means', FORMAT = '(A20)'
FOR i = 0, 16 DO $
    PM, means_labels(i), y_means(i), FORMAT = '(A20, F20.2)'
END

y = [3.28, 3.09, 3.52, 3.48, 2.88, 2.80, 2.46, 2.44, 1.87, $
    1.92, 2.19, 2.19, 2.77, 2.66, 3.74, 3.44, 2.55, 2.55, $
    3.78, 3.87, 4.07, 4.12, 3.31, 3.31]
n_levels = [4, 3, 2]
p = IMSL_ANOVANESTED(3, 1, n_levels, y, Anova_Table = at, $
    Ems=ems, Y_Means = y_means, Var_Comp = var_comp)
print_results, p, at, ems, y_means, var_comp

p value of F statistic =      0.00000
    * * * Analysis of Variance * * *
        degrees of freedom for model      11.00000
        degrees of freedom for error      12.00000
    total (corrected) degrees of freedom  23.00000
        sum of squares for model          10.19054
        sum of squares for error          0.07985
    total (corrected) sum of squares      10.27040
        model mean square                 0.92641
        error mean square                 0.00665
            F-statistic                   139.21599
            p-value                       0.00000
        R-squared (in percent)            99.22248
    adjusted R-squared (in percent)        98.50976
    est. standard deviation of within error 0.08158
        overall mean of y                 3.01208
    coefficient of variation (in percent)   2.70826
    * * * Expected Mean Square Coefficients * * *
        Effect A and Error                1.00
        Effect A and Effect B              2.00
        Effect A and Effect A              6.00
        Effect B and Error                1.00
        Effect B and Effect B              2.00
            Error and Error                1.00
    * * Analysis of Variance / Variance Components * *
        degrees of freedom for A           3.00000
        sum of squares for A               7.56034
        mean square of A                   2.52011
        F-statistic for A                  7.66516
        p-value for A                     0.00973
        Estimate of A                     0.36522
    Percent Variation Explained by A       68.53015
    95% Confidence Interval Lower Limit for A 0.03955
    95% Confidence Interval Upper Limit for A 5.78674
        degrees of freedom for B           8.00000

```

sum of squares for B	2.63020
mean square of B	0.32878
F-statistic for B	49.40642
p-value for B	0.00000
Estimate of B	0.16106
Percent Variation Explained by B	30.22121
95% Confidence Interval Lower Limit for B	0.06967
95% Confidence Interval Upper Limit for B	0.60042
degrees of freedom for Error	12.00000
sum of squares for Error	0.07985
mean square of Error	0.00665
F-statistic for Error	NaN
p-value for Error	NaN
Estimate of Error	0.00665
Percent Explained by Error	1.24864
95% Confidence Interval Lower Limit for Error	0.00342
95% Confidence Interval Upper Limit for Error	0.01813
means	
Grand mean	3.01
A means 1	3.17
A means 2	2.18
A means 3	2.95
A means 4	3.74
AB means 1 1	3.18
AB means 1 2	3.50
AB means 1 3	2.84
AB means 2 1	2.45
AB means 2 2	1.89
AB means 2 3	2.19
AB means 3 1	2.72
AB means 3 2	3.59
AB means 3 3	2.55
AB means 4 1	3.82
AB means 4 2	4.10
AB means 4 3	3.31

Version History

6.4	Introduced

IMSL_ANOVABALANCED

The IMSL_ANOVABALANCED function analyzes a balanced complete experimental design for a fixed, random, or mixed model.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_ANOVABALANCED(n_levels, y, n_random, idx_rand_fct,  
    n_fct_per_eff, idx_fct_per_eff [, ANOVA_TABLE=variable]  
    [, CONFIDENCE=value] [, /DOUBLE] [, MODEL=value]  
    [, VAR_COMP=variable] [, Y_MEANS=variable])
```

Return Value

The p -value for the F -statistic.

Arguments

idx_fct_per_eff

One-dimensional index array of length $N\_ELEMENTS(n\_fct\_per\_effect)$. The first $n\_fct\_per\_eff(0)$ elements give the factor numbers in the first effect. The next $n\_fct\_per\_eff(1)$ elements give the factor numbers in the second effect. The last $n\_fct\_per\_eff(N\_ELEMENTS(n\_fct\_per\_eff))$ elements give the factor numbers in the last effect. Main effects must appear before their interactions. In general, an effect E cannot appear after an effect F if all of the indices for E appear also in F .

idx_rand_fct

One-dimensional index array of length $|n\_random|$ containing either the factor numbers to be considered random (for n_random positive) or containing the effect numbers to be considered random (for n_random negative).

n_fct_per_eff

One-dimensional array containing the number of factors associated with each effect in the model.

n_levels

One-dimensional array containing the number of levels for each of the factors.

n_random

For positive n_random , $|n\_random|$ is the number of random factors. For negative n_random , $|n\_random|$ is the number of random effects (sources of variation).

y

One-dimensional array containing the responses. y must not contain NaN (not a number) for any of its elements, i.e., missing values are not allowed.

Keywords**ANOVA_TABLE**

Named variable into which an array of size 15 containing the analysis of variance table is stored. The analysis of variance statistics are as follows:

- 0—Degrees of freedom for the model
- 1—Degrees of freedom for error
- 2—Total (corrected) degrees of freedom
- 3—Sum of squares for the model
- 4—Sum of squares for error
- 5—Total (corrected) sum of squares
- 6—Model mean square
- 7—Error mean square
- 8—Overall F -statistic
- 9— p -value
- 10— R^2 (in percent)
- 11—adjusted R^2 (in percent)

- 12—estimate of the standard deviation
- 13—overall mean of y
- 14—coefficient of variation (in percent)

CONFIDENCE

Confidence level for two-sided interval estimates on variance components, in percent. *Confidence* percent confidence intervals are computed, hence, *Confidence* must be in the interval $[0.0, 100.0]$. *Confidence* is often 90.0, 95.0, or 99.0. For one-sided intervals with confidence level α , α in the interval $[50.0, 100.0]$, set *Confidence* = $100.0 - 2.0 * (100.0 - \alpha)$. Default: *Confidence* = 95.0

DOUBLE

If present and nonzero, then double precision is used.

MODEL

Model Option

- 0—Searle model (Default)
- 1—Scheffe model

For Scheffe model, effects corresponding to interactions of fixed and random factors have their sum over the subscripts corresponding to fixed factors equal to zero. Also, the variance of a random interaction effect involving some fixed factors has a multiplier for the associated variance component that involves the number of levels in the fixed factors. The Searle model has no summation restrictions on the random interaction effects and has a multiplier of one for each variance component.

VAR_COMP

Named variable into which an array of length $N\_ELEMENTS(n\_fct\_per\_eff) + 1$, by 9 array containing statistics relating to the particular variance components or effects in the model and the error is stored. Rows of *Var_Comp* correspond to the rows of $N\_ELEMENTS(n\_fct\_per\_eff)$ effects plus error.

- 1—Degrees of freedom
- 2—Sum of squares
- 3—Mean squares
- 4— F -statistic
- 5— p -value for F test

- 6—Variance component estimate
- 7—Percent of variance of y explained by random effect
- 8—Lower endpoint for confidence interval on the variance component
- 9—Upper endpoint for confidence interval on the variance component

Columns 6 through 9 contain NaN (not a number) if the effect is fixed, i.e., if there is no variance component to be estimated. If the variance component estimate is negative, columns 8 and 9 contain NaN.

Ems—Named variable into which a one-dimensional array of length $((N\_ELEMENTS(n\_fct\_per\_eff) + 1) * (N\_ELEMENTS(n\_fct\_per\_eff) + 2)) / 2$ containing expected mean square coefficients is stored. Suppose the effects are A , B , and AB . The ordering of the coefficients in **Ems** is as follows:

	Error	AB	B	A
A	<i>Ems</i> (0)	<i>Ems</i> (1)	<i>Ems</i> (2)	<i>Ems</i> (3)
B	<i>Ems</i> (4)	<i>Ems</i> (5)	<i>Ems</i> (6)	
AB	<i>Ems</i> (7)	<i>Ems</i> (8)		
Error	<i>Ems</i> (9)			

Y_MEANS

Named variable into which a one-dimensional array of length $(n\_levels(0) + 1) * (n\_levels(1) + 1) * \dots * (n\_levels(n-1) + 1)$ containing the subgroup means is stored. Suppose the factors are A , B , and C . The ordering of the means is grand mean, A means, B means, C means, AB means, AC means, BC means, and ABC means.

Discussion

The **IMSL_ANOVABALANCED** function analyzes a balanced complete experimental design for a fixed, random, or mixed model. The analysis includes an analysis of variance table, and computation of subgroup means and variance component estimates. A choice of two parameterizations of the variance components for the model can be made.

Scheffé (1959, pages 274–289) discusses the parameterization for $Model = 1$. For example, consider the following model equation with fixed factor A and random factor B :

$$y_{ijk} = \mu + \alpha_i + b_j + c_{ij} + e_{ijk} \quad i = 1, 2, \dots, a; j = 1, 2, \dots, b; k = 1, 2, \dots, n$$

The fixed effects α_i 's are subject to the restriction:

$$\sum_{i=1}^a \alpha_{ij} = 0$$

the b_j 's are random effects identically and independently distributed:

$$N(0, \sigma_B^2)$$

c_{ij} are interaction effects each distributed:

$$N(0, \frac{a-1}{a} \sigma_{AB}^2)$$

and are subject to the restrictions:

$$\sum_{i=1}^a c_{ij} = 0 \quad \text{for } j = 1, 2, \dots, b$$

and the e_{ijk} 's are errors identically and independently distributed $N(0, \sigma^2)$. In general, interactions of fixed and random factors have sums over subscripts corresponding to fixed factors equal to zero. Also in general, the variance of a random interaction effect is the associated variance component times a product of ratios for each fixed factor in the random interaction term. Each ratio depends on the number of levels in the fixed factor. In the earlier example, the random interaction AB has the ratio $(a-1)/a$ as a multiplier of:

$$\sigma_{AB}^2$$

and:
$$\text{var}(y_{ijk}) = \sigma_B^2 + \frac{a-1}{a} \sigma_{AB}^2 + \sigma^2$$

In a three-way crossed classification model, an ABC interaction effect with A fixed, B random, and C fixed would have variance:

$$\frac{(a-1)(c-1)}{ac} \sigma_{ABC}^2$$

Searle (1971, pages 400–401) discusses the parameterization for $Model = 0$. This parameterization does not have the summation restrictions on the effects corresponding to interactions of fixed and random factors. Also, the variance of each random interaction term is the associated variance component, i.e., without the multiplier. This parameterization is also used with unbalanced data, which is one reason for its popularity with balanced data also. In the earlier example:

$$\text{var}(y_{ijk}) = \tilde{\sigma}_B^2 + \tilde{\sigma}_{AB}^2 + \sigma^2$$

Searle (1971, pages 400–404) compares these two parameterizations. Hocking (1973) considers these different parameterizations and concludes they are equivalent because they yield the same variance-covariance structure for the responses. Differences in covariances for individual terms, differences in expected mean square coefficients and differences in F tests are just a consequence of the definition of the individual terms in the model and are not caused by any fundamental differences in the models. For the earlier two-way model, Hocking states that the relations between the two parameterizations of the variance components are:

$$\sigma_B^2 = \tilde{\sigma}_B^2 + \frac{1}{a} \tilde{\sigma}_{AB}^2$$

$$\sigma_{AB}^2 = \tilde{\sigma}_{AB}^2$$

where:

$$\tilde{\sigma}_B^2 \text{ and } \tilde{\sigma}_{AB}^2$$

are the variance components in the parameterization with $Model = 0$.

Computations for degrees of freedom and sums of squares are the same regardless of the *Model* option. IMSL_ANOVABALANCED first computes degrees of freedom and sum of squares for a full factorial design. Degrees of freedom for effects in the factorial design that are missing from the specified model are pooled into the model effect containing the fewest subscripts but still containing the factorial effect. If no such model effect exists, the factorial effect is pooled into error. If more than one such effect exists, a terminal error message is issued indicating a misspecified model.

The analysis of variance method is used for estimating the variance components. This method solves a linear system in which the mean squares are set to the expected mean squares. A problem that Hocking (1985, pages 324–330) discusses is that this method can yield a negative variance component estimate. Hocking suggests a diagnostic procedure for locating the cause of the negative estimate. It may be necessary to re-examine the assumptions of the model.

The percentage of variation explained by each random effect is computed (output in *Var_Comp* element 7) as variance of the associated random effect divided by variance of y . The two parameterizations can lead to different values because of the different definitions of the individual terms in the model. For example, the percentage

associated with the AB interaction term in the earlier two-way mixed model is computed for $Model = 1$ using:

$$\% \text{ variation}(AB|Model = 1) = \frac{\frac{a-1}{a} \sigma_{AB}^2}{\sigma_B^2 + \frac{a-1}{a} \sigma_{AB}^2 + \sigma^2}$$

while for the parameterization $Model = 0$, the percentage is computed using the formula:

$$\% \text{ variation}(AB|Model = 0) = \frac{\tilde{\sigma}_{AB}^2}{\tilde{\sigma}_B^2 + \tilde{\sigma}_{AB}^2 + \sigma^2}$$

In each case, the variance components are replaced by their estimates (stored in *Var_Comp* element 6).

Confidence intervals on the variance components are computed using the method discussed by Graybill (1976, Theorem 15.3.5, page 624, and Note 4, page 620).

Example

An analysis of a generalized randomized block design is performed using data discussed by Kirk (1982, Table 6.10-1, pages 293–297). The model is:

$$y_{ijk} = \mu + \alpha_i + b_j + c_{ij} + e_{ijk} \quad i = 1, 2, 3, 4; j = 1, 2, 3, 4; k = 1, 2$$

where y_{ijk} is the response for k -th experimental unit in block j with treatment i ; the α_i 's are the treatment effects and are subject to the restriction:

$$\sum_{i=1}^2 \alpha_i = 0$$

the b_j 's are block effects identically and independently distributed:

$$N(0, \sigma_B^2)$$

c_{ij} are interaction effects each distributed:

$$N(0, \frac{3}{4} \sigma_{AB}^2)$$

and are subject to the restrictions:

$$\sum_{i=1}^4 c_{ij} = 0 \text{ for } j = 1, 2, 3, 4$$

and the e_{ijk} 's are errors, identically and independently distributed $N(0, \sigma^2)$. The interaction effects are assumed to be distributed independently of the errors. The data is given in Table 16-8.

Treatment	Block			
	1	2	3	4
1	3, 6	3, 1	2, 2	3, 2
2	4, 5	4, 2	3, 4	3, 3
3	7, 8	7, 5	6, 5	6, 6
4	7, 8	9, 10	10, 9	8, 11

Table 16-8: Randomized Block Design

```
.RUN
PRO print_results, p, at, ems, y_means, var_comp
anova_labels = ['degrees of freedom for model', $
'degrees of freedom for error', $
'total (corrected) degrees of freedom', $
'sum of squares for model', 'sum of squares for error', $
'total (corrected) sum of squares', 'model mean square', $
'error mean square', 'F-statistic', 'p-value', $
'R-squared (in percent)', $
'adjusted R-squared (in percent)', $
'est. standard deviation of within error', $
'overall mean of y', $
'coefficient of variation (in percent)']
ems_labels = ['Effect A and Error', $
'Effect A and Effect AB', 'Effect A and Effect B', $
'Effect A and Effect A', 'Effect B and Error', $
'Effect B and Effect AB', 'Effect B and Effect B', $
'Effect AB and Error', 'Effect AB and Effect AB', $
'Error and Error']
components_labels = ['degrees of freedom for A', $
'sum of squares for A', 'mean square of A', $
'F-statistic for A', 'p-value for A', $
'Estimate of A', 'Percent Variation Explained by A', $
'95% Confidence Interval Lower Limit for A', $
'95% Confidence Interval Upper Limit for A', $
'degrees of freedom for B', 'sum of squares for B', $
'mean square of B', 'F-statistic for B', 'p-value for B', $
'Estimate of B', 'Percent Variation Explained by B', $
'95% Confidence Interval Lower Limit for B', $
'95% Confidence Interval Upper Limit for B', $
```



```

'degrees of freedom for AB', 'sum of squares for AB', $
'mean square of AB', 'F-statistic for AB', $
'p-value for AB', 'Estimate of AB', $
'Percent Variation Explained by AB', $
'95% Confidence Interval Lower Limit for AB', $
'95% Confidence Interval Upper Limit for AB', $
'degrees of freedom for Error', $
'sum of squares for Error', 'mean square of Error', $
'F-statistic for Error', 'p-value for Error', $
'Estimate of Error', 'Percent Explained by Error', $
'95% Confidence Interval Lower Limit for Error', $
'95% Confidence Interval Upper Limit for Error']
means_labels = ['Grand mean', ' A means 1', ' A means 2', $
' A means 3', ' A means 4', ' B means 1', ' B means 2', $
' B means 3', ' B means 4', 'AB means 1 1', $
'AB means 1 2', 'AB means 1 3', 'AB means 1 4', $
'AB means 2 1', 'AB means 2 2', 'AB means 2 3', $
'AB means 2 4', 'AB means 3 1', 'AB means 3 2', $
'AB means 3 3', 'AB means 3 4', 'AB means 4 1', $
'AB means 4 2', 'AB means 4 3', 'AB means 4 4']
PRINT, 'p value of F statistic =', p
PRINT
PRINT, '          * * * Analysis of Variance * * *'
FOR i = 0, 14 DO $
    PM, anova_labels(i), at(i), FORMAT = '(A40, F20.5)'
PRINT
PRINT, '          * * * Expected Mean Square Coefficients * * *'
FOR i = 0, 9 DO $
    PM, ems_labels(i), ems(i), FORMAT = '(A40, F20.2)'
PRINT
PRINT, '          * * Analysis of Variance / Variance Components *
* '
k = 0
FOR i = 0, 3 DO BEGIN
    FOR j = 0, 8 DO BEGIN
        PM, components_labels(k), var_comp(i, j), $
        FORMAT = '(A45, F20.5)'
        k = k + 1
    ENDFOR
ENDFOR
PRINT
PRINT, 'means', FORMAT = '(A20)'
FOR i = 0, 24 DO $
    PM, means_labels(i), y_means(i), FORMAT = '(A20, F20.2)'
END

y = [3.0, 6.0, 3.0, 1.0, 2.0, 2.0, 3.0, 2.0, 4.0, 5.0, 4.0, $
2.0, 3.0, 4.0, 3.0, 3.0, 7.0, 8.0, 7.0, 5.0, 6.0, 5.0, $
6.0, 6.0, 7.0, 8.0, 9.0, 10.0, 10.0, 9.0, 8.0, 11.0]

```

```

n_levels = [4, 4, 2]
indrf = [2, 3]
nfef = [1, 1, 2]
indef = [1, 2, 1, 2]
p = IMSL_ANOVABALANCED(n_levels, y, 2, indrf, nfef, indef, $
    Anova_Table = at, Ems = ems, Y_Means = y_means, $
    Var_Comp = var_comp)
print_results, p, at, ems, y_means, var_comp

```

```

p value of F statistic = 4.94719e-06

```

```

* * * Analysis of Variance * * *
degrees of freedom for model          15.00000
degrees of freedom for error          16.00000
total (corrected) degrees of freedom  31.00000
sum of squares for model              216.50000
sum of squares for error              19.00000
total (corrected) sum of squares      235.50000
model mean square                     14.43333
error mean square                     1.18750
F-statistic                           12.15439
p-value                               0.00000
R-squared (in percent)                91.93206
adjusted R-squared (in percent)       84.36836
est. standard deviation of within error 1.08972
overall mean of y                     5.37500
coefficient of variation (in percent) 20.27395

```

```

* * * Expected Mean Square Coefficients * * *
Effect A and Error                    1.00
Effect A and Effect AB                2.00
Effect A and Effect B                 0.00
Effect A and Effect A                 8.00
Effect B and Error                    1.00
Effect B and Effect AB                2.00
Effect B and Effect B                 8.00
Effect AB and Error                   1.00
Effect AB and Effect AB               2.00
Error and Error                       1.00

```

```

* * Analysis of Variance / Variance Components * *
degrees of freedom for A              3.00000
sum of squares for A                 194.50000
mean square of A                     64.83334
F-statistic for A                    32.87324
p-value for A                        0.00004
Estimate of A                        NaN
Percent Variation Explained by A      NaN
95% Confidence Interval Lower Limit for A NaN

```

```

95% Confidence Interval Upper Limit for A           NaN
degrees of freedom for B                          3.00000
sum of squares for B                              4.25000
mean square of B                                  1.41667
F-statistic for B                                 0.71831
p-value for B                                     0.56566
Estimate of B                                    -0.06944
Percent Variation Explained by B                  0.00000
95% Confidence Interval Lower Limit for B           NaN
95% Confidence Interval Upper Limit for B           NaN
degrees of freedom for AB                          9.00000
sum of squares for AB                             17.75000
mean square of AB                                 1.97222
F-statistic for AB                                1.66082
p-value for AB                                    0.18016
Estimate of AB                                    0.39236
Percent Variation Explained by AB                  24.83516
95% Confidence Interval Lower Limit for AB           0.00000
95% Confidence Interval Upper Limit for AB           2.75803
degrees of freedom for Error                       16.00000
sum of squares for Error                           19.00000
mean square of Error                              1.18750
F-statistic for Error                             NaN
p-value for Error                                 NaN
Estimate of Error                                 1.18750
Percent Explained by Error                         75.16483
95% Confidence Interval Lower Limit for Error        0.65868
95% Confidence Interval Upper Limit for Error
42.75057
means
Grand mean                                         5.38
A means 1                                          2.75
A means 2                                          3.50
A means 3                                          6.25
A means 4                                          9.00
B means 1                                          6.00
B means 2                                          5.12
B means 3                                          5.12
B means 4                                          5.25
AB means 1 1                                       4.50
AB means 1 2                                       2.00
AB means 1 3                                       2.00
AB means 1 4                                       2.50
AB means 2 1                                       4.50
AB means 2 2                                       3.00
AB means 2 3                                       3.50
AB means 2 4                                       3.00
AB means 3 1                                       7.50
AB means 3 2                                       6.00

```

AB means 3 3	5.50
AB means 3 4	6.00
AB means 4 1	7.50
AB means 4 2	9.50
AB means 4 3	9.50
AB means 4 4	9.50

```
; Add Outliners
x(0, 1) = 100.0
x(3, 4) = 100.0
x(99, 2) = -100.0
p_cov = IMSL_POOLED_COV(x, n_groups, Idx_Vars = idxv, $
    Idx_Cols = idxc)
PM, p_cov, Title = 'Pooled Cavariance with Outliners'
r_cov = IMSL_ROBUST_COV(x, n_groups, Idx_Vars = idxv, $
    Idx_Cols = idxc, Percentage = percentage)
PM, r_cov, Title = 'Robust Covariance with Outliners'
```

Pooled Cavariance with Outliners

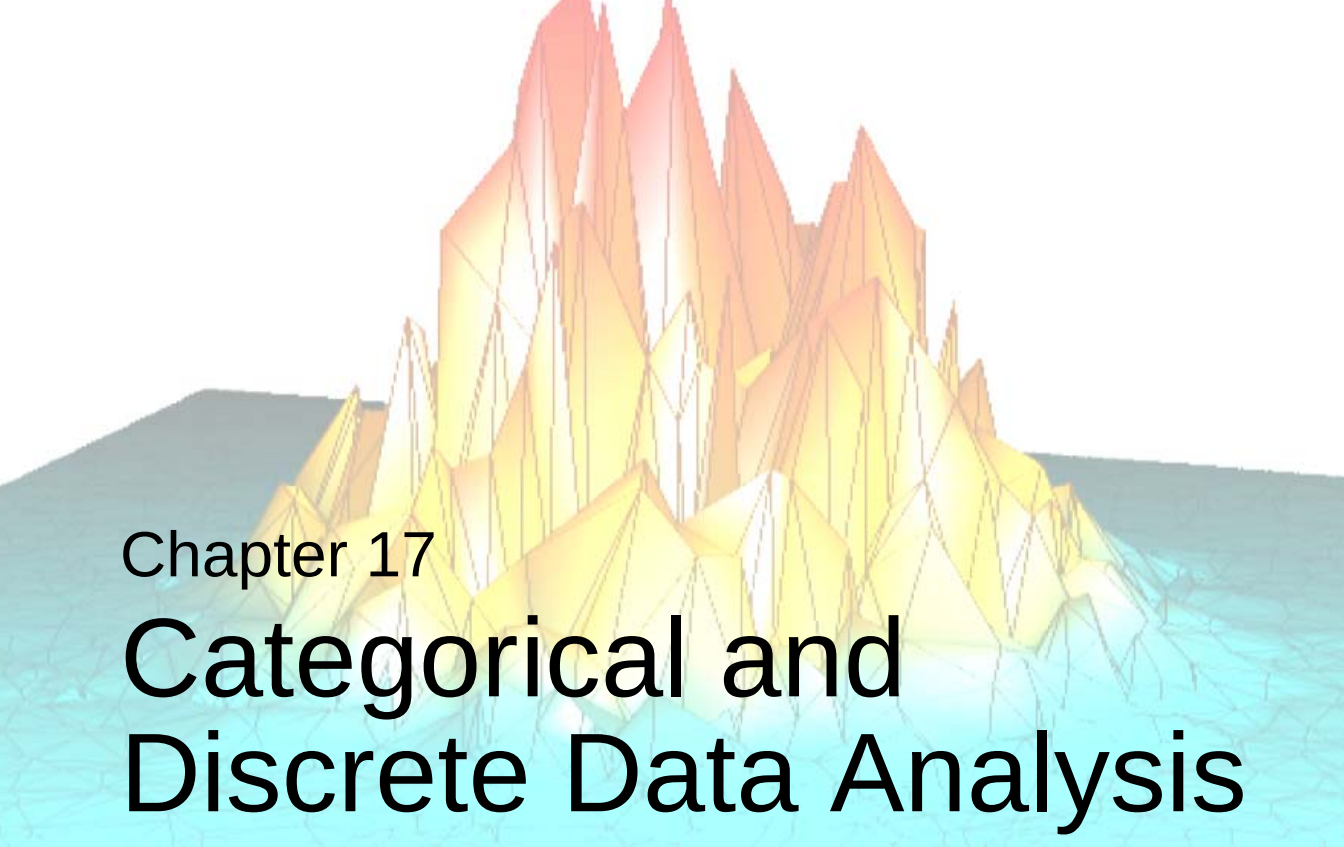
60.4264	0.304244	0.127488	-1.55551
0.304244	70.5257	0.167135	-0.171791
0.127488	0.167135	0.185188	0.0684639
-1.55551	-0.171791	0.0684639	66.3798

Robust Covariance with Outliners

0.255521	0.0876029	0.155279	0.0359198
0.0876029	0.112674	0.0545391	0.0322426
0.155279	0.0545391	0.172263	0.0412149
0.0359198	0.0322426	0.0412149	0.0424182

Version History

6.4	Introduced



Chapter 17

Categorical and Discrete Data Analysis

This section contains the following topics:

Overview: Categorical and Discrete Data Analysis	794	Categorical and Discrete Data Analysis Routines	795

Overview: Categorical and Discrete Data Analysis

Routine `IMSL_CONTINGENCY` computes many statistics of interest in a two-way table. Statistics computed by this routine includes the usual chi-squared statistics, measures of association, Kappa, and many others. Exact probabilities for two-way tables can be computed by `IMSL_EXACT_ENUM`, but this routine uses the total enumeration algorithm and, thus, often uses orders of magnitude more computer time than `IMSL_EXACT_NETWORK` which computes the same probabilities by use of the network algorithm (but can still be quite expensive).

The routine `IMSL_CAT_GLM` in the second section is concerned with generalized linear models (see McCullagh and Nelder 1983) in discrete data. This routine can be used to compute estimates and associated statistics in probit, logistic, minimum extreme value, Poisson, negative binomial (with known number of successes), and logarithmic models. Classification variables as well as weights, frequencies and additive constants may be used so that general linear models can be fit. Residuals, a measure of influence, the coefficient estimates, and other statistics are returned for each model fit. When infinite parameter estimates are required, extended maximum likelihood estimation may be used. Log-linear models can be fit in `IMSL_CAT_GLM` through the use of Poisson regression models. Results from Poisson regression models involving structural and sampling zeros will be identical to the results obtained from the log-linear model routines but will be fit by a quasi-Newton algorithm rather than through iterative proportional fitting.

Categorical and Discrete Data Analysis Routines

Statistics in the Two-Way Contingency Table

[IMSL_CONTINGENCY](#)—Two-way contingency table analysis.

[IMSL_EXACT_ENUM](#)—Exact probabilities in a table; total enumeration.

[IMSL_EXACT_NETWORK](#)—Exact probabilities in a table.

Generalized Categorical Models

[IMSL_CAT_GLM](#)—Generalized linear models.

IMSL_CONTINGENCY

The IMSL_CONTINGENCY function performs a chi-squared analysis of a two-way contingency table.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_CONTINGENCY(table [, CHI_SQ_CONTRIB=variable]  
    [, CHI_SQ_STATS=variable] [, CHI_SQ_TEST=variable] [, /DOUBLE]  
    [, EXPECTED=variable] [, LRT=variable] [, TABLE_STATS=variable] )
```

Return Value

Pearson chi-squared p -value for independence of rows and columns.

Arguments

table

Two-dimensional array containing the observed counts in the contingency table.

Keywords

CHI_SQ_CONTRIB

Named variable into which a two-dimensional array of size (n_rows+1) by $(n\_columns+1)$ containing the contributions for each cell in the table is stored. The contributions to chi-squared for each cell in the table is in the first n_rows rows and $n\_columns$ columns. The last row and column contain the total contribution to chi-squared for that row or column.

CHI_SQ_STATS

Named variable into which an array of length 5 containing chi-squared statistics associated with this contingency table is stored. The last three elements are based on

Pearson's chi-squared statistic (see *Chi_Sq_Test*). The chi-squared statistics are given as follows:

- 0—exact mean
- 1—exact standard deviation
- 2—phi
- 3—contingency coefficient
- 4—Cramer's V

CHI_SQ_TEST

Named variable into which the three-element array containing statistics associated with the chi-squared tests is stored. The first element contains the degrees of freedom for the chi-squared tests associated with the table, the second element contains the Pearson chi-squared test statistic, and the third element contains the probability of a larger Pearson chi-squared, p -value.

DOUBLE

If present and nonzero, double precision is used.

EXPECTED

Named variable into which the two-dimensional array of size (n_rows+1) by $(n\_columns+1)$ containing the expected values of each cell in the table is stored, where $n\_rows=(N\_ELEMENTS(table(*,0))$ and $n\_columns=(N\_ELEMENTS(table(0,*)))$. The expected values are computed under the null hypothesis and stored in the first n_rows rows and $n\_columns$ columns. The marginal totals are in the last row and column.

LRT

Named variable into which the three-element array containing statistics associated with the likelihood ratio G-squared tests is stored. The first element contains the degrees of freedom for the chi-squared tests associated with the table, the second element contains the likelihood ratio G^2 (chi-squared), and the third element contains the probability of a larger G^2 .

TABLE_STATS

Named variable into which a two-dimensional array of size 23×5 containing statistics associated with this table is stored. Each row corresponds to a statistic, as shown in [Table 17-1](#).

Row	Statistic
0	Gamma
1	Kendall's τ_b
2	Stuart's τ_c
3	Somers' D for rows (given columns)
4	Somers' D for columns (given rows)
5	product moment correlation
6	Spearman rank correlation
7	Goodman and Kruskal τ for rows (given columns)
8	Goodman and Kruskal τ for columns (given rows)
9	uncertainty coefficient U (symmetric)
10	uncertainty $U_{r c}$ (rows)
11	uncertainty $U_{c r}$ (columns)
12	optimal prediction λ (symmetric)
13	optimal prediction $\lambda_{r c}$ (rows)
14	optimal prediction $\lambda_{c r}$ (columns)
15	optimal prediction $\lambda_{r c}$ (rows)
16	optimal prediction $\lambda_{c r}$ (columns)
17	test for linear trend in row probabilities if $n\_rows = 2$. If n_rows is not 2, a test for linear trend in column probabilities if $n\_columns = 2$.
18	Kruskal-Wallis test for no-row effect
19	Kruskal-Wallis test for no-column effect
20	kappa (square tables only)

Table 17-1: Row Statistics

Row	Statistic
21	McNemar test of symmetry (square tables only)
22	McNemar one degree of freedom test of symmetry (square tables only)

Table 17-1: Row Statistics (Continued)

If a statistic cannot be computed or if some value is not relevant for the computed statistic, the entry is NaN (Not a Number). The columns are as follows:

- 0—estimated statistic
- 1—standard error for any parameter value
- 2—standard error under the null hypothesis
- 3— t value for testing the null hypothesis
- 4— p -value of the test in column 3

In the McNemar tests, Column 0 contains the statistic, Column 1 contains the chi-squared degrees of freedom, Column 3 contains the exact p -value (1 degree of freedom only), and Column 4 contains the chi-squared asymptotic p -value. The Kruskal-Wallis test is the same except no exact p -value is computed.

Discussion

The `IMSL_CONTINGENCY` function computes statistics associated with an $r \times c$ contingency table. The function computes the chi-squared test of independence, expected values, contributions to chi-squared, row and column marginal totals, some measures of association, correlation, prediction, uncertainty, the McNemar test for symmetry, a test for linear trend, the odds and the log odds ratio, and the kappa statistic (if the appropriate keywords are selected).

Notation

Let x_{ij} denote the observed cell frequency in the ij cell of the table and n denote the total count in the table. Let $p_{ij} = p_i \cdot p_j$ denote the predicted cell probabilities under the null hypothesis of independence, where $p_i \cdot$ and $p \cdot_j$ are the row and column marginal relative frequencies. Next, compute the expected cell counts as $e_{ij} = np_{ij}$.

Also required in the following are a_{uv} and b_{uv} for u , where $v = 1, \dots, n$. Let (r_s, c_s) denote the row and column response of observation s . Then, $a_{uv} = 1, 0$, or -1 ,

depending on whether $r_u < r_v$, $r_u = r_v$, or $r_u > r_v$. The b_{uv} similarly defined in terms of the c_s variables.

Chi-squared Statistic

For each cell in the table, the contribution to χ^2 is given as $(x_{ij} - e_{ij})^2/e_{ij}$. The Pearson chi-squared statistic (denoted χ^2) is computed as the sum of the cell contributions to chi-squared. It has $(r - 1)(c - 1)$ degrees of freedom and tests the null hypothesis of independence, i.e., $H_0: p_{ij} = p_i \cdot p_j$. The null hypothesis is rejected if the computed value of χ^2 is too large.

The maximum likelihood equivalent of χ^2 , G^2 is computed as follows:

$$G^2 = -2 \sum_{i,j} x_{ij} \ln(x_{ij}/np_{ij})$$

G^2 is asymptotically equivalent to χ^2 and tests the same hypothesis with the same degrees of freedom.

Measures Related to Chi-squared (Phi, Contingency Coefficient, and Cramer's V)

There are three measures related to chi-squared that do not depend on sample size:

- phi,

$$\phi = \sqrt{\chi^2/n}$$

- contingency coefficient,

$$P = \sqrt{\chi^2/(n + \chi^2)}$$

- Cramer's V,

$$V = \sqrt{\chi^2/(n \min(r, c))}$$

Since these statistics do not depend on sample size and are large when the hypothesis of independence is rejected, they can be thought of as measures of association and can be compared across tables with different sized samples. While both P and V have a range between 0.0 and 1.0, the upper bound of P is actually somewhat less than 1.0 for any given table (see Kendall and Stuart 1979, p. 587). The significance of all three statistics is the same as that of the χ^2 statistic, *Chi_Sq_Test*.

The distribution of the χ^2 statistic in finite samples approximates a chi-squared distribution. To compute the exact mean and standard deviation of the χ^2 statistic,

Haldane (1939) uses the multinomial distribution with fixed-table marginals. The exact mean and standard deviation generally differ little from the mean and standard deviation of the associated chi-squared distribution.

Standard Errors and p-values for Some Measures of Association

In Columns 1 through 4 of statistics, estimated standard errors and asymptotic p -values are reported. Estimates of the standard errors are computed in two ways. The first estimate, in Column 1 of the array `table_stats`, is asymptotically valid for any value of the statistic. The second estimate, in Column 2 of the array, is only correct under the null hypothesis of no association. The z-scores in Column 3 of statistics are computed using this second estimate of the standard errors. The p -values in column 4 are computed from this z-score. See Brown and Benedetti (1977) for a discussion and formulas for the standard errors in Column 2.

Measures of Association for Ranked Rows and Columns

The measures of association, ϕ , P , and V , do not require any ordering of the row and column categories. The `IMSL_CONTINGENCY` function also computes several measures of association for tables in which the row and column categories correspond to ranked observations. Two of these tests, the product moment correlation and the Spearman correlation, are correlation coefficients computed using assigned scores for the row and column categories. The cell indices are used for the product-moment correlation, while the average of the tied ranks of the row and column marginals is used for the Spearman rank correlation. Other scores are possible.

Gamma, Kendall's τ_b , Stuart's τ_c , and Somers' D are measures of association that are computed like a correlation coefficient in the numerator. In all these measures, the numerator is computed as the "covariance" between the a_{uv} variables and b_{uv} variables defined above, i.e., as follows:

$$\sum_u \sum_v a_{uv} b_{uv}$$

Recall that a_{uv} and b_{uv} can take values -1 , 0 , or 1 . Since the product $a_{uv}b_{uv} = 1$ only if a_{uv} and b_{uv} are both 1 or both -1 , it is easy to show that this "covariance" is twice the total number of agreements minus the number of disagreements, where a disagreement occurs when $a_{uv}b_{uv} = -1$.

Kendall's τ_b is computed as the correlation between a_{uv} and b_{uv} variables (see Kendall and Stuart 1979, p. 593). In a rectangular table ($r \neq c$), Kendall's τ_b cannot be 1.0 (if all marginal totals are positive). For this reason, Stuart suggested a

modification to the denominator of τ in which the denominator becomes the largest possible value of the “covariance.” This maximizing value is approximately $n^2 m / (m - 1)$, where $m = \min(r, c)$. Stuart’s τ_c uses this approximate value in its denominator. For large n :

$$\tau_c \approx m\tau_b / (m - 1)$$

Gamma can be motivated in a slightly different manner. Because the “covariance” of the a_{uv} variables and the b_{uv} variables can be thought of as twice the number of agreements minus the disagreements, $2(A - D)$, where A is the number of agreements and D is the number of disagreements, Gamma is motivated as the probability of agreement minus the probability of disagreement, given that either agreement or disagreement occurred. This is shown as $\gamma = (A - D)/(A + D)$.

Two definitions of Somers’ D are possible, one for rows and a second for columns. Somers’ D for rows can be thought of as the regression coefficient for predicting a_{uv} from b_{uv} . Moreover, Somer’s D for rows is the probability of agreement minus the probability of disagreement, given that the column variable, b_{uv} , is not 0. Somers’ D for columns is defined in a similar manner.

A discussion of all of the measures of association in this section can be found in Kendall and Stuart (1979, p. 592).

Measures of Prediction and Uncertainty

Optimal Prediction Coefficients: The measures in this section do not require any ordering of the row or column variables. They are based entirely upon probabilities. Most are discussed in Bishop et al. (1975, p. 385).

Consider predicting (or classifying) the column for a given row in the table. Under the null hypothesis of independence, choose the column with the highest column marginal probability for all rows. In this case, the probability of misclassification for any row is 1 minus this marginal probability. If independence is not assumed, then within each row, choose the column with the highest row-conditional probability. The probability of misclassification for the row becomes 1 minus this conditional probability.

Define the optimal prediction coefficient $\lambda_{c|r}$ for predicting columns from rows as the proportion of the probability of misclassification that is eliminated because the random variables are not independent. It is estimated by:

$$\lambda_{c|r} = \frac{(1 - p_{\bullet m}) - \left(1 - \sum_i p_{im}\right)}{1 - p_{\bullet m}}$$

where m is the index of the maximum estimated probability in the row (p_{im}) or row margin ($p_{.m}$). A similar coefficient is defined for predicting the rows from the columns. The symmetric version of the optimal prediction λ is obtained by summing the numerators and denominators of $\lambda_{r|c}$ and $\lambda_{c|r}$, then dividing. Standard errors for these coefficients are given in Bishop et al. (1975, p. 388).

A problem with the optimal prediction coefficients λ is that they vary with the marginal probabilities. One way to correct this is to use row-conditional probabilities. The optimal prediction λ^* coefficients are defined as the corresponding λ coefficients in which first the row (or column) marginals are adjusted to the same number of observations. This yields:

$$\lambda_{c|r}^* = \frac{\sum_i \max_j p_{j|i} - \max_j \left(\sum_i p_{j|i} \right)}{R - \max_j \left(\sum_i p_{j|i} \right)}$$

where i indexes the rows, j indexes the columns, and $p_{j|i}$ is the (estimated) probability of column j given row i . $\lambda_{r|c}^*$ is similarly defined.

Goodman and Kruskal τ : A second kind of prediction measure attempts to explain the proportion of the explained variation of the row (column) measure given the column (row) measure. Define the total variation in the rows as follows:

$$n/2 - \left(\sum_i x_{i.}^2 \right) / (2n)$$

Note that this is $1 / (2n)$ times the sums of squares of the a_{uv} variables.

With this definition of variation, the Goodman and Kruskal τ coefficient for rows is computed as the reduction of the total variation for rows accounted for by the columns, divided by the total variation for the rows. To compute the reduction in the total variation of the rows accounted for by the columns, note that the total variation for the rows within column j is defined as follows:

$$q_j = x_{.j}^2 - \left(\sum_i x_{ij}^2 \right) / (2x_{i.})$$

The total variation for rows within columns is the sum of the q_j variables. Consistent with the usual methods in the analysis of variance, the reduction in the total variation is given as the difference between the total variation for rows and the total variation for rows within the columns.

Goodman and Kruskal's τ for columns is similarly defined. See Bishop et al. (1975, p. 391) for the standard errors.

Uncertainty Coefficients: The uncertainty coefficient for rows is the increase in the log-likelihood that is achieved by the most general model over the independence model, divided by the marginal log-likelihood for the rows. This is given by the following equation:

$$U_{r|c} = \frac{\sum_{i,j} x_{ij} \log(x_{i\cdot} x_{\cdot j} / n x_{ij})}{\sum_i x_{i\cdot} \log(x_{i\cdot} / n)}$$

The uncertainty coefficient for columns is similarly defined. The symmetric uncertainty coefficient contains the same numerator as $U_{r|c}$ and $U_{c|r}$ but averages the denominators of these two statistics. Standard errors for U are given in Brown (1983).

Kruskal-Wallis: The Kruskal-Wallis statistic for rows is a one-way analysis-of-variance-type test that assumes the column variable is monotonically ordered. It tests the null hypothesis that no row populations are identical, using average ranks for the column variable. The Kruskal-Wallis statistic for columns is similarly defined. Conover (1980) discusses the Kruskal-Wallis test.

Test for Linear Trend: When there are two rows, it is possible to test for a linear trend in the row probabilities if it is assumed that the column variable is monotonically ordered. In this test, the probabilities for row 1 are predicted by the column index using weighted simple linear regression. This slope is given by:

$$\hat{\beta} = \frac{\sum_j x_{\cdot j} (x_{1j} / x_{\cdot j} - x_{1\cdot} / n) (j - \bar{j})}{\sum_j x_{\cdot j} (j - \bar{j})^2}$$

where:

$$\bar{j} = \sum_j x_{\cdot j} j / n$$

is the average column index. An asymptotic test that the slope is zero may then be obtained (in large samples) as the usual regression test of zero slope.

In two-column data, a similar test for a linear trend in the column probabilities is computed. This test assumes that the rows are monotonically ordered.

Kappa: Kappa is a measure of agreement computed on square tables only. In the kappa statistic, the rows and columns correspond to the responses of two judges. The judges agree along the diagonal and disagree off the diagonal. Let:

$$p_0 = \sum_i x_{ii} / n$$

denote the probability that the two judges agree, and let

$$p_c = \sum_i e_{ii}$$

denote the expected probability of agreement under the independence model. Kappa is then given by $(p_0 - p_c)/(1 - p_c)$.

McNemar Tests: The McNemar test is a test of symmetry in a square contingency table. In other words, it is a test of the null hypothesis $H_0: \theta_{ij} = \theta_{ji}$. The multiple degrees-of-freedom version of the McNemar test with $r(r-1)/2$ degrees of freedom is computed as follows:

$$\sum_{i < j} \frac{(x_{ij} - x_{ji})^2}{(x_{ij} + x_{ji})}$$

The single degree-of-freedom test assumes that the differences, $x_{ij} - x_{ji}$, are all in one direction. The single degree-of-freedom test is more powerful than the multiple degrees-of-freedom test when this is the case. The test statistic is given as follows:

$$\frac{\left(\sum_{i < j} (x_{ij} - x_{ji}) \right)^2}{\sum_{i < j} (x_{ij} + x_{ji})}$$

The exact probability can be computed by the binomial distribution.

Examples

Example 1

The following example, taken from Kendall and Stuart (1979), involves the distance vision in the right and left eyes. Output contains only the p -value.

```
table = [[821,116,72,43], [112,494,151,34], $
         [85,145,583,106], [35,27,87,331]]
print, 'P-Value', IMSL_CONTINGENCY(table)
```

Example 2

The following example, which illustrates the use of Kappa and McNemar tests, uses the same distance vision data as the previous example. The available statistics are obtained using keywords. First, a procedure is defined to output the results.

```

.RUN
PRO print_results, chi_sq_test, lrt, expected, chi_sq_contrib, $
    chi_sq_stats, table_stats
PRINT, 'Pearson Chi_Squared Statistics:'
PM, chi_sq_test(0), Title = 'Degrees of Freedom'
PM, chi_sq_test(1), Title = 'Chi-Squared'
PM, chi_sq_test(2), Title = 'P-Value'
PRINT
PRINT, 'Likelihood Ratio G-Squared ' + 'Statistics:'
PM, lrt(0), Title = 'Degrees of Freedom'
PM, lrt(1), Title = 'G-Squared'
PM, lrt(2), Title = 'P-Value'
PRINT
PM, expected, Title = 'Expected Values:'
PRINT
PM, chi_sq_contrib, Title = 'Contributions to Chi-squared:'
PRINT
PM, chi_sq_stats, Title = 'Chi-square Statistics:'
PRINT
PM, table_stats, Title = 'Table Statistics:'
END

table = [[821,116,72,43], [112,494,151,34], [85,145,583,106], $
    [35,27,87,331]]
p_value = IMSL_CONTINGENCY(table, $
    Chi_Sq_Test      = chi_sq_test, $
    Lrt              = lrt, $
    Expected         = expected, $
    Chi_Sq_Contrib   = chi_sq_contrib, $
    Chi_Sq_Stats     = chi_sq_stats, $
    Table_Stats      = table_stats)
print_results, chi_sq_test, lrt, expected, chi_sq_contrib, $
    chi_sq_stats, table_stats

Pearson Chi_Squared Statistics:
Degrees of Freedom
    9.00000
Chi-Squared
    3304.37
P-Value
    0.00000
Likelihood Ratio G-Squared Statistics:
Degrees of Freedom
    9.00000
G-Squared
    2781.02
P-Value
    0.00000
Expected Values:

```

```

341.689 256.916 298.491 155.904 1053.00
253.752 190.796 221.671 115.780 782.000
289.771 217.879 253.136 132.215 893.000
166.788 125.408 145.702 76.1012 514.000
1052.00 791.000 919.000 480.000 3242.00
Contributions to Chi-squared:
672.363 81.7416 152.696 93.7612 1000.56
74.7802 481.835 26.5189 68.0768 651.211
163.661 20.5287 429.849 15.4625 629.501
91.8743 66.6263 10.8183 853.777 1023.10
1002.68 650.732 619.882 1031.08 3304.37
Chi-square Statistics:
9.00278
4.24016
1.00957
0.710467
0.582877
Table Statistics:
0.775704 0.0122983 0.0148632 52.1897 0.00000
0.642887 0.0122028 0.0123183 52.1897 0.00000
0.629265 0.0120573 NaN 52.1897 0.00000
0.641831 0.0122390 0.0122980 52.1897 0.00000
0.643945 0.0122152 0.0123385 52.1897 0.00000
0.692588 0.0127669 0.0172000 40.2669 0.00000
0.693882 0.0126566 0.0126942 54.6614 0.00000
0.341952 0.0122570 NaN NaN NaN
0.342993 0.0122165 NaN NaN NaN
0.317123 0.0110281 NaN NaN NaN
0.317811 0.0110453 NaN NaN NaN
0.316437 0.0110294 NaN NaN NaN
0.537337 0.0123718 NaN NaN NaN
0.537443 0.0125727 NaN NaN NaN
0.537232 0.0125851 NaN NaN NaN
0.550648 0.0135695 NaN NaN NaN
0.563587 0.0126838 NaN NaN NaN
NaN NaN NaN NaN NaN
1561.49 3.00000 NaN NaN 0.00000
1563.03 3.00000 NaN NaN 0.00000
0.574419 0.0110873 0.0105673 54.3583 0.00000
4.76249 6.00000 NaN NaN 0.574617
0.948667 1.00000 NaN 0.345904 0.330059

```

Errors

Warning Errors

STAT_DF_GT_30—The degrees of freedom for *Chi_Sq_Test* are greater than 30. The exact mean, standard deviation, and the normal distribution function should be used.

STAT_EXP_VALUES_TOO_SMALL—Some expected values are less than #. Some asymptotic *p*-values may not be good.

STAT_PERCENT_EXP_VALUES_LT_5—Twenty percent of the expected values are calculated less than 5.

Version History

6.4	Introduced

IMSL_EXACT_ENUM

The IMSL_EXACT_ENUM function computes exact probabilities in a two-way contingency table using the total enumeration method.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_EXACT_ENUM(table [, /DOUBLE] [, ERROR_CHK=variable]  
[, P_VALUE=variable] [, PROB_TABLE=variable] )
```

Return Value

The p -value for independence of rows and columns. The p -value represents the probability of a more extreme table where “extreme” is taken in the Neyman-Pearson sense. The p -value is “two-sided”.

Arguments

table

Two-dimensional array containing the observed counts in the contingency table.

Keywords

DOUBLE

If present and nonzero, double precision is used.

ERROR_CHK

Named variable into which the sum of the probabilities of all tables with the same marginal totals is stored. Keyword *Error_Chk* should have a value of 1.0. Deviation from 1.0 indicates numerical error.

P_VALUE

Named variable into which the p -value for independence of rows and columns is stored. The p -value represents the probability of a more extreme table where “extreme” is taken in the Neyman-Pearson sense. The p -value is “two-sided”.

The p -value is also returned in functional form (see *Returned Value*).

A table is more extreme if its probability (for fixed marginals) is less than or equal to *Prob_Table*.

PROB_TABLE

Named variable into which the probability of the observed table occurring, given that the null hypothesis of independent rows and columns is true, is stored.

Discussion

The IMSL_EXACT_ENUM function computes exact probabilities for an r by c contingency table for fixed row and column marginals (a marginal is the number of counts in a row or column), where $r = \text{N\_ELEMENTS}(\text{table}(*,0))$ and $c = \text{N\_ELEMENTS}(\text{table}(0,*))$. Let f_{ij} denote the count in row i and column j of a table, and let $f_{i\cdot}$ and $f_{\cdot j}$ denote the row and column marginals. Under the hypothesis of independence, the (conditional) probability of the fixed marginals of the observed table is given by:

$$P_f = \frac{\prod_{i=1}^r f_{i\cdot}! \prod_{j=1}^c f_{\cdot j}!}{f_{\cdot\cdot}! \prod_{i=1}^r \prod_{j=1}^c f_{ij}!}$$

where $f_{\cdot\cdot}$ is the total number of counts in the table. P_f corresponds to output keyword *Prob_Table*.

A more extreme table X is defined in the probabilistic sense as more extreme than the observed table if the conditional probability computed for table X (for the same marginal sums) is less than the conditional probability computed for the observed table. Note that this definition can be considered “two-sided” in the cell counts.

Because IMSL_EXACT_ENUM uses total enumeration in computing the probability of a more extreme table, the amount of computer time required increases very rapidly with the size of the table. Tables with a large total count $f_{\cdot\cdot}$ or a large value of r by c should not be analyzed using IMSL_EXACT_ENUM. In such cases, try using IMSL_EXACT_NETWORK.

Example

In this example, the exact conditional probability for the 2 by 2 contingency table is computed as follows:

$$\begin{bmatrix} 8 & 12 \\ 8 & 2 \end{bmatrix}$$

```
table = [[8, 8], [12, 2]]  
p = IMSL_EXACT_ENUM(table, P_Value=pv, Prob_Table=pt,  
Error_Chk=ec)  
PRINT, 'p-value =', p  
p-value = 0.0576712
```

Version History

6.4	Introduced

IMSL_EXACT_NETWORK

The IMSL_EXACT_NETWORK function computes Fisher exact probabilities and a hybrid approximation of the Fisher exact method for a two-way contingency table using the network algorithm.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_EXACT_NETWORK(table [, APPROX_PARAMS=array]  
[, /DOUBLE] [, /NO_APPROX] [, P_VALUE=variable]  
[, PROB_TABLE=variable] [, WK_PARAMS=array])
```

Return Value

The p -value for independence of rows and columns. The p -value represents the probability of a more extreme table where “extreme” is taken in the Neyman-Pearson sense. The p -value is “two-sided”.

Arguments

table

Two-dimensional array containing the observed counts in the contingency table.

Keywords

APPROX_PARAMS

One-dimensional array of size 3. *Approx_Params*(0) is the expected value used in the hybrid approximation to Fisher’s exact test algorithm for deciding when to use asymptotic probabilities when computing path lengths. *Approx_Params*(1) is the percentage of remaining cells that must have estimated expected values greater than *Approx_Params*(0) before asymptotic probabilities can be used in computing path lengths. *Approx_Params*(2) is the minimum cell estimated value allowed for asymptotic chi-squared probabilities to be used.

Asymptotic probabilities are used in computing path lengths whenever *Approx_Params*(1) or more of the cells in the table have estimated expected values of *Approx_Params*(0) or more, with no cell having expected value less than *Approx_Params*(2). See the *Discussion* section for details.

Defaults: *Approx_Params*(0) = 5.0

Approx_Params(1) = 80.0

Approx_Params(2) = 1.0

Note

These defaults correspond to the “Cochran” condition.

DOUBLE

If present and nonzero, double precision is used.

NO_APPROX

If present and nonzero, the Fisher exact test is used and *Approx_Param* is ignored.

P_VALUE

Named variable into which the *p*-value for independence of rows and columns is stored. The *p*-value represents the probability of a more extreme table where “extreme” is in the Neyman-Pearson sense. The *P_Value* is “two-sided”. The *p*-value is also returned in functional form (see *Returned Value*).

A table is more extreme if its probability (for fixed marginals) is less than or equal to *Prob_Table*.

PROB_TABLE

Named variable into which the probability of the observed table occurring given that the null hypothesis of independent rows and columns is true is stored.

WK_PARAMS

One-dimensional array of size 3. The network algorithm requires a large amount of workspace. Some of the workspace requirements are well-defined, while most of the workspace requirements can only be estimated. The estimate is based primarily on table size.

The *IMSL_EXACT_ENUM* function allocates a default amount of workspace suitable for small problems. If the algorithm determines that this initial allocation of

workspace is inadequate, the memory is freed, a larger amount of memory allocated (twice as much as the previous allocation), and the network algorithm is re-started. The algorithm allows for up to $Wk\_Params(2)$ attempts to complete the algorithm.

Because each attempt requires computer time, it is suggested that $Wk\_Params(0)$ and $Wk\_Params(1)$ be set to some large numbers (like 1,000 and 30,000) if the problem to be solved is large. It is suggested that $Wk\_Params(1)$ be 30 times larger than $Wk\_Params(0)$. Although IMSL_EXACT_ENUM will eventually work its way up to a large enough memory allocation, it is quicker to allocate enough memory initially.

The known (well-defined) workspace requirements are as follows: Define $f_{..} = \sum \sum f_{ij}$ equal to the sum of all cell frequencies in the observed table, $nt = f_{..} + 1$, $mx = \max(n\_rows, n\_columns)$, $mn = \min(n\_rows, n\_columns)$, $t_1 = \max(800 + 7mx, (5 + 2mx)(n\_rows + n\_columns + 1))$, and $t_2 = \max(400 + mx, 1, n\_rows + n\_columns + 1)$ where $n\_rows = N\_ELEMENTS(table(*,0))$ and $n\_columns = N\_ELEMENTS(table(0,*))$.

The following amount of integer workspace is allocated: $3mx + 2mn + t_1$.

The following amount of real workspace is allocated: $nt + t_2$.

The remainder of workspace that is required must be estimated and allocated based on $Wk\_Params(0)$ and $Wk\_Params(1)$. The amount of integer workspace allocated is $6n(Wk\_Params(0) + Wk\_Params(1))$. The amount of real workspace allocated is $n(6 * Wk\_Params(0) + 2 * Wk\_Params(1))$. Variable n is the index for the attempt, $1 < n \leq Wk\_Params(2)$.

Defaults: $Wk\_Params(0) = 100$

$Wk\_Params(1) = 3000$

$Wk\_Params(2) = 10$

Discussion

The IMSL_EXACT_NETWORK function computes Fisher exact probabilities or a hybrid algorithm approximation to Fisher exact probabilities for an r by c contingency table with fixed row and column marginals (a marginal is the number of counts in a row or column), where $r = n\_rows$ and $c = n\_columns$. Let f_{ij} denote the count in row i and column j of a table, and let $f_{i\cdot}$ and $f_{\cdot j}$ denote the row and column

marginals. Under the hypothesis of independence, the (conditional) probability of the fixed marginals of the observed table is given by:

$$P_f = \frac{\prod_{i=1}^r f_{i\bullet}! \prod_{j=1}^c f_{\bullet j}!}{f_{\bullet\bullet}! \prod_{i=1}^r \prod_{j=1}^c f_{ij}!}$$

where $f_{\bullet\bullet}$ is the total number of counts in the table. P_f corresponds to output keyword *Prob_Table*.

A “more extreme” table X is defined in the probabilistic sense as more extreme than the observed table if the conditional probability computed for table X (for the same marginal sums) is less than the conditional probability computed for the observed table. Note that this definition can be considered “two-sided” in the cell counts.

Example

This example demonstrates various methods of computing chi-squared p -value with respect to accuracy. As seen in the output of this example, the Fisher exact probability and the usual asymptotic chi-squared probability (generated using *IMSL_CONTINGENCY*) can be different.

```
.RUN
PRO print_results, p, p2, p3, p4
  PRINT, 'Asymptotic Chi-Squared p-value'
  PRINT, 'p-value =', p
  PRINT, 'Network Algorithm with Approximation'
  PRINT, 'p-value =', p2
  PRINT, 'Network Algorithm without Approximation'
  PRINT, 'p-value =', p3
  PRINT, 'Total Enumeration Method'
  PRINT, 'p-value =', p4
END

table = TRANSPOSE([[20, 20, 0, 0, 0], [10, 10, 2, 2, 1], $
  [20, 20, 0, 0, 0]])
p = IMSL_CONTINGENCY(table)
p2 = IMSL_EXACT_NETWORK(table)
p3 = IMSL_EXACT_NETWORK(table, /NO_APPROX)
p4 = IMSL_EXACT_ENUM(table)
print_results, p, p2, p3, p4

Asymptotic Chi-Squared p-value
p-value =      0.0322604
Network Algorithm with Approximation
```

```
p-value =      0.0601165
Network Algorithm without Approximation
p-value =      0.0598085
Total Enumeration Method
p-value =      0.0597294
```

Errors

Warning Errors

STAT_HASH_TABLE_ERROR_2—The value “ldkey” = # is too small. “ldkey” is calculated as *Wk_Params*(0)*pow(10, *N_Attempts*–1) ending this execution attempt.

STAT_HASH_TABLE_ERROR_3—The value “ldstp” = # is too small. “ldstp” is calculated as *Wk_Params*(1)*pow(10, *N_Attempts*–1) ending this execution attempt.

Fatal Errors

STAT_HASH_TABLE_ERROR_1—The hash table key cannot be computed because the largest key is larger than the largest representable integer. The algorithm cannot proceed.

Version History

6.4	Introduced

IMSL_CAT_GLM

The IMSL_CAT_GLM function analyzes categorical data using logistic, Probit, Poisson, and other generalized linear models.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_CAT_GLM(n_class, n_continuous, model, x
    [, CASE_ANALYSIS=variable] [, CLASS_VALS=variable]
    [, COVARIANCES=variable] [, COEF_STAT=variable]
    [, CRITERION=variable] [, /DOUBLE] [, EPS=value] [, IFIX=value]
    [, IFREQ=value] [, INDICES_EFFECTS=array] [, INIT_EST=array]
    [, IPAR=value] [, ITMAX=value] [, LAST_STEP=variable]
    [, MAX_CLASS=value] [, MEANS=variable] [, N_CLASS_VALS=variable]
    [, /NO_INTERCEPT] [, OBS_STATUS=variable] [, VAR_EFFECTS=array])
```

Return Value

An integer value indicating the number of estimated coefficients in the model.

Arguments

model

Model used to analyze the data. The six models are listed in [Table 17-2](#).

model	Relationship*	PDF of Response Variable
0	Exponential	Poisson
1	Logistic	Negative Binomial
2	Logistic	Logarithmic
3	Logistic	Binomial

Table 17-2: Six Models

model	Relationship*	PDF of Response Variable
4	Probit	Binomial
5	Log-log	Binomial

* Relationship between the parameter, θ or λ , and a linear model of the explanatory variables, $X\beta$.

Table 17-2: Six Models

Note

The lower bound of the response variable is 1 for *model* = 3 and is 0 for all other models. See the *Discussion* section for more information about these models.

n_class

Number of classification variables.

n_continuous

Number of continuous variables.

x

Two-dimensional array of size *n_observations* by (*n_class* + *n_continuous*) + *m* containing data for the independent variables, dependent variable, and optional parameters, where *n_observations* is the number of observations.

The columns must be ordered such that the first *n_class* columns contain data for the class variables, the next *n_continuous* columns contain data for the continuous variables, and the next column contains the response variable. The final (and optional) *m* - 1 columns contain optional parameters, see keywords *Ifreq*, *Ifix*, and *Ipar*.

Keywords

CASE_ANALYSIS

Named variable into which a two-dimensional array of size *n_observations* by 5 containing the case analysis is stored.

- 0—Predicted mean for the observation if *model* = 0. Otherwise, contains the probability of success on a single trial.

- 1—The residual.
- 2—The estimated standard error of the residual.
- 3—The estimated influence of the observation.
- 4—The standardized residual.

Case statistics are computed for all observations except where missing values prevent their computation.

CLASS_VALS

Named variable into which a one-dimensional array of length:

$$\sum_{i=0}^{n\_class-1} N\_Class\_Vals(i)$$

containing the distinct values of the classification variables in ascending order is stored. The first $N\_Class\_Vals(0)$ elements of $Class\_Vals$ contain the values for the first classification variables, the next $N\_Class\_Vals(1)$ elements contain the values for the second classification variable, etc.

COVARIANCES

Named variable into which a two-dimensional array of size $n\_coefficients$ by $n\_coefficients$ containing the estimated asymptotic covariance matrix of the coefficients is stored. For $Itmax = 0$, this is the Hessian computed at the initial parameter estimates.

COEF_STAT

Named variable into which a two-dimensional array of size $n\_coefficients$ by 4 containing the parameter estimates and associated statistics is stored.

- 0—Coefficient Estimate.
- 1—Estimated standard deviation of the estimated coefficient.
- 2—Asymptotic normal score for testing that the coefficient is zero.
- 3—The p -value associated with the normal score in column 2.

CRITERION

Named variable into which the optimized criterion is stored. The criterion to be maximized is a constant plus the log-likelihood.

DOUBLE

If present and nonzero, double precision is used.

EPS

Convergence criterion. Convergence is assumed when maximum relative change in any coefficient estimate is less than *Eps* from one iteration to the next or when the relative change in the log-likelihood, criterion, from one iteration to the next is less than *Eps*/100.0. Default: *Eps* = 0.001

IFIX

Column number *Ifix* in *x* containing a fixed parameter for each observation that is added to the linear response prior to computing the model parameter. The ‘fixed’ parameter allows one to test hypothesis about the parameters via the log-likelihoods.

IFREQ

Column number *Ifreq* in *x* containing the frequency of response for each observation.

INDICES_EFFECTS

One-dimensional index array of length $Var\_Effects(0) + Var\_Effects(1) + \dots + Var\_Effects(n\_effects - 1)$. The first $Var\_Effects(0)$ elements give the column numbers of *x* for each variable in the first effect. The next $Var\_Effects(1)$ elements give the column numbers for each variable in the second effect. The last $Var\_Effects(n\_effects - 1)$ elements give the column numbers for each variable in the last effect. Keywords *Indices_Effects* and *Var_Effects* must be used together.

INIT_EST

One-dimensional array of length *n_coef_input* containing initial estimates of parameters (*n_coef_input* can be completed by *IMSL_REGRESSORS*). By default, unweighted linear regression is used to obtain initial estimates.

IPAR

Column number *Ipar* in *x* containing the value of the known distribution parameter for each observation, where $x(i, Ipar)$ is the known distribution parameter associated

with the i -th observation. The meaning of the distributional parameter depends upon *model* as shown in Table 17-3:

model	Parameter	Meaning of parameter (i)(<i>lpar</i>)
0	E	ln (E) is a fixed intercept to be included in the linear predictor (i.e., the <i>offset</i>).
1	S	Number of successes required for the negative binomial distribution.
2	—	Not used for this model.
3-5	N	Number of trials required for the binomial distribution.

Table 17-3: Distributional Parameters

Default: When *model* $\neq$ 2, each observation is assumed to have a parameter value of 1. When *model* = 2, this parameter is not referenced.

ITMAX

Maximum number of iterations. Use *Itmax* = 0 to compute Hessian, stored in *Covariances*, and the Newton step, stored in *Last_Step*, at the initial estimates (The initial estimates must be input. Use keyword *Init_Est*). Default: *Itmax* = 30

LAST_STEP

Named variable into which an one-dimensional array of length *n_coefficients* containing the last parameter updates (excluding step halvings) is stored. For *Itmax* = 0, *Last_Step* contains the inverse of the Hessian times the gradient vector, all computed at the initial parameter estimates.

MAX_CLASS

An upper bound on the sum of the number of distinct values taken on by each classification variable. Default: *Max_Class* = *n_observations* by *n_class*

MEANS

Named variable into which an one-dimensional array containing the means of the design variables is stored. The array is of length *n_coefficients* if keyword *No_Intercept* is used, and *n_coefficients* – 1 otherwise.

N_CLASS_VALS

Named variable into which an one-dimensional array of length n_class containing the number of values taken by each classification variable is stored; the i -th classification variable has $N\_Class\_Vals(i)$ distinct values.

NO_INTERCEPT

If present and nonzero, there is no intercept in the model. By default, the intercept is automatically included in the model.

OBS_STATUS

Named variable into which an one-dimensional array of length $n\_observations$ indicating which observations are included in the extended likelihood is stored.

- 0—Observation i is in the likelihood
- 1—Observation i cannot be in the likelihood because it contains at least one missing value in x .
- 2—Observation i is not in the likelihood. Its estimated parameter is infinite.

Remarks

1. Dummy variables are generated for the classification variables as follows: An ascending list of all distinct values of each classification variable is obtained and stored in *Class_Vals*. Dummy variables are then generated for each but the last of these distinct values. Each dummy variable is zero unless the classification variable equals the list value corresponding to the dummy variable, in which case the dummy variable is one. See input keyword *Dummy_Method* = 1 in routine IMSL_REGRESSORS (Chapter 2, *Regression*).
2. The “product” of a classification variable with a covariate yields dummy variables equal to the product of the covariate with each of the dummy variables associated with the classification variable.
3. The “product” of two classification variables yields dummy variables in the usual manner. Each dummy variable associated with the first classification variable multiplies each dummy variable associated with the second classification variable. The resulting dummy variables are such that the index of the second classification variable varies fastest.

VAR_EFFECTS

One-dimensional array of length $n\_effects$ containing the number of variables associated with each effect in the model, where $n\_effects$ is the number of effects (source of variation) in the model. Keywords *Var_Effects* and *Indices_Effects* must be used together.

Discussion

The *IMSL_CAT_GLM* function uses iteratively re-weighted least squares to compute (extended) maximum likelihood estimates in some generalized linear models involving categorized data. One of several models, including the probit, logistic, Poisson, logarithmic, and negative binomial models, may be fit.

Note that each row vector in the data matrix can represent a single observation; or, through the use of keyword *Ifreq*, each row can represent several observations. Also note that classification variables and their products are easily incorporated into the models via the usual regression-type specifications. The models available in *IMSL_CAT_GLM* are listed in [Table 17-4](#).

Model	PDF of the Response Variable	Parameterization
0	$f(y) = (\lambda_y \exp(-\lambda)) / y!$	$\lambda = N \times \exp(\omega + \eta)$
1	$f(y) = \binom{S+y-1}{y-1} \theta^S (1-\theta)^y$	$\theta = \frac{\exp(\omega + \eta)}{1 + \exp(\omega + \eta)}$
2	$f(y) = (1-\theta)^y / (y \ln \theta)$	$\theta = \frac{\exp(\omega + \eta)}{1 + \exp(\omega + \eta)}$
3	$f(y) = \binom{N}{y} \theta^y (1-\theta)^{N-y}$	$\theta = \frac{\exp(\omega + \eta)}{1 + \exp(\omega + \eta)}$

Table 17-4: *IMSL_CAT_GLM* Models

Model	PDF of the Response Variable	Parameterization
4	$f(y) = \binom{N}{y} \theta^y (1 - \theta)^{N-y}$	$\theta = \Phi(\omega + \eta)$
5	$f(y) = \binom{N}{y} \theta^y (1 - \theta)^{N-y}$	$\theta = 1 - \exp(-\exp(\omega + \eta))$

Table 17-4: IMSL_CAT_GLM Models (Continued)

Here, Φ denotes the cumulative normal distribution, N and S are known distribution parameters specified for each observation via the keyword *Ipar*, and ω is an optional fixed parameter of the linear response, γ_i , specified for each observation. (If keyword *Ifix* is not used, then ω is taken to be 0.) Since the log-log model (*model* = 5) probabilities are not symmetric with respect to 0.5, quantitatively, as well as qualitatively, different models result when the definitions of “success” and “failure” are interchanged in this distribution. In this model and all other models involving θ , θ is taken to be the probability of a “success”.

Computational Details

The computations proceed as follows:

1. The input parameters are checked for consistency and validity.
2. Estimates of the means of the “independent” or design variables are computed. The frequency or the observation in all but binomial distribution models is taken from vector frequencies. In binomial distribution models, the frequency is taken as the product of n = parameter (i) and frequencies (i). Means are computed as:

$$\bar{x} = \frac{\sum f_i x_i}{\sum f_i}$$

3. By default, unless keyword *Init_Est* is used, initial estimates of the coefficients are obtained (based upon the observation intervals) as multiple regression estimates relating transformed observation probabilities to the observation design vector. For example, in the binomial distribution models, θ may be estimated as:

$$\hat{\theta} = y(i)/\text{parameter}(i)$$

and, when *model* = 3, the linear relationship is given by:

$$\ln(\hat{\theta}/(1 - \hat{\theta})) \approx X\beta$$

while if *model* = 4, $\Phi^{-1}(\theta) = X\beta$. When computing initial estimates, standard modifications are made to prevent illegal operations such as division by zero. Regression estimates are obtained at this point, as well as later, by use of *IMSL_MULTIREGRESS* (Chapter 2, *Regression*).

4. Newton-Raphson iteration for the maximum likelihood estimates is implemented via iteratively re-weighted least squares. Let:

$$\Psi(x_i^T \beta)$$

denote the log of the probability of the *i*-th observation for coefficients β . In the least-squares model, the weight of the *i*-th observation is taken as the absolute value of the second derivative of:

$$\Psi(x_i^T \beta)$$

with respect to:

$$\gamma_i = x_i^T \beta$$

(times the frequency of the observation), and the dependent variable is taken as the first derivative Ψ with respect to γ_i , divided by the square root of the weight times the frequency. The Newton step is given by:

$$\Delta\beta = (\Sigma | \Psi''(\gamma_i) | x_i x_i^T)^{-1} \Sigma \Psi'(\gamma_i) x_i$$

where all derivatives are evaluated at the current estimate of γ and $\beta_{n+1} = \beta - \Delta\beta$. This step is computed as the estimated regression coefficients in the least-squares model. Step halving is used when necessary to ensure a decrease in the criterion.

5. Convergence is assumed when the maximum relative change in any coefficient update from one iteration to the next is less than *Eps* or when the relative change in the log-likelihood from one iteration to the next is less than *Eps*/100.

Convergence is also assumed after *Itmax* iterations or when step halving leads to a step size of less than 0.0001 with no increase in the log-likelihood.

6. Residuals are computed according to methods discussed by Pregibon (1981). Let $l_i(\gamma_i)$ denote the log-likelihood of the i -th observation evaluated at γ_i . Then, the standardized residual is computed as:

$$r_i = \frac{l'_i(\hat{\gamma}_i)}{\sqrt{l''_i(\hat{\gamma}_i)}}$$

where:

$\hat{\gamma}_i$

is the value of γ_i when evaluated at the optimal:

$\hat{\beta}$

The denominator of this expression is used as the “standard error of the residual” while the numerator is “raw” residual. Following Cook and Weisberg (1982), the influence of the i -th observation is assumed to be:

$$l'_i(\hat{\gamma}_i)^T l''_i(\hat{\gamma}_i)^{-1} l'_i(\hat{\gamma}_i)$$

This is a one-step approximation to the change in estimates when the i -th observation is deleted. Here, the partial derivatives are with respect to β .

Programming Notes

1. Indicator (dummy) variables are created for the classification variables using `IMSL_REGRESSORS` (Chapter 2, *Regression*) using keyword `Dummy_Method = 1`.
2. To enhance precision, “centering” of covariates is performed if the model has an intercept and `n_observations - Nmissing > 1`. In doing so, the sample means of the design variables are subtracted from each observation prior to its inclusion in the model. On convergence, the intercept, its variance, and its covariance with the remaining estimates are transformed to the uncentered estimate values.
3. Two methods for specifying a binomial distribution model are possible. In the first method, `Ifreq` contains the frequency of the observation while `x(i, irt-1)` is 0 or 1 depending upon whether the observation is a success or failure. In this case, `x(i, n_class + n_continuous)` is always 1. The model is treated as repeated Bernoulli trials, and interval observations are not possible. A second

method for specifying binomial models is to use to represent the number of successes in parameter (i) trials. In this case, frequencies will usually be 1.

Example

This example is from Prentice (1976) and involves mortality of beetles after five hours exposure to eight different concentrations of carbon disulphide. The table below lists the number of beetles exposed (N) to each concentration level of carbon disulphide (x , given as log dosage) and the number of deaths which result (y). The data is shown in [Table 17-5](#):

Log Dosage	Number of Beetles Exposed	Number of Deaths
1.690	59	6
1.724	60	13
1.755	62	18
1.784	56	28
1.811	63	52
1.836	59	53
1.861	62	61
1.883	60	60

Table 17-5: Beetle Mortality

The number of deaths at each concentration level are fitted as a binomial response using logit ($model = 3$), probit ($model = 4$), and log-log ($model = 5$) models. Note that the log-log model yields a smaller absolute log likelihood (14.81) than the logit model (18.78) or the probit model (18.23). This is to be expected since the response curve of the log-log model has an asymmetric appearance, but both the logit and probit models are symmetric about $\theta = 0.5$.

```
.RUN
PRO print_results, cs, means, ca, crit, ls, cov
  PRINT, ' Coefficient Statistics'
  PRINT, '                Standard   Asymptotic   ', $
    'Asymptotic'
  PRINT, ' Coefficient           Error   Z-statistic   ', $
    'P-value'
  PM, cs, FORMAT = '(4F13.2)'
```

```

PRINT
PRINT, 'Covariate Means = ', means, FORMAT = '(A18, F6.3)'
PRINT
PRINT, '                                Case Analysis'
PRINT, '                                Residual                ', $
      'Standardized'
PRINT, '    Predicted    Residual    Std. Error    Leverage', $
      '    Residual'
PM, ca, FORMAT = '(5F12.3)'
PRINT
PRINT, 'Log-Likelihood = ', crit, FORMAT = '(A18, F9.5)'
PRINT
PRINT, '                Last Step'
PRINT, ls
PRINT
PRINT, 'Asymptotic Coefficient Covariance'
PM, cov, FORMAT = '(2F12.4)'
END

model = 3
nobs = 8
x = ([[1.690, 1.724, 1.755, 1.784, 1.811, 1.836, 1.861, 1.883], $
      [6, 13, 18, 28, 52, 53, 61, 60], $
      [59, 60, 62, 56, 63, 59, 62, 60]])
ncoef = IMSL_CAT_GLM(0, 1, model, x, Ipar = 2, Eps = 1.0e-3, $
      Coef_Stat = cs, Covariances = cov, $
      Criterion = crit, Means = means, $
      Case_Analysis = ca, Last_Step = ls, Obs_Status = os)
print_results, cs, means, ca, crit, ls, cov

```

Coefficient Statistics			
	Standard	Asymptotic	Asymptotic
Coefficient	Error	Z-statistic	P-value
-60.76	5.21	-11.66	0.00
34.30	2.92	11.76	0.00

Covariate Means = 1.793

Case Analysis				
		Residual		Standardized
Predicted	Residual	Std. Error	Leverage	Residual
0.058	2.593	1.792	0.267	1.448
0.164	3.139	2.871	0.347	1.093
0.363	-4.498	3.786	0.311	-1.188
0.606	-5.952	3.656	0.232	-1.628
0.795	1.890	3.202	0.269	0.590
0.902	-0.195	2.288	0.238	-0.085
0.956	1.743	1.619	0.198	1.077


```

0.979      1.278      1.119      0.138      1.143

Log-Likelihood = -18.77818

      Last Step
-3.67824e-08  1.04413e-05

Asymptotic Coefficient Covariance
 27.1368      -15.1243
-15.1243      8.5052

```

Errors

Warning Errors

STAT_TOO_MANY_HALVINGS—Too many step halvings. Convergence is assumed.

STAT_TOO_MANY_ITERATIONS—Too many iterations. Convergence is assumed.

Fatal Errors

STAT_TOO_FEW_COEF—*Init_Est* is used and “n_coef_input” = #. The model specified requires # coefficients.

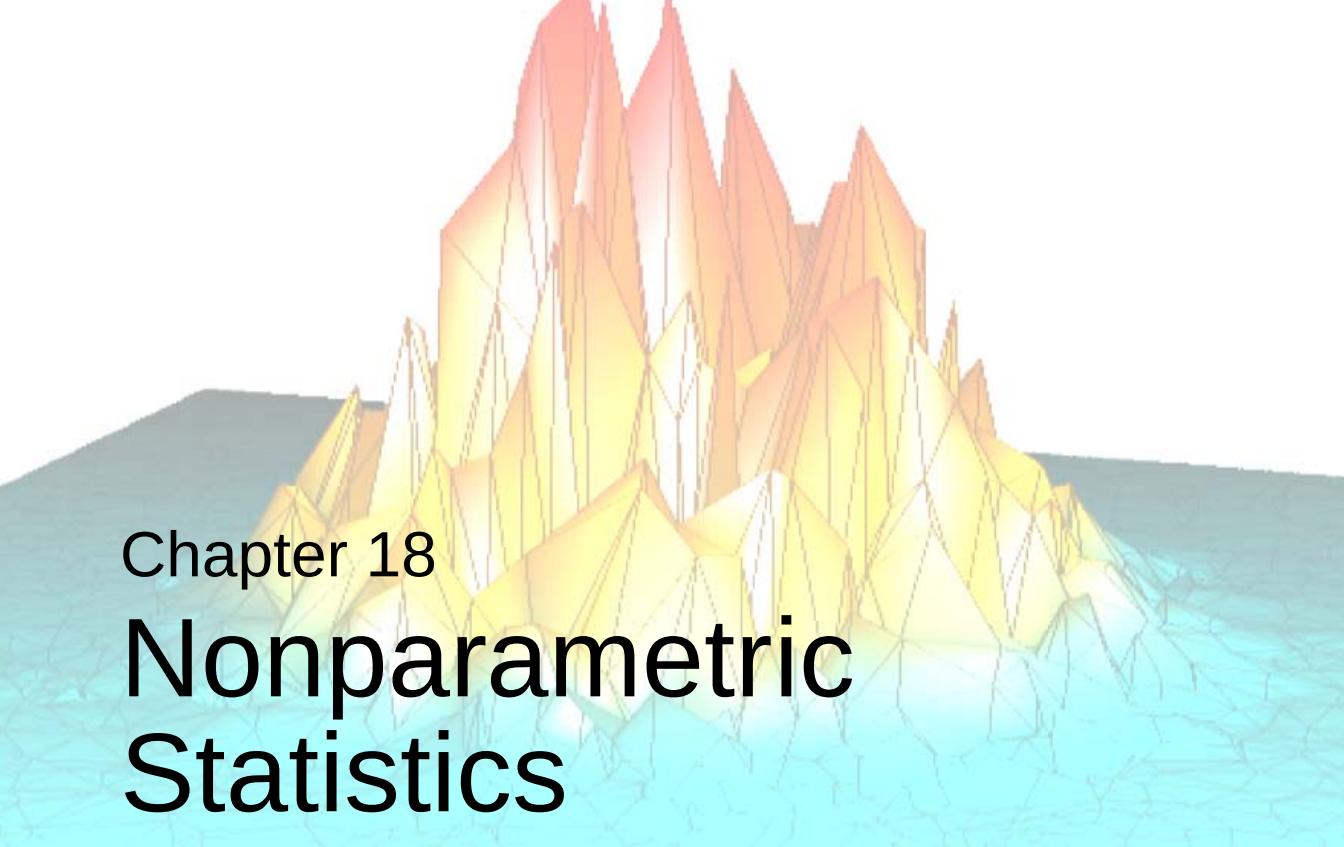
STAT_MAX_CLASS_TOO_SMALL—The number of distinct values of the classification variables exceeds “Max_Class” = #.

STAT_INVALID_DATA_8—“N_Class_Values(#)” = #. The number of distinct values for each classification variable must be greater than one.

STAT_NMAX_EXCEEDED—The number of observations to be deleted has exceeded “lp_max” = #. Rerun with a different model or increase the workspace.

Version History

6.4	Introduced



Chapter 18

Nonparametric Statistics

This section contains the following topics:

Overview	832	Nonparametric Statistics Routines	833

Overview

This chapter contains nonparametric statistics routines. Much about nonparametric statistics is also included in other chapters. Topics that can be found in other chapters are:

- Nonparametric measures of location and scale ([Chapter 13, “Basic Statistics”](#))
- Nonparametric measures in a contingency table ([Chapter 17, “Categorical and Discrete Data Analysis”](#))
- Measures of correlation in a contingency table ([Chapter 15, “Correlation and Covariance”](#))
- Tests of goodness of fit and randomness ([Chapter 19, “Goodness of Fit”](#))

Missing Values

Most routines in this chapter automatically handle missing values (*NaN* — not a number).

Tied Observations

Many of the routines described in this chapter contain a keyword FUZZ in the input. Observations that are within FUZZ of each other in absolute value are said to be tied. Moreover, in some routines, an observation within FUZZ of some value is said to be equal to that value. In the “[IMSL_WILCOXON](#)” on page 837, for example, such observations are eliminated from the analysis. If $FUZZ = 0.0$, observations must be identically equal before they are considered to be tied. Other positive values of FUZZ allow for numerical imprecision or roundoff error.

Nonparametric Statistics Routines

One Sample Tests—Nonparametric Statistics

[IMSL_SIGNTEST](#)—Sign test.

[IMSL_WILCOXON](#)—Wilcoxon rank sum test.

[IMSL_NCTRENDS](#)—Noether's test for cyclical trend.

[IMSL_CSTRENDS](#)—Cox and Stuarts' sign test for trends in location and dispersion.

[IMSL_TIE_STATS](#)—Tie statistics.

Two or More Samples Tests—Nonparametric Statistics

[IMSL_KW_TEST](#)—Kruskal-Wallis test.

[IMSL_FRIEDMANS_TEST](#)—Friedman's test.

[IMSL_COCHRANQ](#)—Cochran's Q test.

[IMSL_KTRENDS](#)—K-sample trends test.

IMSL_SIGNTEST

The IMSL_SIGNTEST function performs a sign test.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_SIGNTEST(x [, /DOUBLE] [, N_POS_DEV=value]
                        [, N_ZERO_DEV=value] [, PERCENTAGE=value] [, PERCENTILE=value] )
```

Return Value

Binomial probability of N_Pos_Dev or more positive differences in $N\_ELEMENTS(x) - N\_Zero\_Dev$ trials. Call this value *probability*. If no option is chosen, the null hypothesis is that the median equals 0.0.

Arguments

x

One-dimensional array containing the input data.

Keywords

DOUBLE

If present and nonzero, double precision is used.

N_POS_DEV

Number of positive differences $x(j - 1) - Percentile$, for $j = 1, 2, \dots, N\_ELEMENTS(x)$.

N_ZERO_DEV

Number of zero differences (ties) $x(j - 1) - Percentile$, for $j = 1, 2, \dots, N\_ELEMENTS(x)$.

PERCENTAGE

Scalar value in the range (0,1). Keyword *Percentage* is the 100 x *Percentage* percentile of the population. Default: *Percentage* = 0.5

PERCENTILE

Hypothesized percentile of the population from which x was drawn. Default: *Percentile* = 0.0

Discussion

The IMSL_SIGNTEST function tests hypotheses about the proportion p of a population that lies below a value q , where p corresponds to keyword *Percentage* and q corresponds to keyword *Percentile*. In continuous distributions, this can be a test that q is the 100 p -th percentile of the population from which x was obtained. To carry out testing, IMSL_SIGNTEST tallies the number of values above q in N_Pos_Dev . The binomial probability of N_Pos_Dev or more values above q is then computed using the proportion p and the sample size $N\_ELEMENTS$ (x) (adjusted for the missing observations and ties).

Hypothesis testing is performed as follows for the usual null and alternative hypotheses:

- $H_0: Pr(X \leq q) \geq p$ (the p -th quantile is at least q)
 $H_1: Pr(X < q) < p$
 Reject H_0 if *probability* is less than or equal to the significance level.
- $H_0: Pr(X \leq q) \leq p$ (the p -th quantile is at least q)
 $H_1: Pr(X < q) > p$
 Reject H_0 if *probability* is greater than or equal to 1 minus the significance level.
- $H_0: Pr(X = q) = p$ (the p -th quantile is q)
 $H_1: Pr((X < q) < p \text{ or } Pr((X < q) > p$
 Reject H_0 if *probability* is less than or equal to half the significance level or greater than or equal to 1 minus half the significance level.

The assumptions are as follows:

1. The X_i 's form a random sample; i.e., they are independent and identically distributed.
2. Measurement scale is at least ordinal; i.e., an ordering less than, greater than, and equal to exists in the observations.

Many uses for the sign test are possible with various values of p and q . For example, to perform a matched sample test that the difference of the medians of Y and Z is 0.0, let $p = 0.5$, $q = 0.0$, and $X_i = Y_i - Z_i$ in matched observations Y and Z . To test that the median difference is c , let $q = c$.

Examples

Example 1

This example tests the hypothesis that at least 50 percent of a population is negative. Because $0.18 < 0.95$, the null hypothesis at the 5-percent level of significance is not rejected.

```
x=[92, 139, -6, 10, 81, -11, 45, -25, -4, $
    22, 2, 41, 13, 8, 33, 45, -33, -45, -12]
PRINT, 'Probability = ', IMSL_SIGNTEST(x)

Probability =          0.179642
```

Example 2

This example tests the null hypothesis that at least 75 percent of a population is negative. Because $0.923 < 0.95$, the null hypothesis at the 5-percent level of significance is rejected.

```
x=[92, 139, -6, 10, 81, -11, 45, -25, -4, $
    22, 2, 41, 13, 8, 33, 45, -33, -45, -12]
probability = IMSL_SIGNTEST(x, Percentage = 0.75, $
    Percentile = 0, N_Pos_Dev = np, N_Zero_Dev = nz)
PM, probability, Title = 'Probability'
PM, np, Title = 'Number of Positive Deviations'
PM, nz, Title = 'Number of Ties'

Probability
    0.922543
Number of Positive Deviations
    12
Number of Ties
    0
```

Version History

6.4	Introduced

IMSL_WILCOXON

The IMSL_WILCOXON function performs a Wilcoxon rank sum test or a Wilcoxon signed rank test.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_WILCOXON( x1 [, x2] [, /DOUBLE] [, FUZZ=value]  
                        [, STATS=variable] )
```

Return Value

If a Wilcoxon rank sum test is performed, returns the two-sided p -value for the Wilcoxon rank sum statistic that is computed with average ranks used in the case of ties.

If a Wilcoxon signed rank test is performed, returns an array of length two containing the following values:

- The asymptotic probability of not exceeding the standardized (to an asymptotic variance of 1.0) minimum of ($W+$, $W-$) using method 1 under the null hypothesis that the distribution is symmetric about 0.0.
- And, the asymptotic probability of not exceeding the standardized (to an asymptotic variance of 1.0) minimum of ($W+$, $W-$) using method 2 under the null hypothesis that the distribution is symmetric about 0.0.

Arguments

x1

One-dimensional array containing the first sample.

x2

(Optional) One-dimensional array containing the second sample.

Keywords

DOUBLE

If present and nonzero, double precision is used.

FUZZ

Nonnegative constant used to determine ties in computing ranks in the combined samples. A tie is declared when two observations in the combined sample are within *Fuzz* of each other. Default: $Fuzz = 100 \times \epsilon \times \max \{ |x_{i1}|, |x_{j2}| \}$, where ϵ is machine precision for a Wilcoxon rank sum test, and $Fuzz = 0.0$ for a Wilcoxon signed rank test.

STATS

Named variable into which one-dimensional array of length 10 containing the statistics shown in [Table 18-1](#) and [Table 18-2](#) is stored. If a Wilcoxon rank sum test is performed:

Row	Statistics
0	Wilcoxon <i>W</i> statistic (the sum of the ranks of the <i>x</i> observations) adjusted for ties in such a manner that <i>W</i> is as small as possible
1	$2 \times E(W) - W$, where $E(W)$ is the expected value of <i>W</i>
2	probability of obtaining a statistic less than or equal to $\min \{W, 2 \times E(W) - W\}$
3	<i>W</i> statistic adjusted for ties in such a manner that <i>W</i> is as large as possible
4	$2 \times E(W) - W$, where $E(W)$ is the expected value of <i>W</i> , adjusted for ties in such a manner that <i>W</i> is as large as possible
5	probability of obtaining a statistic less than or equal to $\min \{W, 2 \times E(W) - W\}$, adjusted for ties in such a manner that <i>W</i> is as large as possible
6	<i>W</i> statistic with average ranks used in case of ties
7	estimated standard error of <i>Stats</i> (6) under the null hypothesis of no difference

Table 18-1: Stats Values for Wilcoxon Rank Sum Test

Row	Statistics
8	standard normal score associated with <i>Stats</i> (6)
9	two-sided p-value associated with <i>Stats</i> (6)

Table 18-1: Stats Values for Wilcoxon Rank Sum Test (Continued)

If a Wilcoxon signed rank test is performed:

Row	Statistics
0	The positive rank sum, W^+ , using method 1.
1	The absolute value of the negative rank sum, W^- , using method 1.
2	The standardized (to an asymptotic variance of 1.0) minimum of (W^+ , W^-) using method 1.
3	The asymptotic probability of not exceeding <i>stats</i> (2) under the null hypothesis that the distribution is symmetric about 0.0.
4	The positive rank sum, W^+ , using method 2.
5	The absolute value of the negative rank sum, W^- , using method 2.
6	The standardized (to an asymptotic variance of 1.0) minimum of (W^+ , W^-) using method 2.
7	The asymptotic probability of not exceeding <i>stats</i> (6) under the null hypothesis that the distribution is symmetric about 0.0.
8	The number of zero observations.
9	The total number of observations that are tied, and that are not within fuzz of zero.

Table 18-2: Stats Values for Wilcoxon Signed Rank Test

Discussion

If Two Positional Arguments Are Supplied

The `IMSL_WILCOXON` function performs the Wilcoxon rank sum test for identical population distribution functions. The Wilcoxon test is a linear transformation of the Mann-Whitney U test. If the difference between the two populations can be attributed solely to a difference in location, then the Wilcoxon test becomes a test of equality of the population means (or medians) and is the nonparametric equivalent of the two-sample t -test. The `IMSL_WILCOXON` function obtains ranks in the combined sample after first eliminating missing values from the data. The rank sum statistic is then computed as the sum of the ranks in the $x1$ sample.

Three methods for handling ties are used. (A tie is counted when two observations are within *Fuzz* of each other.) Method 1 uses the largest possible rank for tied observations in the smallest sample, while Method 2 uses the smallest possible rank for these observations. Thus, the range of possible rank sums is obtained. Method 3 for handling tied observations between samples uses the average rank of the tied observations. Asymptotic standard normal scores are computed for the W score (based on a variance that has been adjusted for ties) when average ranks are used (see Conover 1980, p. 217). The probability associated with the two-sided alternative is then computed.

Hypothesis Tests

In each of the tests listed in [Table 18-3](#), the first line gives the hypothesis (and its alternative) under the assumptions 1 to 3 below, while the second line gives the hypothesis when assumption 4 is also true. The rejection region is the same for both hypotheses and is given in terms of Method 3 for handling ties. Another output statistic should be used, (*Stats*(0) or *Stats*(3)), if another method for handling ties is desired.

Test	Null Hypothesis	Alternative Hypothesis	Action
1	$H_0 : Pr(x1 < x2) = 0.5$ $H_0 : E(x1) = E(x2)$	$H_1 : Pr(x1 < x2) \neq 0.5$ $(H_1 : E(x1) \neq E(x2))$	Reject if <i>Stats</i> (9) is less than the significance level of the test. Alternatively, reject the null hypothesis if <i>Stats</i> (6) is too large or too small.

Table 18-3: Hypothesis Tests

Test	Null Hypothesis	Alternative Hypothesis	Action
2	$H_0 : Pr(x1 < x2) \leq 0.5$	$H_1 : Pr(x1 < x2) > 0.5$	Reject if Stats (6) is too small.
	$H_0 : E(x1) \geq E(x2)$	$H_1 : E(x1) < E(x2)$	
3	$H_0 : Pr(x1 < x2) \geq 0.5$	$H_1 : Pr(x1 < x2) < 0.5$	Reject if Stats (6) is too large.
	$H_0 : E(x1) \leq E(x2)$	$H_1 : E(x1) > E(x2)$	

Table 18-3: Hypothesis Tests (Continued)

Assumptions

1. $x1$ and $x2$ contain random samples from their respective populations.
2. All observations are mutually independent.
3. The measurement scale is at least ordinal (i.e., an ordering less than, greater than, or equal to exists among the observations).
4. If $f(x)$ and $g(y)$ are the distribution functions of x and y , then $g(y) = f(x + c)$ for some constant c (i.e., the distribution of y is, at worst, a translation of the distribution of x).

Tables of critical values of the W statistic are given in the references for small samples.

If One Positional Argument is Supplied

The `IMSL_WILCOXON` function performs a Wilcoxon signed rank test of symmetry about zero. In one sample, this test can be viewed as a test that the population median is zero. In matched samples, a test that the medians of the two populations are equal can be computed by first computing difference scores. These difference scores would then be used as input to `IMSL_WILCOXON`. A general reference for the methods used is Conover (1980).

Routine `IMSL_WILCOXON` computes statistics for two methods for handling zero and tied observations. In the first method, observations within *Fuzz* of zero are not counted, and the average rank of tied observations is used. (Observations within *Fuzz* of each other are said to be tied.) In the second method, observations within *Fuzz* of

zero are randomly assigned a positive or negative sign, and the ranks of tied observations are randomly permuted.

The W^+ and W^- statistics are computed as the sums of the ranks of the positive observations and the sum of the ranks of the negative observations, respectively. Asymptotic probabilities are computed using standard methods (see, e.g., Conover 1980, page 282).

Hypothesis Tests

The W^+ and W^- statistics may be used to test the following hypotheses about the median, M . In deciding whether to reject the null hypothesis, use the bracketed statistic if method 2 for handling ties is preferred. Possible null hypotheses and alternatives are given as follows:

- $H_0 : M \leq 0$
 $H_1 : M > 0$
- Reject if `stats(0)` [or `stats(4)`] is too large.
- $H_0 : M \geq 0$
 $H_1 : M < 0$
- Reject if `stats(1)` [or `stats(5)`] is too large.
- $H_0 : M = 0$
 $H_1 : M \neq 0$
- Reject if `stats(2)` [or `stats(6)`] is too small. Alternatively, if an asymptotic test is desired, reject if `2*stats(3)` [or `2*stats(7)`] is less than the significance level.

Tabled values of the test statistic can be found in the references. If possible, tabled values should be used. If the number of nonzero observations is too large, then the asymptotic probabilities computed by `IMSL_WILCOXON` can be used.

Assumptions

The assumptions required for the hypothesis tests are as follows:

1. The distribution of each X_i is symmetric.
2. The X_i are mutually independent.
3. All X_i 's have the same median.
4. An ordering of the observations exists (i.e., $X_1 > X_2$ and $X_2 > X_3$ implies that $X_1 > X_3$).

If other assumptions are made, related hypotheses that are more (or less) restrictive can be tested.

Examples

Example 1

The following example is taken from Conover (1980, p. 224). It involves the mixing time of two mixing machines using a total of 10 batches of a certain kind of batter, five batches for each machine. The null hypothesis is not rejected at the 5-percent level of significance. The warning error is always printed when one or more ties are detected.

```
x1 = [7.3, 6.9, 7.2, 7.8, 7.2]
x2 = [7.4, 6.8, 6.9, 6.7, 7.1]
p = IMSL_WILCOXON(x1, x2, Stats = stats)
PRINT, 'p-Value = ', p

p-Value =          0.141238
```

Example 2

The following example uses the same data as the previous example. Now, all the statistics are output in the array *Stats*. First, a procedure is defined to output the results.

```
.RUN
PRO print_results, stats
  PRINT, 'Wilcoxon W Statistic .....', stats(0)
  PRINT, '2*E(W) - W .....', stats(1)
  PRINT, 'P-Value .....', stats(2)
  PRINT, 'Adjusted Wilcoxon Statistic..', stats(3)

  PRINT, 'Adjusted 2*E(W) - W .....', stats(4)
  PRINT, 'Adjusted P-Value .....', stats(5)
  PRINT, 'W Statistics for Averaged Ranks ..', stats(6)
  PRINT, 'Std Error of W (Averaged Ranks) ..', stats(7)
  PRINT, 'Std Normal Score of W (Averaged Ranks)..' , stats(8)
  PRINT, 'Two-Sided P-Value of W (Averaged Ranks) ..', stats(9)
END

x1 = [7.3, 6.9, 7.2, 7.8, 7.2]
x2 = [7.4, 6.8, 6.9, 6.7, 7.1]
p = IMSL_WILCOXON(x1, x2, Stats = stats)
print_results, stats

Wilcoxon W Statistic ..... 34.0000
2*E(W) - W ..... 21.0000
P-Value ..... 0.110072
Adjusted Wilcoxon Statistic ..... 35.0000
Adjusted 2*E(W) - W ..... 20.0000
```

```

Adjusted P-Value ..... 0.0745036
W Statistics for Averaged Ranks ..... 34.5000
Std Error of W (Averaged Ranks) ..... 4.75803
Std Normal Score of W (Averaged Ranks)... 1.47120
Two-Sided P-Value of W (Averaged Ranks). 0.141238

```

Example 3

This example illustrates the application of the Wilcoxon signed rank test to a test on a difference of two matched samples (matched pairs) $\{X1 = 223, 216, 211, 212, 209, 205, 201; \text{ and } X2 = 208, 205, 202, 207, 206, 204, 203\}$. A test that the median difference is 10.0 (rather than 0.0) is performed by subtracting 10.0 from each of the differences prior to calling `IMSL_WILCOXON`. As can be seen from the output, the null hypothesis is rejected. The warning error will always be printed when the number of observations is 50 or less unless printing is turned off for warning errors.

```

.RUN
PRO output_results, stats
  PRINT, 'Statistic' , Method 1 , Method2'
  PRINT, 'W+ .....', stats(0), stats(4)
  PRINT, 'W- .....', stats(1), stats(5)
  PRINT, 'Standardized Minimum...', stats(2), stats(6)
  PRINT, 'p-value .....', stats(3), stats(7)
  PRINT
  PRINT, 'Number of zeros .....', stats(8)
  PRINT, 'Number of ties .....', stats(9)
END

x = [-25.0, -21.0, -19.0, -15.0, -13.0, -11.0, -8.0]
p = IMSL_WILCOXON(x, Fuzz = 0.0001, Stats = stats)
OUTPUT_RESULTS, stats

Statistic          Method 1      Method 2
W+ .....0.00000    0.00000
W- .....28.0000    28.0000
Standardized Minimum ... -2.36643  -2.36643
p-value ..... 0.00898023  0.00898024

Number of zeros .....0.00000
Number of ties .....0.00000

```

Errors

Warning Errors

`STAT_AT_LEAST_ONE_TIE`—At least one tie is detected between the samples.

Fatal Errors

STAT_ALL_X_Y_MISSING—Each element of *x1* and/or *x2* is a missing NaN (Not a Number) value.

Version History

6.4	Introduced

IMSL_NCTRENDS

The IMSL_NCTRENDS function performs the Noether test for cyclical trend.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
result = IMSL_NCTRENDS(x [, /DOUBLE] [, FUZZ=value]  
                        [, NMISSING=variable] [, NSTAT=variable])
```

Return Value

One-dimensional array of length 3 containing the probabilities of *Nstat*(1) or more, *Nstat*(2) or more, or *Nstat*(3) or more monotonic sequences. If *Nstat*(0) is less than 1, *Result*(0) is set to NaN (not a number).

Arguments

x

One-dimensional array containing the data in chronological order.

Keywords

DOUBLE

If present and nonzero, double precision is used.

FUZZ

Nonnegative constant used to determine ties in computing ranks in the combined samples. A tie is declared when two observations in the combined sample are within *Fuzz* of each other. Default: *Fuzz* = 0.0.

NMISSING

Named variable into which the number of missing values in *x* is stored.

NSTAT

Named variable into which the one-dimensional array of length 6 containing the statistics below is stored:

- *Nstat (0)*—The number of consecutive sequences of length three used to detect cyclical trend when tying middle elements are eliminated from the sequence, and the next consecutive observation is used.
- *Nstat (1)*—The number of monotonic sequences of length three in the set defined by *Nstat(0)*.
- *Nstat (2)*—The number of nonmonotonic sequences where tied threesomes are counted as nonmonotonic.
- *Nstat (3)*—The number of monotonic sequences where tied threesomes are counted as monotonic.
- *Nstat (4)*—The number of middle observations eliminated because they were tied in forming the *Nstat(0)* sequences.
- *Nstat (5)*—The number of tied sequences found in forming the *Nstat(2)* and *Nstat(3)* sequences. A sequence is called a tied sequence if the middle element is tied with either of the two other elements.

Discussion

Routine IMSL_NCTRENDS performs the Noether test for cyclical trend (Noether 1956) for a sequence of measurements. In this test, the observations are first divided into sets of three consecutive observations. Each set is then inspected, and if the set is monotonically increasing or decreasing, the count variable is made incremental.

The count variables, *Nstat(1)*, *Nstat(2)*, and *Nstat(3)*, differ in the manner in which ties are handled. A tie can occur in a set (of size three) only if the middle element is tied with either of the two ending elements. Tied ending elements are not considered. In *Nstat(1)*, tied middle observations are eliminated, and a new set of size 3 is obtained by using the next observation in the sample. In *Nstat(2)*, the original set of size three is used, and tied middle observations are counted as nonmonotonic. In *Nstat(3)*, tied middle observations are counted as monotonic.

The probabilities of occurrence of the counts are obtained from the binomial distribution with $p = 1/3$, where p is the probability that a random sample of size three from a continuous distribution is monotonic. The binomial sample size is, of course, the number of sequences of size three found (adjusted for ties).

Hypothesis test:

$$H_0 : q = \Pr(X_i > X_{i-1} > X_{i-2}) + \Pr(X_i < X_{i-1} < X_{i-2}) \leq 1/3 \quad H_I : q > 1/3$$

Reject if *Result*(0) (or *Result*(1) or *Result*(2) depending on the method used for handling ties) is less than the significance level of the test.

Assumption: The observations are independent and are from a continuous distribution.

Example

A test for cyclical trend in a sequence of 1000 randomly generated observations is performed. Because of the sample used, there are no ties and all three test statistics yield the same result.

```
IMSL_RANDOMOPT, set = 123457
x = IMSL_RANDOM(1000, /Uniform)
pval = IMSL_NCTRENDS(x, Nstat = nstat)
PM, pval
PM, nstat

0.697881
0.697881
0.697881

333
107
107
107
0
0
```

Version History

6.4	Introduced

IMSL_CSTRENDS

The IMSL_CSTRENDS function performs the Cox and Stuart sign test for trends in location and dispersion.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_CSTRENDS(x [, /DOUBLE] [, DISPERSION=array]  
[, FUZZ=value] [, NMISsing=variable] [, NSTAT=variable])
```

Return Value

One-dimensional array of length 8 containing the probabilities.

The first four elements of *Result* are computed from two groups of observations.

- 0—Probability of $Nstat(0) + Nstat(2)$ or more negative signs (ties are considered negative).
- 1—Probability of obtaining $Nstat(1)$ or more positive signs (ties are considered negative).
- 2—Probability of $Nstat(0) + Nstat(2)$ or more negative signs (ties are considered positive).
- 3 —Probability of obtaining $Nstat(1)$ or more positive signs (ties are considered positive).

The last four elements of *Result* are computed from three groups of observations.

- 4—Probability of $Nstat(0) + Nstat(2)$ or more negative signs (ties are considered negative).
- 5—Probability of obtaining $Nstat(1)$ or more positive signs (ties are considered negative).
- 6—Probability of $Nstat(0) + Nstat(2)$ or more negative signs (ties are considered positive).

- 7—Probability of obtaining $Nstat(1)$ or more positive signs (ties are considered positive).

Arguments

x

One-dimensional array containing the data in chronological order.

Keywords

DOUBLE

If present and nonzero, double precision is used.

DISPERSION

A one-dimensional array of length 2. If *Dispersion* is set, the Cox and Stuart tests for trends in dispersion are computed. Otherwise, as default, the Cox and Stuart tests for trends in location are computed.

$k = Dispersion(0)$ is the number of consecutive x elements to be used to measure dispersion. If $ids = Dispersion(1)$ is zero, the range is used as a measure of dispersion. Otherwise, the centered sum of squares is used.

FUZZ

A nonnegative constant used to determine when elements in x are tied. If $|x(i) - x(j)|$ is less than or equal to *Fuzz*, $x(i)$ and $x(j)$ are said to be tied. *Fuzz* must be nonnegative. Default: $Fuzz = 0.0$.

NMISSING

Named variable into which the number of missing values in x is stored.

NSTAT

Named variable into which the one-dimensional array of length 8 containing the statistics below is stored:

- 0—Number of negative differences (two groups)
- 1—Number of positive differences (two groups)
- 2—Number of zero differences (two groups)

- 3—Number of differences used to calculate *Result*(0) through *Result*(3) (two groups).
- 4—Number of negative differences (three groups)
- 5—Number of positive differences (three groups)
- 6—Number of zero differences (three groups)
- 7—Number of differences used to calculate *Result*(4) through *Result*(7) (three groups).

Discussion

The IMSL_CSTRENDS function tests for trends in dispersion or location in a sequence of random variables depending upon the usage of *Dispersion*. A derivative of the sign test is used (see Cox and Stuart 1955).

Location Test

For the location test (Default) with two groups, the observations are first divided into two groups with the middle observation thrown out if there are an odd number of observations. Each observation in group one is then compared with the observation in group two that has the same lexicographical order. A count is made of the number of times a group-one observation is less than (*Nstat*(0)), greater than (*Nstat*(1)), or equal to (*Nstat*(2)), its counterpart in group two. Two observations are counted as equal if they are within *Fuzz* of one another.

In the three-group test, the observations are divided into three groups, with the center group losing observations if the division is not exact. The first and third groups are then compared as in the two-group case, and the counts are stored in *Nstat*(4) through *Nstat*(6).

Probabilities in *Result* are computed using the binomial distribution with sample size equal to the number of observations in the first group (*Nstat*(3) or *Nstat*(7)), and binomial probability $p = 0.5$.

Dispersion Test

The dispersion tests (when keyword *Dispersion* is set) proceed exactly as with the tests for location, but using one of two derived dispersion measures. The input value $k = \text{Dispersion}(0)$ is used to define $N\_ELEMENTS(x)/k$ groups of consecutive observations starting with observation 1. The first k observations define the first group, the next k observations define the second group, etc., with the last observations omitted if $N\_ELEMENTS(x)$ is not evenly divisible by k . A dispersion score is then computed for each group as either the range (*ids* = 0), or a multiple of the variance

(ids $\neq 0$) of the observations in the group. The dispersion scores form a derived sample. The tests proceed on the derived sample as above.

Ties

Ties are defined as occurring when a group one observation is within *Fuzz* of its last group counterpart. Ties imply that the probability distribution of x is not strictly continuous, which means that $\Pr(x_1 > x_2) \neq 0.5$ under the null hypothesis of no trend (and the assumption of independent identically distributed observations). When ties are present, the computed binomial probabilities are not exact, and the hypothesis tests will be conservative.

Hypothesis tests

In the following, i indexes an observation from group 1, while j indexes the corresponding observation in group 2 (two groups) or group 3 (three groups).

- $H_0 : \Pr(X_i > X_j) = \Pr(X_i < X_j) = 0.5$
 $H_1 : \Pr(X_i > X_j) < \Pr(X_i < X_j)$
 Hypothesis of upward trend. Reject if *Result*(2) (or *Result*(6)) is less than the significance level.
- $H_0 : \Pr(X_i > X_j) = \Pr(X_i < X_j) = 0.5$
 $H_1 : \Pr(X_i > X_j) > \Pr(X_i < X_j)$
 Hypothesis of downward trend. Reject if *Result*(1) (or *Result*(5)) is less than the significance level.
- $H_0 : \Pr(X_i > X_j) = \Pr(X_i < X_j) = 0.5$
 $H_1 : \Pr(X_i > X_j) \neq \Pr(X_i < X_j)$
 Two tailed test. Reject if $2 \max(\text{Result}(1), \text{Result}(2))$ (or $2 \max(\text{Result}(5), \text{Result}(6))$) is less than the significance level.

Assumptions

1. The observations are a random sample; i.e., the observations are independently and identically distributed.
2. The distribution is continuous.

Example

This example illustrates both the location and dispersion tests. The data, which are taken from Bradley (1968), page 176, give the closing price of AT&T on the New York stock exchange for 36 days in 1965. Tests for trends in location (Default), and for trends in dispersion (*Dispersion*) are performed. Trends in location are found.


```

x = [9.5, 9.875, 9.25, 9.5, 9.375, 9.0, 8.75, 8.625, 8.0, $
      8.25, 8.25, 8.375, 8.125, 7.875, 7.5, 7.875, 7.875, $
      7.75,7.75, 7.75, 8.0, 7.5,7.5, 7.125, 7.25, 7.25, 7.125, $
      6.75,6.5, 7.0, 7.0, 6.75, 6.625, 6.625,7.125, 7.75]
k = 2
ids = 0
pstat = IMSL_CSTRENDS(x, Nstat = nstat)
PM, nstat, Title = '          NSTAT'
PM, pstat, Title = '          PSTAT'
pstat = IMSL_CSTRENDS(x, Nstat = nstat, Dispersion = [k, ids])
PM, nstat, Title = '          NSTAT'
PM, pstat, Title = '          PSTAT'

          NSTAT
          0
          17
          1
          18
          0
          12
          0
          12

          PSTAT
          0.999996
          7.24792e-05
          1.00000
          3.81470e-06
          1.00000
          0.000244141
          1.00000
          0.000244141

          NSTAT
          4
          3
          2
          9
          4
          2
          0
          6

          PSTAT
          0.253906
          0.910156
          0.746094
          0.500000
          0.343750

```

0.890625
0.343750
0.890625

Version History

6.4	Introduced

IMSL_TIE_STATS

The IMSL_TIE_STATS function computes tie statistics for a sample of observations.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_TIE_STATS(*x* [, /DOUBLE] [, FUZZ=*value*])

Return Value

One-dimensional array of length 4 containing the tie statistics.

$$\text{result}(0) = \sum_{j=1}^{\tau} [t_j(t_j - 1)]/2$$

$$\text{result}(1) = \sum_{j=1}^{\tau} ([t_j(t_j - 1)](t_j + 1))/2$$

$$\text{result}(2) = \sum_{j=1}^{\tau} t_j(t_j - 1)(2t_j + 5)$$

$$\text{result}(3) = \sum_{j=1}^{\tau} t_j(t_j - 1)(t_j - 2)$$

where t_j is the number of ties in the j -th group (rank) of ties, and τ is the number of tie groups in the sample.

Arguments

x

One-dimensional array containing the observations. *x* must be ordered monotonically increasing with all missing values removed.

Keywords

DOUBLE

If present and nonzero, double precision is used.

FUZZ

Nonnegative constant used to determine ties. Observations i and j are tied if the successive differences $x(k + 1) - x(k)$ between observations i and j , inclusive, are all less than $Fuzz$. Default: $Fuzz = 0.0$

Discussion

The `IMSL_TIE_STATS` function computes tie statistics for a monotonically increasing sample of observations. “Tie statistics” are statistics that may be used to correct a continuous distribution theory nonparametric test for tied observations in the data. Observations i and j are tied if the successive differences $x(k + 1) - x(k)$, inclusive, are all less than $Fuzz$. Note that if each of the monotonically increasing observations is equal to its predecessor plus a constant, if that constant is less than $Fuzz$, then all observations are contained in one tie group. For example, if $Fuzz = 0.11$, then the following observations are all in one tie group.

0.0, 0.10, 0.20, 0.30, 0.40, 0.50, 0.60, 0.70, 0.80, 0.90, 1.00

Example

We want to compute tie statistics for a sample of length 7.

```
fuzz = 0.001
x = [1.0, 1.0001, 1.0002, 2.0, 3.0, 3.0, 4.0]
tstat = IMSL_TIE_STATS(x, FUZZ = fuzz)
PRINT, tstat

4.00000      2.50000      84.0000      6.00000
```

Version History

6.4	Introduced

IMSL_KW_TEST

The IMSL_KW_TEST function performs a Kruskal-Wallis test¹ for identical population medians.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_KW_TEST(*n*, *y* [, /DOUBLE] [, FUZZ=*value*])

Return Value

One-dimensional array of length 4 containing the Kruskal-Wallis statistics.

- 0—Kruskal-Wallis H statistic.
- 1—Asymptotic probability of a larger H under the null hypothesis of identical population medians.
- 2—H corrected for ties.
- 3—Asymptotic probability of a larger H (corrected for ties) under the null hypothesis of identical populations

Arguments

n

One-dimensional array containing the number of responses for each of the groups.

y

One-dimensional array of length N_ELEMENTS(*n*) that contains the responses for each of the groups. *y* must be sorted by group, with the *n*(0) observations in group 1 coming first, the *n*(1) observations in group two coming second, and so on.

Keywords

DOUBLE

If present and nonzero, double precision is used.

FUZZ

Nonnegative constant used to determine ties in y . If (after sorting) $|y(i) - y(i + 1)|$ is less than or equal to $Fuzz$, then a tie is counted. Default: $Fuzz = 0.0$

Discussion

The `IMSL_KW_TEST` function generalizes the Wilcoxon two-sample test computed by “[IMSL_WILCOXON](#)” on page 837 to more than two populations. It computes a test statistic for testing that the population distribution functions in each of K populations are identical. Under appropriate assumptions, this is a nonparametric analogue of the one-way analysis of variance. Since more than two samples are involved, the alternative is taken as the analogue of the usual analysis of variance alternative, namely that the populations are not identical.

The calculations proceed as follows: All observations are ranked regardless of the population to which they belong. Average ranks are used for tied observations (observations within $Fuzz$ of each other). Missing observations (observations equal to NaN, not a number) are not included in the ranking. Let R_i denote the sum of the ranks in the i -th population. The test statistic H is defined as:

$$H = \frac{1}{S^2} \sum_{i=1}^K \left(\frac{R_i^2}{n_i} - \frac{N(N+1)^2}{4} \right)$$

where N is the total of the sample sizes, n_i is the number of observations in the i -th sample, and S^2 is computed as the (bias corrected) sample variance of the R_i .

The null hypothesis is rejected when $Result(3)$ (or $Result(1)$) is less than the significance level of the test. If the null hypothesis is rejected, then the procedures given in Conover (1980, page 231) may be used for multiple comparisons. The `IMSL_KW_TEST` function computes asymptotic probabilities using the chi-squared distribution when the number of groups is 6 or greater, and a Beta approximation (see Wallace 1959) when the number of groups is 5 or less. Tables yielding exact probabilities in small samples may be obtained from Owen (1962).

Example

The following example is taken from Conover (1980, page 231). The data represents the yields per acre of four different methods for raising corn. Since $H = 25.5$, the four methods are clearly different. The warning error is always printed when the Beta approximation is used, unless printing for warning errors is turned off.

```
y = [83.0, 91.0, 94.0, 89.0, 89.0, 96.0, 91.0, 92.0, 90.0, $
     91.0, 90.0, 81.0, 83.0, 84.0, 83.0, 88.0, 91.0, 89.0, $
     84.0, 101.0, 100.0, 91.0, 93.0, 96.0, 95.0, 94.0, 78.0, $
     82.0, 81.0, 77.0, 79.0, 81.0, 80.0, 81.0]
n   = [9, 10, 7, 8]
fuzz = 0.001
rlabel = ['H (no ties)      =', $
          'Prob (no ties)   =', $
          'H (ties)        =', $
          'Prob (ties)      =']
s = IMSL_KW_TEST(n, y, Fuzz = fuzz)
FOR i = 0, 3 DO PM, rlabel(i), s(i), FORMAT = '(A18, F6.2)'
```

```
H (no ties)      = 25.46
Prob (no ties)   = 0.00
H (ties)        = 25.63
Prob (ties)      = 0.00
```

Version History

6.4	Introduced

IMSL_FRIEDMANS_TEST

The IMSL_FRIEDMANS_TEST function performs Friedman's test for a randomized complete block design.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_FRIEDMANS_TEST(y [, ALPHA=value] [, DIFF=variable]  
                               [, /DOUBLE] [, FUZZ=value] [, STATS=variable] [, SUM_RANK=variable])
```

Return Value

The Chi-squared approximation of the asymptotic p -value for Friedman's two-sided test statistic.

Arguments

y

Two-dimensional array containing the observations. The first row of *y* contain the observations on treatments 1, 2, ..., N_ELEMENTS(*y*(0, *)) in the first block. The second row of *y* contain the observations in the second block, etc., and so on.

Keywords

ALPHA

Critical level for multiple comparisons. *Alpha* should be between 0 and 1 exclusive. Default: *Alpha* = 0.05.

DIFF

Named variable into which the minimum absolute difference in two elements of *Sum_Rank* to infer at the *Alpha* level of significance that the medians of the corresponding treatments are different is stored.

DOUBLE

If present and nonzero, double precision is used.

FUZZ

Nonnegative constant used to determine ties. In the ordered observations, if $|y(i) - y(i + 1)|$ is less than or equal to *Fuzz*, then $y(i)$ and $y(i + 1)$ are said to be tied. Default: *Fuzz* = 0.0.

STATS

Named variable into which the one-dimensional array of length 6 containing the Friedman statistics below is stored. Probabilities reported are computed under the appropriate null hypothesis.

- 0—Friedman two-sided test statistic.
- 1—Approximate *F* value for *Stats*(0).
- 2—Page test statistic for testing the ordered alternative that the median of treatment *i* is less than or equal to the median of treatment *i* + 1, with strict inequality holding for some *i*.
- 3—Asymptotic *p*-value for *Stats*(0). Chi-squared approximation.
- 4—Asymptotic *p*-value for *Stats*(1). *F* approximation.
- 5—Asymptotic *p*-value for *Stats*(2). Normal approximation.

SUM_RANK

Named variable into which a one-dimensional array of length *N_ELEMENTS(x(0, *))* containing the sum of the ranks of each treatment is stored.

Discussion

The *IMSL_FRIEDMANS_TEST* function may be used to test the hypothesis of equality of treatment effects within each block in a randomized block design. No missing values are allowed. Ties are handled by using the average ranks. The test statistic is the nonparametric analogue of an analysis of variance *F* test statistic.

The test proceeds by first ranking the observations within each block. Let A denote the sum of the squared ranks, i.e., let:

$$A = \sum_{i=1}^k \sum_{j=1}^b \text{Rank}(Y_{ij})^2$$

where $\text{Rank}(Y_{ij})$ is the rank of the i -th observation within the j -th block, b is the number of blocks, and k is the number of treatments. Let:

$$B = \frac{1}{b} \sum_{i=1}^k R_i^2$$

where:

$$R_i = \sum_{j=1}^b \text{Rank}(Y_{ij})$$

The Friedman test statistic ($\text{Stats}(0)$) is given by:

$$T = \frac{(k-1)(bB - b^2k(k+1)^2/4)}{A - bk(k+1)^2/4}$$

that, under the null hypothesis, has an approximate chi-squared distribution with $k-1$ degrees of freedom. The asymptotic probability of obtaining a larger chi-squared random variable is returned in $\text{Stats}(3)$.

If the F distribution is used in place of the chi-squared distribution, then the usual one way analysis of variance F -statistic computed on the ranks is used. This statistic, reported in $\text{Stats}(1)$, is given by:

$$F = \frac{(b-1)T}{b(k-1) - T}$$

and asymptotically follows an F distribution with $(k-1)$ and $(b-1)(k-1)$ degrees of freedom under the null hypothesis. $\text{Stats}(4)$ is the asymptotic probability of obtaining a larger F random variable. (If $A = B$, $\text{Stats}(0)$ and $\text{Stats}(1)$ are set to machine infinity, and the significance levels are reported as $k!/(k!)^b$, unless this computation would cause underflow, in which case the significance levels are reported as zero.) Iman and Davenport (1980) discuss the relative advantages of the chi-squared and F approximations. In general, the F approximation is considered best.

The Friedman T statistic is related both to the Kendall coefficient of concordance and to the Spearman rank correlation coefficient. See Conover (1980) for a discussion of the relationships.

If, at the $\alpha = \text{Alpha}$ level of significance, the Friedman test results in rejection of the null hypothesis, then an asymptotic test that treatments i and j are different is given by: reject H_0 if $|R_i - R_j| > D$, where:

$$D = t_{1-\alpha/2} \sqrt{2b(A-B)/((b-1)k-1)}$$

where t has $(b-1)(k-1)$ degrees of freedom. Page's statistic ($\text{Stats}(2)$) is used to test the same null hypothesis as the Friedman test but is sensitive to a monotonic increasing alternative. The Page test statistic is given by

$$Q = \sum_{i=1}^k jR_i$$

It is largest (and thus most likely to reject) when the R_i are monotonically increasing.

Assumptions

The assumptions in the Friedman test are as follows:

1. The k -vectors of responses within each of the b blocks are mutually independent (i.e., the results within one block have no effect on the results within another block).
2. Within each block, the observations may be ranked.

The hypothesis tested is that each ranking of random variables within each block is equally likely. The alternative is that at least one treatment tends to have larger values than one or more of the other treatments. The Friedman test is a test for the equality of treatment means or medians.

Example

The following example is taken from Bradley (1968), page 127, and tests the hypothesis that 4 drugs have the same effects upon a person's visual acuity. Five subjects were used.

```
y = TRANSPOSE([[0.39, 0.55, 0.33, 0.41], $
               [0.21, 0.28, 0.19, 0.16], [0.73, 0.69, 0.64, 0.62], $
               [0.41, 0.57, 0.28, 0.35], [0.65, 0.57, 0.53, 0.60]])
fuzz = 0.001
p = IMSL_FRIEDMANS_TEST(y, Fuzz = fuzz, Diff = diff, $
                        Sum_Rank = sr, Stats = stat)
PM, stat, Title = 'STATS'
PM, diff, Title = 'DIFF'
PM, sr, Title = 'Sum_Rank'
```

```
STATS
```

8.28000
4.92857
111.000
0.0405658
0.0185906
0.984954

DIFF
6.65638

Sum_Rank
16.0000
17.0000
7.00000
10.0000

The Friedman null hypothesis is rejected at the $\alpha = 0.05$ while the Page null hypothesis is not. (A Page test with a monotonic decreasing alternative would be rejected, however.) Using *Sum_Rank* and *Diff*, one can conclude that treatment 3 is different from treatments 1 and 2, and that treatment 4 is different from treatment 2, all at the $\alpha= 0.05$ level of significance.

Version History

6.4	Introduced

IMSL_COCHRANQ

The IMSL_COCHRANQ function performs a Cochran Q test for related observations.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_COCHRANQ(*x* [, /DOUBLE] [, *Q*=*variable*])

Return Value

The p -value for the Cochran Q statistic.

Arguments

X

Two-dimensional array containing the matrix of dichotomized data.

Keywords

DOUBLE

If present and nonzero, double precision is used.

Q

Named variable into which the Cochran's Q statistic is stored.

Discussion

The IMSL_COCHRANQ function computes the Cochran Q test statistic that may be used to determine whether or not M matched sets of responses differ significantly among themselves. The data may be thought of as arising out of a randomized block design in which the outcome variable must be success or failure, coded as 1.0 and 0.0, respectively. Within each block, a multivariate vector of 1's or 0's is observed. The

hypothesis is that the probability of success within a block does not depend upon the treatment.

Assumptions

1. The blocks are a random sample from the population of all possible blocks.
2. The outcome of each treatment is dichotomous.

Hypothesis

The hypothesis being tested may be stated in at least two ways.

1. H_0 : All treatments have the same effect.
 H_1 : The treatments do not all have the same effect.
2. Let p_{ij} denote the probability of outcome 1.0 in block i , treatment j .
 $H_0: p_{i1} = p_{i2} = \dots = p_{ic}$ for each i .
 $H_1: p_{ij} \neq p_{ik}$ for some i , and some $j \neq k$.
 where c (equal to $N\_ELEMENTS(x(0, *))$) is the number of treatments.

The null hypothesis is rejected if Cochran's Q statistic is too large.

Remarks

1. The input data must consist of zeros and ones only. For example, let $n\_variables = N\_ELEMENTS(x(0, *))$ and $n\_observations = N\_ELEMENTS(x(*, 0))$, then the data may be pass-fail information on $n\_variables$ questions asked of $n\_observations$ people or the test responses of $n\_observations$ individuals to $n\_variables$ different conditions.
2. The resulting statistic is distributed approximately as chi-squared with $n\_variables - 1$ degrees of freedom if $n\_observations$ is not too small. $n\_observations$ greater than or equal to $5 \times n\_variables$ is a conservative recommendation.

Example

The following example is taken from Siegal (1956, p. 164). It measures the responses of 18 women to 3 types of interviews.

```
x = TRANSPOSE([ [0.0, 0.0, 0.0], [1.0, 1.0, 0.0], $
  [0.0, 1.0, 0.0], [0.0, 0.0, 0.0], $
  [1.0, 0.0, 0.0], [1.0, 1.0, 0.0], $
  [1.0, 1.0, 0.0], [0.0, 1.0, 0.0], $
  [1.0, 0.0, 0.0], [0.0, 0.0, 0.0], $
  [1.0, 1.0, 1.0], [1.0, 1.0, 1.0], $
```

```

[1.0, 1.0, 0.0], [1.0, 1.0, 0.0], $
[1.0, 1.0, 0.0], [1.0, 1.0, 1.0], $
[1.0, 1.0, 0.0], [1.0, 1.0, 0.0]])
pq = IMSL_COCHRANQ(x)
PRINT, 'pq =', pq

pq = 0.000240266

```

Errors

Warning Errors

STAT_ALL_0_OR_1—“ x ” consists of either all ones or all zeros. “ q ” is set to NaN (not a number). “*Result*” is set to 1.0.

Fatal Errors

STAT_INVALID_X_VALUES—“ $x(\#, \#)$ ” = #. “ x ” must consist of zeros and ones only.

Version History

6.4	Introduced

IMSL_KTRENDS

The IMSL_KTRENDS function performs a k-sample trends test against ordered alternatives.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_KTRENDS(*n*, *y* [, /DOUBLE])

Return Value

One-dimensional array of length 17 containing the test results.

- 0—Test statistic (ties are randomized).
- 1—Conservative test statistic with ties counted in favor of the null hypothesis.
- 2—*p*-value associated with *Result*(0).
- 3—*p*-value associated with *Result*(1).
- 4—Continuity corrected *Result*(2).
- 5—Continuity corrected *Result*(3).
- 6—Expected mean of the statistic.
- 7—Expected kurtosis of the statistic. (The expected skewness is zero.)
- 8—Total sample size.
- 9—Coefficient of rank correlation based upon *Result*(0).
- 10—Coefficient of rank correlation based upon *Result*(1).
- 11—Total number of ties between samples.
- 12—The t-statistic associated with *Result*(2).
- 13—The t-statistic associated with *Result*(3).
- 14—The t-statistic associated with *Result*(4).
- 15—The t-statistic associated with *Result*(5).

- 16—Degrees of freedom for each t-statistic.

Arguments

n

One-dimensional array containing the number of responses for each of the groups.

y

One-dimensional array that contains the responses for each of the groups. *y* must be sorted by group, with the *n*(0) observations in group 1 coming first, the *n*(1) observations in group two coming second, and so on.

Keywords

DOUBLE

If present and nonzero, double precision is used.

Discussion

The IMSL_KTRENDS function performs a *k*-sample trends test against ordered alternatives. The alternative to the null hypothesis of equality is that $F_1(X) < F_2(X) < \dots < F_k(X)$, where F_1, F_2 , etc., are cumulative distribution functions, and the operator $<$ implies that the less than relationship holds for all values of *x*. While the trends test used in IMSL_KTRENDS requires that the background populations be continuous, ties occurring within a sample have no effect on the test statistic or associated probabilities. Ties between samples are important, however. Two methods for handling ties between samples are used. These are:

1. Ties are randomly split (*Result*(0)).
2. Ties are counted in a manner that is unfavorable to the alternative hypothesis (*Result*(1)).

Computational Procedure

Consider the matrices:

$$M^{km} = (m_{ij}^{km}) = \begin{pmatrix} 2 & \text{if } X_{ki} < X_{mj} \\ 0 & \text{otherwise} \end{pmatrix}$$

where X_{ki} is the i -th observation in the k -th population, X_{mj} is the j -th observation in the m -th population, and each matrix M^{km} is n_k by n_m where $n_i = n(i)$. Let S_{km} denote the sum of all elements in M^{km} . Then, *Result(1)* is computed as the sum over all elements in S_{km} , minus the expected value of this sum (computed as:

$$\sum_{k < m} n_k n_m$$

when there are no ties and the distributions in all populations are equal). In *Result(0)*, ties are broken randomly, and the element in the summation is taken as 2.0 or 0.0 depending upon the result of breaking the tie.

Result(2) and *Result(3)* are computed using the t distribution. The probabilities reported are asymptotic approximations based upon the t statistics in *Result(12)* and *Result(13)*, which are computed as in Jonckheere (1954, page 141).

Similarly, *Result(4)* and *Result(5)* give the probabilities for *Result(14)* and *Result(15)*, the continuity corrected versions of *Result(2)* and *Result(3)*. The degrees of freedom for each t statistic (*Result(16)*) are computed so as to make the t distribution selected as close as possible to the actual distribution of the statistic (see Jonckheere 1954, page 141).

Result(6), the variance of the test statistic *Result(0)*, and *Result(7)*, the kurtosis of the test statistic, are computed as in Jonckheere (1954, page 138). The coefficients of rank correlation in *Result(8)* and *Result(9)* reduce to the Kendall τ statistic when there are just two groups.

Exact probabilities in small samples can be obtained from tables in Jonckheere (1954). Note, however, that the t approximation appears to be a good one.

Assumptions

1. The X_{mi} for each sample are independently and identically distributed according to a single continuous distribution.
2. The samples are independent.

Hypothesis tests

$$H_0 : F_1(X) \geq F_2(X) \geq \dots \geq F_k(X)$$

$$H_1 : F_1(X) < F_2(X) < \dots < F_k(X)$$

Reject if *Result(2)* (or *Result(3)*, or *Result(4)* or *Result(5)*, depending upon the method used) is too large.

Example

The following example is taken from Jonckheere (1954, page 135). It involves four observations in four independent samples.

```

y = [19.0, 20.0, 60.0, 130.0, 21.0, 61.0, 80.0, 129.0, $
     40.0, 99.0, 100.0, 149.0, 49.0, 110.0, 151.0, 160.0]
n = [4, 4, 4, 4]
rlabel = ['stat(0) - Test Statistic (random) .....', $
          'stat(1) - Test Statistic (null hypothesis) ....', $
          'stat(2) - p-value for stat(0) .....', $
          'stat(3) - p-value for stat(1) .....', $
          'stat(4) - Continuity corrected for stat(2) ....', $
          'stat(5) - Continuity corrected for stat(3) ....', $
          'stat(6) - Expected mean .....', $
          'stat(7) - Expected kurtosis .....', $
          'stat(8) - Total sample size .....', $
          'stat(9) - Rank corr. coef. based on stat(0) ...', $
          'stat(10)- Rank corr. coef. based on stat(1) ...', $
          'stat(11)- Total number of ties .....', $
          'stat(12)- t-statistic associated w/stat(2) ....', $
          'stat(13)- t-statistic associated w/stat(3) ....', $
          'stat(14)- t-statistic associated w/stat(4) ....', $
          'stat(15)- t-statistic associated w/stat(5) ....', $
          'stat(16)- Degrees of freedom .....']
s = IMSL_KTRENDS(n, y)
FOR i = 0, 16 DO PM, rlabel(i), s(i), FORMAT = '(A45, F10.5)'

stat(0) - Test Statistic (random) ..... 46.00000
stat(1) - Test Statistic (null hypothesis) .. 46.00000
stat(2) - p-value for stat(0) ..... 0.01483
stat(3) - p-value for stat(1) ..... 0.01483
stat(4) - Continuity corrected for stat(2) .. 0.01683
stat(5) - Continuity corrected for stat(3) .. 0.01683
stat(6) - Expected mean ..... 458.66666
stat(7) - Expected kurtosis ..... -0.15365
stat(8) - Total sample size ..... 16.00000
stat(9) - Rank corr. coef. based on stat(0) . 0.47917
stat(10)- Rank corr. coef. based on stat(1) . 0.47917
stat(11)- Total number of ties ..... 0.00000
stat(12)- t-statistic associated w/stat(2) .. 2.26435
stat(13)- t-statistic associated w/stat(3) .. 2.26435
stat(14)- t-statistic associated w/stat(4) .. 2.20838
stat(15)- t-statistic associated w/stat(5) .. 2.20838
stat(16)- Degrees of freedom ..... 36.04963

```

Version History

6.4	Introduced



Chapter 19

Goodness of Fit

This section contains the following topics:

Overview: Goodness of Fit	874	Goodness of Fit Routines	875

Overview: Goodness of Fit

The routines in this chapter are used to test for goodness of fit and randomness. The goodness-of-fit tests are described in Conover (1980). There are two goodness-of-fit tests for general distributions, a Kolmogorov-Smirnov test and a chi-squared test. You will supply the hypothesized cumulative distribution function for these two tests. There are three routines that can be used to test specifically for the normal or exponential distributions.

The tests for randomness are often used to evaluate the adequacy of pseudorandom number generators. These tests are discussed in Knuth (1981).

The Kolmogorov-Smirnov routines in this chapter compute exact probabilities in small to moderate sample sizes. The chi-squared goodness-of-fit test may be used with discrete as well as continuous distributions.

The Kolmogorov-Smirnov and chi-squared goodness-of-fit test routines allow for missing values (NaN, not a number) in the input data. The routines that test for randomness do not allow for missing values.

Goodness of Fit Routines

General Goodness of Fit Tests

[IMSL_CHISQTEST](#)—Chi-squared goodness of fit test.

[IMSL_NORMALITY](#)—Shapiro-Wilk W test for normality.

[IMSL_KOLMOGOROV1](#)—One-sample continuous data Kolmogorov-Smirnov.

[IMSL_KOLMOGOROV2](#)—Two-sample continuous data Kolmogorov-Smirnov.

[IMSL_MVAR_NORMALITY](#)—Mardia's test for multivariate normality.

Tests for Randomness

[IMSL_RANDOMNESS_TEST](#)—Runs test, Paris-serial test, d^2 test or triplets tests.

IMSL_CHISQTEST

The IMSL_CHISQTEST function performs a chi-squared goodness-of-fit test.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_CHISQTEST(f, n_categories, x [, CELL_COUNTS=variable]
[, CELL_EXPECTED=variable] [, CELL_CHISQ=variable]
[, CHI_SQUARED=variable] [, CUTPOINTS=variable] [, DF=variable]
[, /DOUBLE] [, /EQUAL_CUTPOINTS] [, FREQUENCIES=variable]
[, LOWER_BOUND=value] [, N_PARAMS_ESTIMATED=value]
[, UPPER_BOUND=value] [, USED_CUTPOINTS=variable])
```

Return Value

The p -value for the goodness-of-fit chi-squared statistic.

Arguments

f

Scalar string specifying a user-supplied function. Function f accepts one scalar parameter and returns the hypothesized, cumulative distribution function at that point.

n_categories

Number of cells into which the observations are to be tallied.

x

One-dimensional array containing the vector of data elements for this test.

Keywords

CELL_COUNTS

Named variable into which the cell counts are stored. The cell counts are the observed frequencies in each of the $n\_categories$ cells.

CELL_EXPECTED

Named variable into which the cell expected values are stored. The expected value of a cell is the expected count in the cell given that the hypothesized distribution is correct.

CELL_CHISQ

Named variable into which an array of length $n\_categories$ containing the cell contributions to chi-squared are stored.

CHI_SQUARED

Named variable into which the chi-squared test statistic is stored.

CUTPOINTS

Specifies the named variable containing user-defined cutpoints to be used by `IMSL_CHISQTEST`. Keywords *Cutpoints* and *Equal_Cutpoints* cannot be used together.

DF

Named variable into which the degrees of freedom for the chi-squared goodness-of-fit test are stored.

DOUBLE

If present and nonzero, double precision is used.

EQUAL_CUTPOINTS

If present and nonzero, equal probability cutpoints are used. Keywords *Equal_Cutpoints* and *Cutpoints* cannot be used together.

FREQUENCIES

Named variable into which the array containing the vector frequencies for the observations stored in *x* is stored.

LOWER_BOUND

Lower bound of the range of the distribution. If *Lower Bound* = *Upper Bound*, a range on the whole real line is used (the default). If the lower and upper endpoints are different, points outside of the range of these bounds are ignored. Distributions conditional on a range can be specified when *Lower_Bound* and *Upper_Bound* are used. If *Lower_Bound* is specified, then *Upper_Bound* also must be specified. By convention, *Lower_Bound* is excluded from the first interval, but *Upper_Bound* is included in the last interval.

N_PARAMS_ESTIMATED

Number of parameters estimated in computing the cumulative distribution function.

UPPER_BOUND

Upper bound of the range of the distribution. If *Lower Bound* = *Upper Bound*, a range on the whole real line is used (the default). If the lower and upper endpoints are different, points outside of the range of these bounds are ignored. Distributions conditional on a range can be specified when *Lower_Bound* and *Upper_Bound* are used. If *Upper_Bound* is specified, then *Lower_Bound* also must be specified. By convention, *Lower_Bound* is excluded from the first interval, but *Upper_Bound* is included in the last interval.

USED_CUTPOINTS

Specifies the named variable into which the cutpoints to be used by *IMSL_CHISQTEST* are stored.

Discussion

The *IMSL_CHISQTEST* function performs a chi-squared goodness-of-fit test that a random sample of observations is distributed according to a specified theoretical cumulative distribution. The theoretical distribution, which may be continuous, discrete, or a mixture of discrete and continuous distributions, is specified by defined function *f*. Because you are allowed to give a range for the observations, a test that is conditional upon the specified range is performed.

Parameter *n_categories* gives the number of intervals into which the observations are to be divided. By default, equi-probable intervals are computed by IMSL_CHISQTEST, but intervals that are not equi-probable can be specified (through the use of keyword *Cutpoints*).

Regardless of the method used to obtain the cutpoints, the intervals are such that the lower endpoint is not included in the interval, while the upper endpoint is always included. If the cumulative distribution function has discrete elements, then user-provided cutpoints should always be used since IMSL_CHISQTEST cannot determine the discrete elements in discrete distributions.

By default, the lower and upper endpoints of the first and last intervals are *-infinity* and *+infinity*. The endpoints can be specified by using the keywords *Lower_Bound* and *Upper_Bound*.

A tally of counts is maintained for the observations in *x* as follows:

- If the cutpoints are specified, the tally is made in the interval to which x_i belongs using the endpoints specified.
- If the cutpoints are determined by IMSL_CHISQTEST, then the cumulative probability at x_i , $F(x_i)$, is computed by the function *f*.

The tally for x_i is made in interval number:

$$\lfloor mF(x_i) + 1 \rfloor$$

where $m = n\_categories$ and:

$$\lfloor \cdot \rfloor$$

is the function that takes the greatest integer that is no larger than the parameter of the function. Thus, if the computer time required to calculate the cumulative distribution function is large, user-specified cutpoints may be preferred in order to reduce the total computing time.

If the expected count in any cell is less than 1, then a rule of thumb is that the chi-squared approximation may be suspect. A warning message to this effect is issued in this case, as well as when an expected value is less than 5.

Programming Notes

You must supply a function *f* with calling sequence $F(y)$ that returns the value of the cumulative distribution function at any point *y* in the (optionally) specified range.

Many of the cumulative distribution functions in this reference manual can be used for *f*. It is, however, necessary to write a user-defined IDL Advanced Math and Stats function that calls the CDF, and then pass the name of this user-defined function for *f*.

Example

This example illustrates the use of `IMSL_CHISQTEST` on a randomly generated sample from the normal distribution. One-thousand randomly generated observations are tallied into 10 equi-probable intervals. In this example, the null hypothesis is not rejected.

```
.RUN
; Define the hypothesized, cumulative distribution function.
FUNCTION user_cdf, k
    RETURN, IMSL_NORMALCDF(k)
END

IMSL_RANDOMOPT, Set = 123457
x = IMSL_RANDOM(1000, /Normal)
; Generate normal deviates.
p_value = IMSL_CHISQTEST('user_cdf', 10, x)
; Perform chi-squared test.
PM, p_value

; Output the results.
0.154603
```

Errors

Warning Errors

`STAT_EXPECTED_VAL_LESS_THAN_1`—An expected value is less than 1.

`STAT_EXPECTED_VAL_LESS_THAN_5`—An expected value is less than 5.

Fatal Errors

`STAT_ALL_OBSERVATIONS_MISSING`—All observations contain missing values.

`STAT_INCORRECT_CDF_1`—Function f is not a cumulative distribution function. The value at the lower bound must be nonnegative, and the value at the upper bound must not be greater than 1.

`STAT_INCORRECT_CDF_2`—Function f is not a cumulative distribution function. The probability of the range of the distribution is not positive.

`STAT_INCORRECT_CDF_3`—Function f is not a cumulative distribution function. Its evaluation at an element in x is inconsistent with either the evaluation at the lower or upper bound.

STAT_INCORRECT_CDF_4—Function f is not a cumulative distribution function. Its evaluation at a cutpoint is inconsistent with either the evaluation at the lower or upper bound.

STAT_INCORRECT_CDF_5—An error has occurred when inverting the cumulative distribution function. This function must be continuous and defined over the whole real line.

Version History

6.4	Introduced

IMSL_NORMALITY

The IMSL_NORMALITY function performs a test for normality.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_NORMALITY(x [, CHISQ=variable] [, DF=variable] [, /DOUBLE]  
[, LILLIEFORS=variable] [, NCAT=value] [, SHAPIRO_WILK=variable] )
```

Return Value

The p -value for the Shapiro-Wilk W test or the Lilliefors test for normality. The Shapiro-Wilk test is the default. If the Lilliefors test is used, probabilities less than 0.01 are reported as 0.01, and probabilities greater than 0.10 for the normal distribution are reported as 0.5; otherwise, an approximate probability is computed.

Arguments

x

One-dimensional array containing the observations.

Keywords

CHISQ

Specifies a variable into which the chi-square statistic is stored. Keywords *Ncat*, *Df*, and *Chisq* must be used together and indicate that the chi-squared goodness-of-fit test is to be performed.

DF

Specifies a variable into which the degrees of freedom for the test are stored. Keywords *Ncat*, *Df* and *Chisq* must be used together and indicate that the chi-squared goodness-of-fit test is to be performed.

DOUBLE

If present and nonzero, double precision is used.

LILLIEFORS

Named variable into which the maximum absolute difference between the empirical and the theoretical distributions is stored. If *Lilliefors* is present, then Lilliefors test is performed.

NCAT

An integer specifying number of cells into which the observations are to be tallied. Keywords *Ncat*, *Df*, and *Chisq* must be used together and indicate that the chi-squared goodness-of-fit test is to be performed.

SHAPIRO_WILK

Named variable into which the Shapiro-Wilk *W* statistic is stored. If *Shapiro_Wilk* is present, then the Shapiro-Wilk *W* test is performed. Default: Shapiro-Wilk *W* test is performed

Discussion

Three methods are provided for testing normality: the Chi-Squared test, the Shapiro-Wilk *W* test, and the Lilliefors test.

Chi-Squared Test

This function computes the chi-squared statistic, its *p*-value, and the degrees of freedom of the test. Keyword *Ncat* finds the number of intervals into which the observations are to be divided. The intervals are equi-probable except for the first and last interval which are infinite in length. If more flexibility is desired for the specification of intervals, the same test can be performed with a call to `IMSL_CHISQTEST` using the optional arguments described for that function.

Shapiro-Wilk *W* Test

D'Agostino and Stevens (1986, p. 406) refer to the Shapiro-Wilk *W* test as the best omnibus tests of normality. The function is based on the approximations and code

given by Royston (1982a, b, c). It can be used in samples as large as 2,000 or as small as 3. In the Shapiro and Wilk test, W is given by:

$$W = \left(\sum a_i x_{(i)} \right)^2 / \left(\sum (x_i - \bar{x})^2 \right)$$

where $x_{(i)}$ is the i -th smallest order statistic and:

$$\bar{x}$$

is the sample mean. Royston (1982) gives approximations and tabled values that can be used to compute the coefficients a_i , $i = 1, \dots, n$, and obtains the significance level of the W statistic.

Lilliefors Test

This function computes Lilliefors test and its p -values for a normal distribution in which both the mean and variance are estimated. The one-sample, two-sided Kolmogorov-Smirnov statistic D is first computed. The p -values are then computed using an analytic approximation given by Dallal and Wilkinson (1986). Because Dallal and Wilkinson give approximations in the range (0.01, 0.10) if the computed probability of a greater D is less than 0.01, a note is issued and the p -value is set to 0.50. Note that because parameters are estimated, p -values in Lilliefors test are not the same as in the Kolmogorov-Smirnov Test.

Observations should not be tied. If tied observations are found, an informational message is printed. A general reference for the Lilliefors test is Conover (1980). The original reference for the test for normality is Lilliefors (1967).

Examples

Example 1

The following example is taken from Conover (1980, pp. 195, 364). The data consists of 50 two-digit numbers taken from a telephone book. The W test fails to reject the null hypothesis of normality at the .05 level of significance.

```
x = [23, 36, 54, 61, 73, 23, 37, 54, 61, 73, $
      24, 40, 56, 62, 74, 27, 42, 57, 63, 75, $
      29, 43, 57, 64, 77, 31, 43, 58, 65, 81, $
      32, 44, 58, 66, 87, 33, 45, 58, 68, 89, $
      33, 48, 58, 68, 93, 35, 48, 59, 70, 97]
p = IMSL_NORMALITY(x)
PRINT, 'P-Value = ', p

P-Value =          0.230858
```


Example 2

The following example uses the same data as the previous example. Here, the Shapiro-Wilk W statistic is output.

```
p = IMSL_NORMALITY(x, SHAPIRO_WILK = sw)
PRINT, 'p-Value          = ', p
PRINT, 'Shapiro Wilk W Statistic = ', sw

p-Value          =          0.230858
Shapiro Wilk W Statistic =      0.964217
```

Errors

Warning Errors

STAT_ALL_OBS_TIED—All observations in x are tied.

Fatal Errors

STAT_NEED_AT_LEAST_5—All but # elements of x are missing. At least five nonmissing observations are necessary to continue.

STAT_NEG_IN_EXPONENTIAL—In testing the exponential distribution, an invalid element in x is found ($x[\] = \#$). Negative values are not possible in exponential distributions.

STAT_NO_VARIATION_INPUT—There is no variation in the input data. All nonmissing observations are tied.

Version History

6.4	Introduced

IMSL_KOLMOGOROV1

The IMSL_KOLMOGOROV1 function performs a Kolmogorov-Smirnov one-sample test for continuous distributions.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_KOLMOGOROV1(f, x [, DIFFERENCES=variable] [, /DOUBLE]
[, NMISsing=variable])
```

Return Value

One-dimensional array of length 3 containing Z , p_1 , and p_2 .

Arguments

f

A scalar string specifying a user-supplied function to compute the cumulative distribution function (CDF) at a given value. The function f must accept a scalar input argument and return the computed value at that point.

x

A one-dimensional array containing the observations.

Keywords

DIFFERENCES

Named variable into which an array containing D_n , D_n^+ , D_n^- is stored.

DOUBLE

If present and nonzero, double precision is used.

NMISSING

Named variable into which the number of missing values is stored.

Discussion

The `IMSL_KOLMOGOROV1` function performs a Kolmogorov-Smirnov goodness-of-fit test in one sample. The hypotheses tested follow:

- $H_0 : F(x) = F^*(x)$ $H_1 : F(x) \neq F^*(x)$
- $H_0 : F(x) \geq F^*(x)$ $H_1 : F(x) < F^*(x)$
- $H_0 : F(x) \leq F^*(x)$ $H_1 : F(x) > F^*(x)$

where F is the cumulative distribution function (CDF) of the random variable, and the theoretical CDF, F^* , is specified via the supplied function f . Let $n = N\_ELEMENTS(x) - Nmissing$. The test statistics for both one-sided alternatives:

$$D_n^+ = Differences(1)$$

and:

$$D_n^- = Differences(2)$$

and the two-sided ($D_n = Differences(0)$) alternative are computed as well as an asymptotic z-score ($Result(0)$) and p -values associated with the one-sided ($Result(1)$) and two-sided ($Result(2)$) hypotheses. For $n > 80$, asymptotic p -values are used (see Gibbons 1971). For $n \leq 80$, exact one-sided p -values are computed according to a method given by Conover (1980, page 350). An approximate two-sided test p -value is obtained as twice the one-sided p -value. The approximation is very close for one-sided p -values less than 0.10 and becomes very bad as the one-sided p -values get larger.

Programming Notes

1. The theoretical CDF is assumed to be continuous. If the CDF is not continuous, the statistics:

$$D_n^*$$

will not be computed correctly.

2. Estimation of parameters in the theoretical CDF from the sample data will tend to make the p -values associated with the test statistics too liberal. The empirical CDF will tend to be closer to the theoretical CDF than it should be.

- 3. No attempt is made to check that all points in the sample are in the support of the theoretical CDF. If all sample points are not in the support of the CDF, the null hypothesis must be rejected.

Example

In this example, a random sample of size 100 is generated via routine IMSL_RANDOM for the uniform (0, 1) distribution. We want to test the null hypothesis that the CDF is the standard normal distribution with a mean of 0.5 and a variance equal to the uniform (0, 1) variance (1/12).

```
.RUN
FUNCTION l_Cdf, x
  mean = 0.5
  std = 0.2886751
  z = (x - mean)/std
  val = IMSL_NORMALCDF(z)
  RETURN, val
END

IMSL_RANDOMOPT, set = 123457
x = IMSL_RANDOM(100, /UNIFORM)
stats = IMSL_KOLMOGOROV1('l_cdf', x, DIFFERENCES = d, $
  NMISSING = nm)
PRINT, 'D =', d(0)
PRINT, 'D+ =', d(1)
PRINT, 'D- =', d(2)
PRINT, 'Z =', stats(0)
PRINT, 'Prob greater D one sided =', stats(1)

D = 0.147083
D+ = 0.0809559
D- = 0.147083
Z = 1.47083
Prob greater D one sided = 0.0132111
```

Version History

6.4	Introduced

IMSL_KOLMOGOROV2

The IMSL_KOLMOGOROV2 function performs a Kolmogorov-Smirnov two-sample test.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = KOLMORGOROV2(x, y [, DIFFERENCES=variable] [, /DOUBLE]
[, NMISSINGX=variable] [, NMISSINGY=variable] )
```

Return Value

One-dimensional array of length 3 containing Z , p_1 , and p_2 .

Arguments

x

One-dimensional array containing the observations from sample one.

y

One-dimensional array containing the observations from sample two.

Keywords

DIFFERENCES

Named variable into which a one-dimensional array containing D_n , D_n^+ , D_n^- is stored.

DOUBLE

If present and nonzero, double precision is used.

NMISSINGX

Named variable into which the number of missing values in the x sample is stored.

NMISSINGY

Named variable into which the number of missing values in the y sample is stored.

Discussion

The IMSL_KOLMOGOROV2 function computes Kolmogorov-Smirnov two-sample test statistics for testing that two continuous cumulative distribution functions (CDF's) are identical based upon two random samples. One- or two-sided alternatives are allowed. If $n\_observations\_x = N\_ELEMENTS(x)$ and $n\_observations\_y = N\_ELEMENTS(y)$, then the exact p -values are computed for the two-sided test when $n\_observations\_x * n\_observations\_y$ is less than 104.

Let $F_n(x)$ denote the empirical CDF in the X sample, let $G_m(y)$ denote the empirical CDF in the Y sample, where $n = n\_observations\_x - Nmissingx$ and $m = n\_observations\_y - Nmissingy$, and let the corresponding population distribution functions be denoted by $F(x)$ and $G(y)$, respectively. Then, the hypotheses tested by IMSL_KOLMOGOROV2 are as follows:

- $H_0: F(x) = G(x) \quad H_1: F(x) \neq G(x)$
- $H_0: F(x) \leq G(x) \quad H_1: F(x) > G(x)$
- $H_0: F(x) \geq G(x) \quad H_1: F(x) < G(x)$

The test statistics are given as follows:

$$D_{mn} = \max(D_{mn}^+, D_{mn}^-) \quad (\text{Differences}(0))$$

$$D_{mn}^+ = \max_x (F_n(x) - G_m(x)) \quad (\text{Differences}(1))$$

$$D_{mn}^- = \max_x (G_m(x) - F_n(x)) \quad (\text{Differences}(2))$$

Asymptotically, the distribution of the statistic

$$Z = D_{mn} \sqrt{(m+n) / (m*n)}$$

(returned in *Result* (0)) converges to a distribution given by Smirnov (1939).

Exact probabilities for the two-sided test are computed when $m * n$ is less than or equal to 10^4 , according to an algorithm given by Kim and Jennrich (1973;). When $m * n$ is greater than 10^4 , the very good approximations given by Kim and Jennrich are used to obtain the two-sided p -values. The one-sided probability is taken as one half

the two-sided probability. This is a very good approximation when the p -value is small (say, less than 0.10) and not very good for large p -values.

Example

The following example illustrates the IMSL_KOLMOGOROV2 routine with two randomly generated samples from a uniform(0,1) distribution. Since the two theoretical distributions are identical, we would not expect to reject the null hypothesis.

```
IMSL_RANDOMOPT, set = 123457
x = IMSL_RANDOM(100, /Uniform)
y = IMSL_RANDOM(60, /Uniform)
stats = IMSL_KOLMOGOROV2(x, y, DIFFERENCES = d, $
    NMISSINGX = nmX, NMISSINGY = nmy)
PRINT, 'D = ', d(0)
PRINT, 'D+ = ', d(1)
PRINT, 'D- = ', d(2)
PRINT, 'Z = ', stats(0)
PRINT, 'Prob greater D one sided =', stats(1)
PRINT, 'Prob greater D two sided =', stats(2)
PRINT, 'Missing X =', nmX
PRINT, 'Missing Y =', nmy

D =      0.180000
D+ =      0.180000
D- =      0.0100001
Z =      1.10227
Prob greater D one sided =      0.0720105
Prob greater D two sided =      0.144021
Missing X =      0
Missing Y =      0
```

Version History

6.4	Introduced

IMSL_MVAR_NORMALITY

The IMSL_MVAR_NORMALITY function computes Mardia's multivariate measures of skewness and kurtosis and tests for multivariate normality.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_MVAR_NORMALITY(x [, /DOUBLE] [, FREQUENCIES=array]
[, MEANS=variable] [, NMISsing=variable] [, R_MATRIX=variable]
[, SUM_FREQS=variable] [, SUM_WEIGHTS=variable] [, WEIGHTS=array])
```

Return Value

One-dimensional array of size 13 containing output statistics as shown in [Table 19-1](#).

I	<i>result</i> (I)
0	estimated skewness
1	expected skewness assuming a multivariate normal distribution
2	asymptotic chi-squared statistic assuming a multivariate normal distribution
3	probability of a greater chi-squared
4	Mardia and Foster's standard normal score for skewness
5	estimated kurtosis
6	expected kurtosis assuming a multivariate normal distribution
7	asymptotic standard error of the estimated kurtosis
8	standard normal score obtained from <i>Result</i> (5) through <i>Result</i> (7)
9	<i>p</i> -value corresponding to <i>Result</i> (8)

Table 19-1: Output Statistics

I	<i>result (I)</i>
10	Mardia and Foster's standard normal score for kurtosis
11	Mardia's S_W statistic based upon <i>Result(4)</i> and <i>Result(10)</i>
12	<i>p</i> -value for <i>Result(11)</i>

Table 19-1: Output Statistics (Continued)

Arguments

x

2D array containing data in which $N\_ELEMENTS(x(*,0))$ is the number of observations (numbers of rows of data) in x and $N\_ELEMENTS(x(0,*))$ is the dimensionality of the multivariate space for which the skewness and kurtosis are to be computed (number of variables in x).

Keywords

DOUBLE

If present and nonzero, double precision is used.

FREQUENCIES

One-dimensional array containing the frequencies. *Frequencies* must be an integer value. Default assumes all *Frequencies* equal one.

MEANS

Named variable into which a one-dimensional array of length $N\_ELEMENTS(x(0,*))$ containing the sample means is stored.

NMISSING

Named variable into which the number of rows of data in x containing any missing values (NaN) is stored.

R_MATRIX

Named variable into which an upper triangular array containing the Cholesky $R^T R$ factorization of the covariance matrix is stored.

SUM_FREQS

Named variable into which the sum of the frequencies of all observations used in the computations is stored.

SUM_WEIGHTS

Named variable into which the sum of the weights times the frequencies for all observations used in the computations is stored.

WEIGHTS

One-dimensional array containing the weights. *Weights* must be non-negative. Default assumes all *Weights* equal one.

Discussion

The IMSL_MVAR_NORMALITY function computes Mardia's (1970) measures $b_{1,p}$ and $b_{2,p}$ of multivariate skewness and kurtosis, respectfully, for $p = \text{N\_ELEMENTS}(x(0,*))$. These measures are then used in computing tests for multivariate normality. Three test statistics, one based upon $b_{1,p}$ alone, one based upon $b_{2,p}$ alone, and an omnibus test statistic formed by combining normal scores obtained from $b_{1,p}$ and $b_{2,p}$ are computed. On the order of np^3 , operations are required in computing $b_{1,p}$ when the method of Isogai (1983) is used, where $n = \text{N\_ELEMENTS}(x(*,0))$. On the order of np^2 , operations are required in computing $b_{2,p}$.

Let:

$$d_{ij} = \sqrt{w_i w_j} (x_i - \bar{x})^T S^{-1} (x_j - \bar{x})$$

where:

$$S = \frac{\sum_{i=1}^n w_i f_i (x_i - \bar{x})(x_i - \bar{x})^T}{\sum_{i=1}^n f_i}$$

$$\bar{x} = \frac{1}{\sum_{i=1}^n w_i f_i} \sum_{i=1}^n w_i f_i x_i$$

f_i is the frequency of the i -th observation, and w_i is the weight for this observation. (Weights w_i are defined such that x_i is distributed according to a multivariate normal,

$N(\mu, \Sigma/w_i)$ distribution, where Σ is the covariance matrix.) Mardia's multivariate skewness statistic is defined as:

$$b_{1,p} = \frac{1}{n^2} \sum_{i=1}^n \sum_{j=1}^n f_i f_j d_{ij}^3$$

while Mardia's kurtosis is given as:

$$b_{2,p} = \frac{1}{n} \sum_{i=1}^n f_i d_{ii}^2$$

Both measures are invariant under the affine (matrix) transformation $AX + D$, and reduce to univariate measures when $p = N\_ELEMENTS(x(0,*)) = 1$. Using formulas given in Mardia and Foster (1983), the approximate expected value, asymptotic standard error, and asymptotic p -value for $b_{2,p}$, and the approximate expected value, an asymptotic chi-squared statistic, and p -value for the $b_{1,p}$ statistic are computed. These statistics are all computed under the null hypothesis of a multivariate normal distribution. In addition, standard normal scores $W_1(b_{1,p})$ and $W_2(b_{2,p})$ (different from but similar to the asymptotic normal and chi-squared statistics above) are computed. These scores are combined into an asymptotic chi-squared statistic with two degrees of freedom:

$$S_W = W_1^2(b_{1,p}) + W_2^2(b_{2,p})$$

This chi-squared statistic may be used to test for multivariate normality. A p -value for the chi-squared statistic is also computed.

Example

In the following example, 150 observations from a 5 dimensional standard normal distribution are generated via routine `IMSL_RANDOM` (Chapter 12, *Random Number Generation*). The skewness and kurtosis statistics are then computed for these observations.

```
m = 150
n = 5
IMSL_RANDOMOPT, set = 123457
x = FLTARR(n, m)
x(*) = IMSL_RANDOM(m*n, /Normal)
x = TRANSPOSE(x)
stats = IMSL_MVAR_NORMALITY(x, Sum_Weights = sw, Sum_Freq = sf, $
    Means = means, R_Matrix = r_mat)
PRINT, 'Sum of Frequencies =', sf, FORMAT = '(A25, I4)'
PRINT, 'Sum of the weights =', sw, FORMAT = '(A25, F8.3)'
FOR i = 0, 12 DO PM, i, stats(i), FORMAT = '(I5, F10.2)'
```

Sum of Frequencies = 150
Sum of the weights = 150.000
0 0.73
1 1.36
2 18.62
3 0.99
4 -2.37
5 32.67
6 34.54
7 1.27
8 -1.48
9 0.14
10 1.62
11 8.24
12 0.02

Version History

6.4	Introduced

IMSL_RANDOMNESS_TEST

The IMSL_RANDOMNESS_TEST function performs a test for randomness.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_RANDOMNESS_TEST(x, n_run [, COVARIANCES=variable]
    [, DCUBE_COUNTS=variable] [, /DOUBLE]
    [, DSQUARE_COUNTS=variable] [, EXPECT=variable]
    [, PAIRS_COUNTS=variable] [, PAIRS_LAG=value]
    [, RUNS_COUNTS=variable] [, RUNS_EXPECT=variable])
```

Return Value

The probability of a larger chi-squared statistic for testing the null hypothesis of a uniform distribution.

Arguments

n_run

Length of longest run for which tabulation is desired. For keywords *Pairs_Counts*, *Dsquare_Counts*, and *Dcube_Counts*, *n_run* stands for the number of equiprobable cells into which the statistics are to be tabulated.

x

One-dimensional array containing the data.

Keywords

COVARIANCES

Named variable into which an array of size $N\_ELEMENTS(x)$ by $N\_ELEMENTS(x)$ containing the variances and covariances of the counts is stored. Keywords *Runs_Counts* and *Covariances* must be used together.

Exactly one of the options listed in [Table 19-2](#) is used to specify which test is to be performed.

Keyword	Test to be Performed
<i>Runs_Counts</i> with <i>Covariances</i>	Runs Test
<i>Pairs_Counts</i> with <i>Pairs_Lag</i>	Pairs Test
<i>Dsquare_Counts</i>	d^2 Test
<i>Dcube_Counts</i>	Triplets Test

Table 19-2: Output Keywords

DCUBE_COUNTS

Named variable into which an array of length n_run by n_run by n_run containing the tabulations for the triplets test is stored. Keywords *Runs_Counts*, *Pairs_Counts*, *Dsquare_Counts*, and *Dcube_Counts* can not be used together.

Chisq—Named variable into which the Chi-squared statistic for testing the null hypothesis of a uniform distribution is stored.

Df—Named variable into which the degrees of freedom for chi-squared is stored.

Exactly one of the options listed in [Table 19-3](#) is used to specify which test is to be performed.

Keyword	Test to be Performed
<i>Runs_Counts</i> with <i>Covariances</i>	Runs Test
<i>Pairs_Counts</i> with <i>Pairs_Lag</i>	Pairs Test
<i>Dsquare_Counts</i>	d^2 Test
<i>Dcube_Counts</i>	Triplets Test

Table 19-3: Output Keywords

DOUBLE

If present and nonzero, double precision is used.

DSQUARE_COUNTS

Named variable into which an array of length n_run containing the tabulations for the d^2 test is stored. Keywords *Dsquare_Counts*, *Runs_Counts*, *Pairs_Counts*, and *Dcube_Counts* can not be used together

Exactly one of the options listed in [Table 19-4](#) is used to specify which test is to be performed.

Keyword	Test to be Performed
<i>Runs_Counts</i> with <i>Covariances</i>	Runs Test
<i>Pairs_Counts</i> with <i>Pairs_Lag</i>	Pairs Test
<i>Dsquare_Counts</i>	d^2 Test
<i>Dcube_Counts</i>	Triplets Test

Table 19-4: Output Keywords

EXPECT

Named variable into which the expected number of counts for each cell is stored.

Note

This keyword is optional only if one of the keywords *Pairs_Counts*, *Dsquare_Counts*, or *Dcube_Count* is used. Keywords *Runs_Counts* and *Expect* can not be used together.

PAIRS_COUNTS

Named variable into which an array of size n_run by n_run containing the count of the number of pairs in each cell is stored. The lag to be used in computing the pairs statistic is stored in *Pairs_Lag*. Pairs ($X(i)$, $X(i + Pairs\_Lag)$) for $i = 0, \dots, N - Pairs\_Lag - 1$ are tabulated, where N is the total sample size. Keywords *Pairs_Counts* and *Pairs_Lag* must be used together. Keywords *Pairs_Counts*, *Runs_Counts*, *Dsquare_Counts*, and *Dcube_Counts* can not be used together.

Exactly one of the options listed in [Table 19-5](#) is used to specify which test is to be performed.

Keyword	Test to be Performed
<i>Runs_Counts</i> with <i>Covariances</i>	Runs Test
<i>Pairs_Counts</i> with <i>Pairs_Lag</i>	Pairs Test
<i>Dsquare_Counts</i>	d^2 Test
<i>Dcube_Counts</i>	Triplets Test

Table 19-5: Output Keywords

PAIRS_LAG

The lag to be used in computing the pairs statistic. Keywords *Pairs_Lag* and *Pairs_Counts* must be used together.

RUNS_COUNTS

Named variable into which an array of size `N_ELEMENTS(x)` containing the counts of the number of runs up each length is stored. The Runs Test is the default test, however, to return the counts and covariances, the *Runs_Counts* keyword must be used. Keywords *Runs_Counts* and *Covariances* must be used together. Keywords *Runs_Counts*, *Pairs_Counts*, *Dsquare_Counts*, and *Dcube_Counts* can not be used together.

Exactly one of the options listed in [Table 19-6](#) is used to specify which test is to be performed.

Keyword	Test to be Performed
<i>Runs_Counts</i> with <i>Covariances</i>	Runs Test
<i>Pairs_Counts</i> with <i>Pairs_Lag</i>	Pairs Test
<i>Dsquare_Counts</i>	d^2 Test
<i>Dcube_Counts</i>	Triplets Test

Table 19-6: Output Keywords

RUNS_EXPECT

Named variable into which an array of length *n_run* containing the expected number of runs of each length is expected is stored.

Note

This keyword is optional if *Runs_Counts* is used.

Discussion

Runs Up Test

The IMSL_RANDOMNESS_TEST function performs one of four different tests for randomness. Input keyword *Runs_Counts* computes statistics for the runs up test. Runs tests are used to test for cyclical trend in sequences of random numbers. If the runs down test is desired, each observation should first be multiplied by -1 to change its sign, and *Runs_Counts* used with the modified vector of observations.

Runs_Counts first tallies the number of runs up (increasing sequences) of each desired length. For $i = 1, \dots, r - 1$, where $r = n\_run$, *Runs_Counts*(*i*) contains the number of runs of length *i*. *Runs_Counts*(*n_run*) contains the number of runs of length *n_run* or greater. As an example of how runs are counted, the sequence (1, 2, 3, 1) contains 1 run up of length 3, and one run up of length 1.

After tallying the number of runs up of each length, *Runs_Counts* computes the expected values and the covariances of the counts according to methods given by Knuth (1981, pages 65(67)). Let *R* denote a vector of length *n_run* containing the number of runs of each length so that the *i*-th element of *R*, r_i , contains the count of the runs of length *i*. Let Σ_R denote the covariance matrix of *R* under the null hypothesis of randomness, and let μ_R denote the vector of expected values for *R* under this null hypothesis, then an approximate chi-squared statistic with *n_run* degrees of freedom is given as:

$$\chi^2 = (R - \mu_R)^T \Sigma_R^{-1} (R - \mu_R)$$

In general, the larger the value of each element of μ_R , the better the chi-squared approximation.

Pairs Test

Pairs_Counts computes the pairs test (or the Good's serial test) on a hypothesized sequence of uniform (0,1) pseudorandom numbers. The test proceeds as follows.

Subsequent pairs ($X(i)$, $X(i + Pairs\_Lag)$) are tallied into a $k \times k$ matrix, where $k = n\_run$. In this tally, element (j, m) of the matrix is incremented, where:

$$j = \lfloor kX(i) \rfloor + 1$$

$$m = \lfloor kX(i + l) \rfloor + 1$$

where $l = Pairs\_Lag$, and the notation $\lfloor \cdot \rfloor$ represents the greatest integer function, $\lfloor Y \rfloor$ is the greatest integer less than or equal to Y , where Y is a real number. If $l = 1$, then $i = 1, 3, 5, \dots, n - 1$. If $l > 1$, then $i = 1, 2, 3, \dots, n - l$, where n is the total number of pseudorandom numbers input on the current usage of *Pairs_Counts* (i.e., $n = N\_ELEMENTS(x)$).

Given the tally matrix in *Pairs_Counts*, chi-squared is computed as:

$$\chi^2 = \sum_{i,j=0}^{k-1} \frac{(o_{ij} - e)^2}{e}$$

where $e = \sum o_{ij}/k^2$, and o_{ij} is the observed count in cell (i, j) ($o_{ij} = Pairs\_Counts(i, j)$).

Because pair statistics for the trailing observations are not tallied on any call, You should use *Pairs_Counts* with $N\_ELEMENTS(x)$ as large as possible. For *Pairs_Lag* < 20 and $N\_ELEMENTS(x) = 2000$, little power is lost.

d² Test

Dsquare_Counts computes the d^2 test for succeeding quadruples of hypothesized pseudorandom uniform (0, 1) deviates. The d^2 test is performed as follows. Let X_1 , X_2 , X_3 , and X_4 denote four pseudorandom uniform deviates, and consider:

$$D^2 = (X_3 - X_1)^2 + (X_4 - X_2)^2$$

The probability distribution of D^2 is given as:

$$\Pr(D^2 \leq d^2) = d^2\pi - \frac{8d^3}{3} + \frac{d^4}{2}$$

when $D^2 \leq 1$, where π denotes the value of pi. If $D^2 > 1$, this probability is given as:

$$\Pr(D^2 \leq d^2) = \frac{1}{3} + (\pi - 2)d^2 + 4\sqrt{d^2 - 1} + 8\frac{(d^2 - 1)^{\frac{3}{2}}}{3} - \frac{d^4}{2} - 4d^2 \operatorname{atan}\left(\frac{\sqrt{1 - \frac{1}{d^2}}}{\frac{1}{d}}\right)$$

See Gruenberger and Mark (1951) for a derivation of this distribution.

For each succeeding set of 4 pseudorandom uniform numbers input in x , d^2 and the cumulative probability of d^2 ($\Pr(D^2 \leq d^2)$) are computed. The resulting probability is tallied into one of $k = n\_run$ equally spaced intervals.

Let n denote the number of sets of four random numbers input (n = the total number of observations/4). Then, under the null hypothesis that the numbers input are random uniform (0, 1) numbers, the expected value for each element in *Dsquare_Counts* is $e = n/k$. An approximate chi-squared statistic is computed as:

$$\chi^2 = \sum_{i=0}^{k-1} \frac{(o_i - e)^2}{e}$$

where $o_i = \text{Dsquare\_Counts}(i)$ is the observed count. Thus, χ^2 has $k - 1$ degrees of freedom, and the null hypothesis of pseudorandom uniform (0, 1) deviates is rejected if χ^2 is too large. As n increases, the chi-squared approximation becomes better. A useful generalization is that $e > 5$ yields a good chi-squared approximation.

Triplets Test

Dcube_Counts computes the triplets test on a sequence of hypothesized pseudorandom uniform(0, 1) deviates. The triplets test is computed as follows: Each set of three successive deviates, X_1 , X_2 , and X_3 , is tallied into one of m^3 equal sized cubes, where $m = n\_run$. Let $i = [mX_1] + 1$, $j = [mX_2] + 1$, and $k = [mX_3] + 1$. For the triplet (X_1, X_2, X_3) , *Dcube_Counts*(i, j, k) is incremented.

Under the null hypothesis of pseudorandom uniform(0, 1) deviates, the m^3 cells are equally probable and each has expected value $e = n/m^3$, where n is the number of triplets tallied. An approximate chi-squared statistic is computed as:

$$\chi^2 = \sum_{i,j,k=0}^{m-1} \frac{(o_{ijk} - e)^2}{e}$$

where $o_{ijk} = \text{Dcube\_Counts}(i, j, k)$.

The computed chi-squared has $m^3 - 1$ degrees of freedom, and the null hypothesis of pseudorandom uniform (0, 1) deviates is rejected if χ^2 is too large.

Examples

Example 1

The following example illustrates the use of the runs test on 10^4 pseudo-random uniform deviates. In the example, 2000 deviates are generated for each use of

Runs_Counts. Since the probability of a larger chi-squared statistic is 0.1872, there is no strong evidence to support rejection of this null hypothesis of randomness.

```
.RUN
PRO print_results, n_run, num, rc, re, cov, chisq, df, p
  PRINT, '          runs_count'
  PRINT, num + 1, FORMAT = '(6I5)'
  PRINT, rc, FORMAT = '(6I5)'
  PRINT
  PRINT, '          runs_expect'
  PRINT, num + 1, FORMAT = '(6I7)'
  PRINT, re, FORMAT = '(6F7.1)'
  PRINT
  PRINT, '          covariances'
  PRINT, num + 1, FORMAT = '(7X, 6I8)'
  FOR i = 0, n_run - 1 DO $
    PRINT, num(i) + 1, cov(i, *), FORMAT = '(I8, 6F8.1)'
  PRINT
  PRINT, 'chisq =', chisq
  PRINT, 'df    =', df
  PRINT, 'pvalue =', p
END

nran = 10000
n_run = 6
num = INDGEN(n_run)
IMSL_RANDOMOPT, set = 123457
x = IMSL_RANDOM(nran, /Uniform)
p = IMSL_RANDOMNESS_TEST(x, n_run, Runs_Counts = rc, $
  Covariances = cov, Chisq = chisq, Df = df, Runs_Expect = re)
print_results, n_run, num, rc, re, cov, chisq, df, p
```

runs_count					
1	2	3	4	5	6
1709	2046	953	260	55	4

runs_expect					
1	2	3	4	5	6
1667.3	2083.4	916.5	263.8	57.5	11.9

covariances						
	1	2	3	4	5	6
1	1278.2	-194.6	-148.9	-71.6	-22.9	-6.7
2	-194.6	1410.1	-490.6	-197.2	-55.2	-14.4
3	-148.9	-490.6	601.4	-117.4	-31.2	-7.8
4	-71.6	-197.2	-117.4	222.1	-10.8	-2.6
5	-22.9	-55.2	-31.2	-10.8	54.8	-0.6
6	-6.7	-14.4	-7.8	-2.6	-0.6	11.7

```

chisq  =      8.76515
df     =      6.00000
pvalue =      0.187223

```

Example 2

The following example illustrates the calculations of the *Pairs_Counts* statistics when a random sample of size 10^4 is used and the *Pairs_Lag* is 1. The results are not significant.

```

.RUN
PRO print_results, n_run, num, pc, expect, chisq, df, p
  PRINT, '                                pairs_count'
  PRINT, num + 1, FORMAT = '(5X, 10I5)'
  FOR i = 0, n_run - 1 DO $
    PRINT, num(i) + 1, pc(i, *), FORMAT = '(I5, 10I5)'
  PRINT
  PRINT, 'expect  =', expect
  PRINT, 'chisq   =', chisq
  PRINT, 'df      =', df
  PRINT, 'pvalue  =', p
END

nran = 10000
n_run = 10
num = INDGEN(n_run)
lag = 5
IMSL_RANDOMOPT, set = 123467
x = IMSL_RANDOM(nran, /Uniform)
p = IMSL_RANDOMNESS_TEST(x, n_run, Pairs_Counts = pc, $
  Pairs_Lag = lag, Chisq = chisq, $
  Df = df, Expect = expect)
print_results, n_run, num, pc, expect, chisq, df, p

```

	pairs_count									
	1	2	3	4	5	6	7	8	9	10
1	112	82	95	118	103	103	113	84	90	74
2	104	106	109	108	101	98	102	92	109	88
3	88	111	86	106	112	79	103	105	106	101
4	91	110	108	92	88	108	113	93	105	114
5	104	105	103	104	101	94	96	87	93	104
6	98	104	103	104	79	89	92	104	92	100
7	103	91	97	101	116	83	118	118	106	99
8	105	105	111	91	93	82	100	104	110	89
9	92	102	82	101	94	128	102	110	125	98
10	79	99	103	98	104	101	93	93	98	105

```

expect  =      99.9500

```

```

chisq   =      104.860
df      =      99.0000
pvalue  =      0.324242

```

Example 3

In the following example, 2000 observations generated using the routine `IMSL_RANDOM` are input to `Dsquare_Counts` in one call. In the example, the null hypothesis of a uniform distribution is not rejected.

```

.RUN
PRO print_results, n_run, num, dc, expect, chisq, df, p
  PRINT, '          dsquare_counts'
  PRINT, num + 1, FORMAT = '(6I5)'
  PRINT, dc, FORMAT = '(6I5)'
  PRINT
  PRINT, 'expect   =', expect
  PRINT, 'chisq    =', chisq
  PRINT, 'df       =', df
  PRINT, 'pvalue   =', p
END

nran = 2000
n_run = 6
num = INDGEN(n_run)
IMSL_RANDOMOPT, set = 123457
x = IMSL_RANDOM(nran, /Uniform)
p = IMSL_RANDOMNESS_TEST(x, n_run, Chisq = chisq, Df = df, $
  Expect = expect, Dsquare_Counts = dc)
print_results, n_run, num, dc, expect, chisq, df, p

          dsquare_counts
      1      2      3      4      5      6
      87     84     78     76     92     83

expect   =      83.3333
chisq    =      2.05600
df       =      5.00000
pvalue   =      0.841343

```

Example 4

In the following example, 2001 deviates generated by the routine `IMSL_RANDOM` are input to `Dcube_Counts`, and tabulated in 27 equally sized cubes. In the example, the null hypothesis is not rejected.

```

.RUN
PRO print_results, n_run, num, dc, expect, chisq, df, p
  FOR j = 0, n_run - 1 DO BEGIN

```

```

        PRINT, ' dcube_counts'
        PRINT, num + 1, FORMAT = '(5X, 3I5)'
    FOR i = 0, n_run - 1 DO $
        PRINT, num(i) + 1, dc(j, i, *), FORMAT = '(I5, 3I5)'
        PRINT
    ENDFOR
    PRINT, 'expect =', expect
    PRINT, 'chisq =', chisq
    PRINT, 'df      =', df
    PRINT, 'pvalue =', p
END

nran = 2001
n_run = 3
num = INDGEN(n_run)
IMSL_RANDOMOPT, set = 123457
x = IMSL_RANDOM(nran, /Uniform)
p = IMSL_RANDOMNESS_TEST(x, n_run, Chisq = chisq, Df = df, $
    Expect = expect, Dcube_Counts = dc)
print_results, n_run, num, dc, expect, chisq, df, p

```

```

dcube_counts
  1      2      3
1  26    27    24
2  20    17    32
3  30    18    21

```

```

dcube_counts
  1      2      3
1  20    16    26
2  22    22    27
3  30    24    26

```

```

dcube_counts
  1      2      3
1  28    30    22
2  23    24    22
3  33    30    27

```

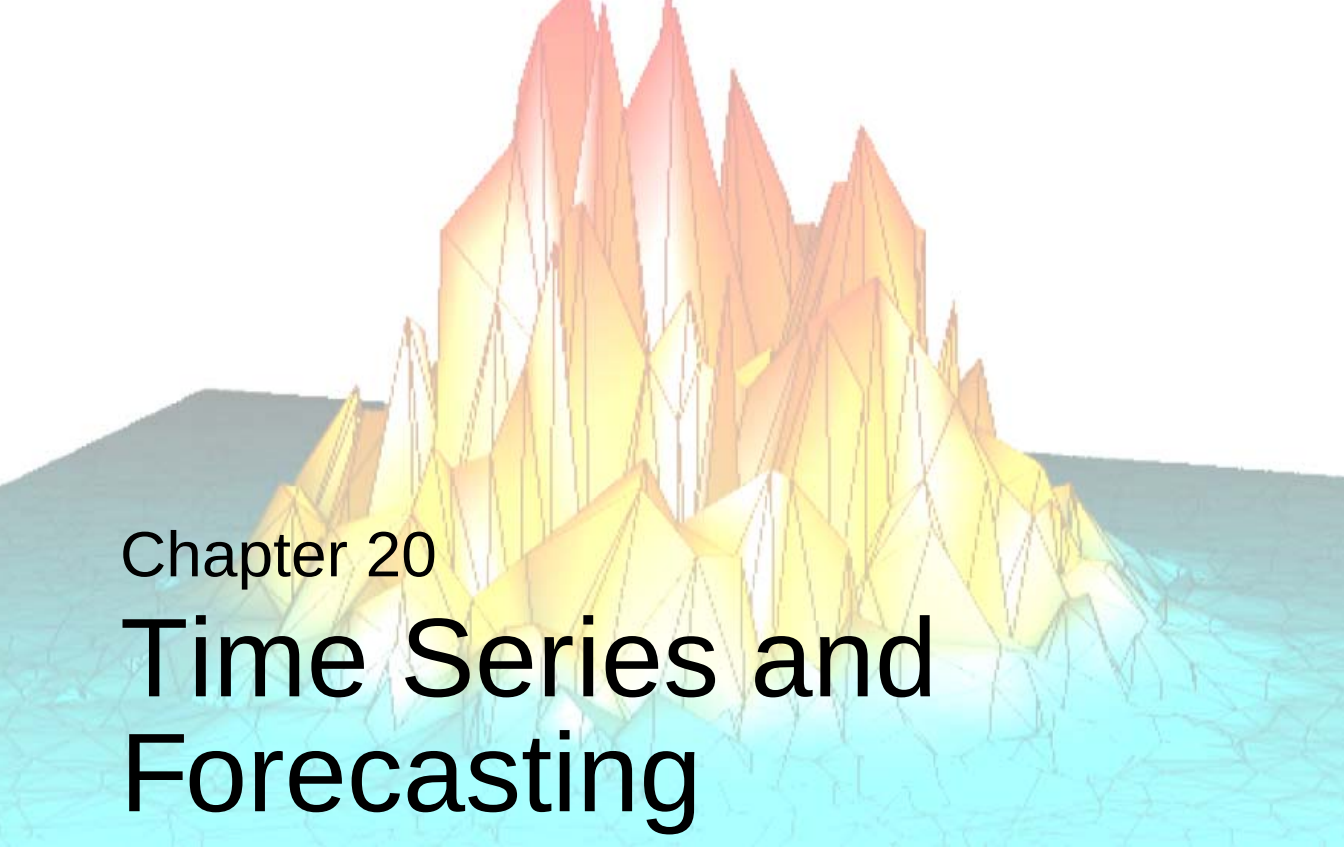
```

expect =      24.7037
chisq   =      21.7631
df      =      26.0000
pvalue  =      0.701585

```

Version History

6.4	Introduced



Chapter 20

Time Series and Forecasting

This section contains the following topics:

[Overview: Time Series and Forecasting](#) . . 910 [Time Series and Forecasting Routines](#) . . . 912

Overview: Time Series and Forecasting

The routines in this chapter assume the time series does not contain any missing observations. If missing values are present, they should be set to the special floating-point value Not a Number (NaN), and the routine will return an appropriate error message. To enable fitting of the model, the missing values must be replaced by appropriate estimates.

General Methodology

A major component of the model identification step concerns determining if a given time series is stationary. The sample correlation functions computed by routines [“IMSL_AUTOCORRELATION”](#) on page 940, and [“IMSL_PARTIAL_AC”](#) on page 945 may be used to diagnose the presence of non-stationarity in the data, as well as to indicate the type of transformation¹ require to induce stationarity. The family of power transformations provided by routine [“IMSL_BOXCOXTRANS”](#) on page 935 coupled with the ability to difference the transformed data using routine [“IMSL_DIFFERENCE”](#) on page 929 affords a convenient method of transforming a wide class of nonstationary time series to stationarity.

The “raw” data, transformed data, and sample correlation functions also provide insight into the nature of the underlying model. Typically, this information is displayed in graphical form via time series plots, plots of the lagged data, and various correlation function plots.

The observed time series may also be compared with time series generated from various theoretical models to help identify possible candidates for model fitting. The routine `IMSL_RANDOM_ARMA` may be used to generate a time series according to a specified autoregressive moving average model.

Time Domain Methodology

Once the data are transformed to stationarity, a tentative model in the time domain is often proposed and parameter estimation¹, diagnostic checking and forecasting are performed.

ARIMA Model (Autoregressive Integrated Moving Average)

A small, yet comprehensive, class of stationary time-series models consists of the nonseasonal `IMSL_ARMA` processes defined by:

$$\phi(B)(W_t - \mu) = \theta(B)A_t, \quad t \in Z$$

where $Z = \{\dots, -2, -1, 0, 1, 2, \dots\}$ denotes the set of integers, B is the backward shift operator defined by $B^k W_t = W_{t-k}$, μ is the mean of W_t , and the following equations are true:

$$\phi(B) = 1 - \phi_1 B - \phi_2 B^2 - \dots - \phi_p B^p, \quad p \geq 0$$

$$\theta(B) = 1 - \theta_1 B - \theta_2 B^2 - \dots - \theta_q B^q, \quad q \geq 0$$

The model is of order (p, q) and is referred to as an `IMSL_ARMA` (p, q) model.

An equivalent version of the `IMSL_ARMA` (p, q) model is given by:

$$\phi(B)W_t = \theta_0 + \theta(B)A_t, \quad t \in Z$$

where θ_0 is an overall constant defined by the following:

$$\theta_0 = \mu \left(1 - \sum_{i=1}^p \phi_i \right)$$

See Box and Jenkins (1976, pp. 92–93) for a discussion of the meaning and usefulness of the overall constant.

If the “raw” data, $\{Z_t\}$, are homogeneous and nonstationary, then differencing using the “[IMSL_DIFFERENCE](#)” on page 929 induces stationarity, and the model is called ARIMA (AutoRegressive Integrated Moving Average). Parameter estimation is performed on the stationary time series $W_t = \nabla^d Z_t$, where $\nabla^d = (1 - B)^d$ is the backward difference operator with period 1 and order d , $d > 0$.

Typically, the method of moments includes keyword *Moments* in a call to the “[IMSL_ARMA](#)” on page 913 for preliminary parameter estimates. These estimates can be used as initial values into the least-squares procedure by including keyword *Lsq* in a call to function `ARMA`. Other initial estimates provided can be used. The least-squares procedure can be used to compute conditional or unconditional least-squares estimates of the parameters, depending on the choice of the backcasting length.

Time Series and Forecasting Routines

IMSL_ARMA Models

IMSL_ARMA—Computes least-squares or method-of-moments estimates of parameters and optionally computes forecasts and their associated probability limits.

IMSL_DIFFERENCE—Performs differencing on a time series.

IMSL_BOXCOXTRANS—Perform a Box-Cox transformation.

IMSL_AUTOCORRELATION—Sample autocorrelation function.

IMSL_PARTIAL_AC—Sample partial autocorrelation function.

IMSL_LACK_OF_FIT—Lack-of-fit test based on the correlation function.

IMSL_GARCH—Compute estimates of the parameters of a GARCH(p,q) model.

IMSL_KALMAN—Performs Kalman filtering and evaluates the likelihood function for the statespace model.

IMSL_ARMA

The IMSL_ARMA function computes method-of-moments or least-squares estimates of parameters for a nonseasonal IMSL_ARMA model.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_ARMA(z, p, q [, AR_LAGS=array] [, AUTOCOV=variable]
    [, BACKWARD_ORIGIN=value] [, CONFIDENCE=value] [, CONSTANT]
    [, /DOUBLE] [, ERR_REL=value] [, FORECAST=variable]
    [, INIT_EST_AR=array] [, INIT_EST_MA=array] [, ITMAX=value] [, /LSQ]
    [, LGTH_BACKCAST=value] [, MA_LAGS=array] [, MEAN_EST=value]
    [, /MOMENTS] [, N_PREDICT=value] [, /NO_CONSTANT]
    [, PARAM_EST_COV=variable] [, RESIDUAL=variable]
    [, SS_RESIDUAL=variable] [, TOL_BACKCAST=value]
    [, TOL_CONVERGENCE=value])
```

Return Value

An array of length $1 + p + q$ with the estimated constant, AR, and MA parameters. If *No_Constant* is specified, the 0-th element of this array is 0.0.

Arguments

p

Number of autoregressive parameters.

q

Number of moving average parameters.

z

One-dimensional array containing the observations.

Keywords

AR_LAGS

One-dimensional array of length p containing the order of the nonzero autoregressive parameters. The elements of *Ar_Lags* must be greater than or equal to 1. Default: $Ar\_Lags = [1, 2, \dots, p]$

AUTOCOV

Named variable into which an array of length $p + q + 2$ containing the variance and autocovariances of the time series z is stored. Keyword *Autocov*(0) contains the variance of the series z . Keyword *Autocov*(k) contains the autocovariance of lag k , where $k = 1, \dots, p + q + 1$.

BACKWARD_ORIGIN

Maximum backward origin. Keyword *Backward_Origin* must be greater than or equal to zero and less than or equal to $N\_ELEMENTS(z) - (\max(\maxar, \maxma))$, where $\maxar = \max(Ar\_Lags)$ and $\maxma = \max(Ma\_Lags)$.

Forecasts at origins $N\_ELEMENTS(z) - Backward\_Origin$ through $N\_ELEMENTS(z)$ are generated. Default: $Backward\_Origin = 0$

CONFIDENCE

Value in the exclusive interval (0, 100) used to specify the confidence level of the forecasts. Typical choices for *Confidence* are 90.0, 95.0, and 99.0. Default: $Confidence = 95.0$

CONSTANT

If present and nonzero, the time series is centered about its mean. Keywords *No_Constant* and *Constant* cannot be used together.

DOUBLE

If present and nonzero, double precision is used.

ERR_REL

Stopping criterion for use in the nonlinear equation solver used in both the method-of-moments and least-squares algorithms. Default: $Err\_Rel = 100 \times \epsilon$, where ϵ is machine precision

FORECAST

Named variable into which an array of length $N\_Predict \times (Backward\_Origin + 3)$ containing the forecasts up to $N\_Predict$ steps ahead and the information necessary to obtain confidence intervals is stored. Keywords *Forecast* and *N_Predict* must be used together.

INIT_EST_AR

Array of length p containing preliminary estimates of the autoregressive parameters, internally. Keywords *Init_Est_Ar* and *Init_Est_Ma* must be used together and are only applicable if *Lsq* is also present and nonzero.

INIT_EST_MA

Array of length q containing preliminary estimates of the moving average parameters. Keywords *Init_Est_Ar* and *Init_Est_Ma* must be used together and are only applicable if *Lsq* is also present and nonzero.

The following keywords are used to forecast up to $N\_Predict$ steps ahead and the information necessary to obtain confidence intervals:

ITMAX

Maximum number of iterations allowed in the nonlinear equation solver used in both the method-of-moments and least-squares algorithms. Default: *Itmax* = 200

LSQ

If present and nonzero, the autoregressive and moving average parameters are estimated by a least-squares procedure. Keywords *Moments* and *Lsq* cannot be used together.

LGTH_BACKCAST

Specifies the maximum length of backcasting. Must be greater than or equal to zero. Keywords *Lgth_Backcast* and *Tol_Backcast* must be used together. Default: *Lgth_Backcast* = 10

MA_LAGS

One-dimensional array of length q containing the order of the nonzero moving average parameters. The elements of *Ma_Lags* must be greater than or equal to 1. Default: *Ma_Lags* = [1, 2, ..., q]

MEAN_EST

Initial estimate of the mean of the time series z .

Default:

$$Mean\_Est = \sum_{t=1}^n z_t / n$$

MOMENTS

If present and nonzero, the autoregressive and moving average parameters are estimated by a method-of-moments procedure. Keywords *Moments* and *Lsq* cannot be used together. (Default)

N_PREDICT

Maximum lead time for forecasts. Keyword *N_Predict* must be greater than zero. Keywords *Forecast* and *N_Predict* must be used together.

NO_CONSTANT

If present and nonzero, the time series is not centered about its mean. Keywords *No_Constant* and *Constant* cannot be used together.

PARAM_EST_COV

Named variable into which an array, containing the covariance matrix of the final parameter estimates, is stored. The array is of size $np \times np$, where $np = p + q + 1$ if z is centered about its mean and $np = p + q$ if z is not centered. The ordering of variables in *Param_Est_Cov* is *Mean_Est*, *Ar_lags*, and *Ma_lags*.

RESIDUAL

Named variable into which an array of length $N\_ELEMENTS(z) - (\max(Ar\_Lags)) + Lgth\_Backcast$ containing the residuals (including backcasts) at the final parameter estimate point in the first $N\_ELEMENTS(z) - (\max(Ar\_Lags)) + nb$, where nb is the number of values backcast is stored.

SS_RESIDUAL

Named variable into which the sum of squares of the random error is stored.

TOL_BACKCAST

Specifies the tolerance level used to determine convergence of the backcast algorithm. Typically, *Tol_Backcast* is set to a fraction of an estimate of the standard deviation of the time series. Keywords *Lgth_Backcast* and *Tol_Backcast* must be used together. Default: *Tol_Backcast* = 0.01 x standard deviation of *l*

TOL_CONVERGENCE

Tolerance level used to determine convergence of the nonlinear least-squares algorithm. Keyword *Tol_Convergence* represents the minimum relative decrease in sum of squares between two iterations required to determine convergence. Hence, *Tol_Convergence* must be greater than or equal to zero. Default: $\max \{10^{-10}, \epsilon^{2/3}\}$ for single precision, $\max \{10^{-20}, \epsilon^{2/3}\}$ for double precision, where ϵ is machine precision.

Discussion

The *IMSL_ARMA* function computes estimates of parameters for a nonseasonal *IMSL_ARMA* model given a sample of observations, $\{Z_t\}$, for $t = 1, 2, \dots, n$, where $n = N\_ELEMENTS(z)$. You may choose either method of moments or least squares. The default is method of moments.

Choose the method-of-moments algorithm with the keyword *Moments*. The least-squares algorithm is used if *Lsq* is specified. If you wish to use the least-squares algorithm, the preliminary estimates are the method-of-moments estimates by default; otherwise, you can input initial estimates by specifying keywords *Init_Est_Ar* and *Init_Est_Ma*. [Table 20-1](#) lists the appropriate keywords for both the method-of-moments and least-squares algorithm:

Method of Moments only	Least Squares only	Both Method of Moments and Least Squares
Moments	Lsq	Err_Rel
	Constant (or No_Constant)	Itmax
	Ar_Lags	Mean_Estimate
	Ma_Lags	Autocov

Table 20-1: Method-of-Moments and Least-Squares Keywords

Method of Moments only	Least Squares only	Both Method of Moments and Least Squares
	Lgth_Backcast	Forecast
	Tol_Backcast	N_Predict
	Tol_Convergence	Confidence
	Init_Est_Ar	Backward_Origin
	Init_Est_Ma	
	Residual	
	Param_Est_Cov	
	Ss_Residual	

Table 20-1: Method-of-Moments and Least-Squares Keywords (Continued)

Method-of-moments Estimation

Suppose the time series $\{Z_t\}$ is generated by an IMSL_ARMA(p, q) model of the form:

$$\phi(B)Z_t = \theta_0 + \theta(B)A_t$$

for

$$t \in \{0, \pm 1, \pm 2, \dots\}$$

Let

$$\hat{\mu} = \text{Mean\_Est}$$

be the estimate of the mean μ of the time series $\{Z_t\}$, where:

$$\hat{\mu}$$

equals the following:

$$\hat{\mu} = \begin{cases} \mu & \text{for } \mu \text{ known} \\ \frac{\sum_{t=1}^n Z_t}{n} & \text{for } \mu \text{ unknown} \end{cases}$$

The autocovariance function is estimated by:

$$\hat{\sigma}(k) = \frac{1}{n} \sum_{t=1}^{n-k} (Z_t - \hat{\mu})(Z_{t+k} - \hat{\mu})$$

for $k = 0, 1, \dots, K$, where $K = p + q + 1$. Note that:

$$\hat{\sigma}(0)$$

is an estimate of the sample variance.

Given the sample autocovariances, the function computes the method-of-moments estimates of the autoregressive parameters using the extended Yule-Walker equations as follows:

$$\hat{\Sigma} \hat{\phi} = \hat{\sigma}$$

where:

$$\hat{\phi} = (\hat{\phi}_1, \dots, \hat{\phi}_p)^T$$

$$\Sigma_{ij} = \sigma(|q + i - j|), \quad i, j = 1, \dots, p$$

$$\hat{\sigma}_i = \hat{\sigma}(q + i), \quad i, j = 1, \dots, p$$

The overall constant θ_0 is estimated by the following:

$$\hat{\theta}_0 = \begin{cases} \hat{\mu} & \text{for } p = 0 \\ \hat{\mu} \left(1 - \sum_{i=1}^p \hat{\phi}_i \right) & \text{for } p > 0 \end{cases}$$

The moving average parameters are estimated based on a system of nonlinear equations given $K = p + q + 1$ autocovariances, $\sigma(k)$ for $k = 1, \dots, K$, and p autoregressive parameters ϕ_i for $i = 1, \dots, p$.

Let $Z'_t = \phi(B)Z_t$. The autocovariances of the derived moving average process $Z'_t = \theta(B)A_t$ are estimated by the following relation:

$$\hat{\sigma}'(k) = \begin{cases} \hat{\sigma}(k) & \text{for } p = 0 \\ \sum_{i=0}^p \sum_{j=0}^p \hat{\phi}_i \hat{\phi}_j (\hat{\sigma}(|k + i - j|)) & \text{for } p \geq 1, \hat{\phi}_0 \equiv -1 \end{cases}$$

The iterative procedure for determining the moving average parameters is based on the relation:

$$\sigma(k) = \begin{cases} (1 + \theta_1^2 + \dots + \theta_q^2)\sigma_A^2 & \text{for } k = 0 \\ (-\theta_k + \theta_1\theta_{k+1} + \dots + \theta_{q-k}\theta_q)\sigma_A^2 & \text{for } k \geq 1 \end{cases}$$

where $\sigma(k)$ denotes the autocovariance function of the original Z_t process.

Let $\tau = (\tau_0, \tau_1, \dots, \tau_q)^T$, $f = (f_0, f_1, \dots, f_q)^T$, and T be a $(q+1) \times (q+1)$ matrix, where τ_j , f_j , and T are as follows:

$$\tau_j = \begin{cases} \sigma_A & \text{for } j = 0 \\ -\theta_j/\tau_0 & \text{for } j = 1, \dots, q \end{cases}$$

and:

$$f_j = \sum_{i=0}^{q-j} \tau_i \tau_{i+j} - \hat{\sigma}'(j) \text{ for } j = 0, 1, \dots, q$$

$$T = \begin{bmatrix} \tau_0 & \tau_1 & \dots & \dots & \tau_q \\ \tau_1 & \tau_2 & \dots & \tau_q & 0 \\ \dots & \dots & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots \\ \tau_q & 0 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} \tau_0 & \dots & \dots & \tau_q \\ 0 & \tau_0 & \dots & \tau_{q-1} \\ 0 & 0 & \dots & \dots \\ \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \tau_0 \end{bmatrix}$$

Then, the value of τ at the $(i+1)$ -th iteration is determined by:

$$\tau^{i+1} = \tau^i - (T^i)^{-1} f^i$$

The estimation procedure begins with the initial value:

$$\tau^0 = (\sqrt{\hat{\sigma}'(0)}, 0, \dots, 0)^T$$

and terminates at iteration i when either $|f^i|$ is less than Err_Rel or i equals $Itmax$. The moving average parameter estimates are obtained from the final estimate of τ by setting:

$$\hat{\theta}_j = -\tau_j/\tau_0$$

for $j = 1, \dots, q$. The random error variance is estimated by the following:

See Box and Jenkins (1976, pp. 498–500) for a description of a function that performs similar computations.

$$\hat{\sigma}_A^2 = \begin{cases} \hat{\sigma}(0) - \sum_{i=1}^p \hat{\phi}_i \hat{\sigma}(i) & \text{for } q = 0 \\ \tau_0^2 & \text{for } q > 0 \end{cases}$$

Least-squares Estimation

Suppose the time series $\{Z_t\}$ is generated by a nonseasonal IMSL_ARMA model of the form:

$$\theta(B)(Z_t - \mu) = \theta(B)A_t, \quad \text{for } t \in \{0, \pm 1, \pm 2, \dots\}$$

where B is the backward-shift operator, μ is the mean of Z_t , and:

$$\phi(B) = 1 - \phi_1 B^{l_\phi(1)} - \phi_2 B^{l_\phi(2)} - \dots - \phi_p B^{l_\phi(p)} \quad \text{for } p \geq 0$$

$$\theta(B) = 1 - \theta_1 B^{l_\theta(1)} - \theta_2 B^{l_\theta(2)} - \dots - \theta_q B^{l_\theta(q)} \quad \text{for } q \geq 0$$

with p autoregressive and q moving average parameters. Without loss of generality, the following is assumed:

$$1 \leq l_\phi(1) \leq l_\phi(2) \leq \dots \leq l_\phi(p)$$

$$1 \leq l_\theta(1) \leq l_\theta(2) \leq \dots \leq l_\theta(q)$$

so that the nonseasonal IMSL_ARMA model is of order (p', q') , where:

$$p' = l_\phi(p) \quad \text{and} \quad q' = l_\theta(q)$$

Note that the usual hierarchical model assumes the following:

$$l_\phi(i) = i, \quad 1 \leq i \leq p$$

$$l_\theta(j) = j, \quad 1 \leq j \leq q$$

Consider the sum-of-squares function:

$$S_T(\mu, \phi, \theta) = \sum_{-T+1}^n [A_t]^2$$

where:

$$[A_t] = E[A_t | (\mu, \phi, \theta, Z)]$$

and $T = \text{Lgth\_Backcast}$ is the length of backcasting from the beginning of the series. The random errors $\{A_t\}$ are assumed to be independent and identical distributed $N(0, \sigma_A^2)$ random variables. Hence, the log-likelihood function is given by:

$$l(\mu, \phi, \theta, \sigma_A) = f(\mu, \phi, \theta) - n \ln(\sigma_A) - \frac{S_T(\mu, \phi, \theta)}{2\sigma_A^2}$$

where $f(\mu, \phi, \theta)$ is a function of μ , ϕ , and θ .

For $T = 0$, the log-likelihood function is conditional on the past values of both Z_t and A_t required to initialize the model. The method of selecting these initial values usually introduces transient bias into the model (Box and Jenkins 1976, pp. 210–211). For $T = \text{infinity}$, this dependency vanishes, and the estimation problem concerns maximization of the unconditional log-likelihood function. Box and Jenkins (1976, p. 213) argue that:

$$S_\infty(\mu, \phi, \theta) / (2\sigma_A^2) \quad \text{dominates} \quad l(\mu, \phi, \theta, \sigma_A^2)$$

The parameter estimates that minimize the sum-of-squares function are called least-squares estimates. For large n , the unconditional least-squares estimates are approximately equal to the maximum likelihood-estimates.

In practice, a finite value of T enables sufficient approximation of the unconditional sum-of-squares function. The values of $[A_t]$ needed to compute the unconditional sum of squares are computed iteratively with initial values of Z_t obtained by backcasting. The residuals (including backcasts), estimate of random error variance, and covariance matrix of the final parameter estimates also are computed. ARIMA parameters can be computed using the “[IMSL_DIFFERENCE](#)” on page 929, together with [IMSL_ARMA](#).

Forecasting Option

The Box-Jenkins forecasts and their associated confidence intervals for a nonseasonal [IMSL_ARMA](#) model are computed given a sample of $n = N\_ELEMENTS(z)$ $\{Z_t\}$ for $t = 1, 2, \dots, n$.

Suppose the time series $\{Z_t\}$ is generated by a nonseasonal [IMSL_ARMA](#) model of the form:

$$\begin{aligned} \phi(B) Z_t &= \theta_0 + \theta(B) A_t \\ \text{for } t \in \{0, \pm 1, \pm 2, \dots\} \end{aligned}$$

where B is the backward-shift operator, θ_0 is the constant, and:

$$\phi(B) = 1 - \phi_1 B^{l_\phi(1)} - \phi_2 B^{l_\phi(2)} - \dots - \phi_p B^{l_\phi(p)}$$

$$\theta(B) = 1 - \theta_1 B^{l_\theta(1)} - \theta_2 B^{l_\theta(2)} - \dots - \theta_q B^{l_\theta(q)}$$

with p autoregressive and q moving average parameters. Without loss of generality, the following is assumed:

$$1 \leq l_\phi(1) \leq l_\phi(2) \leq \dots \leq l_\phi(p)$$

$$1 \leq l_\theta(1) \leq l_\theta(2) \leq \dots \leq l_\theta(q)$$

so that the nonseasonal IMSL_ARMA model is of order (p', q') , where:

$$p' = l_\theta(p) \text{ and } q' = l_\theta(q)$$

Note that the usual hierarchal model assumes the following:

$$l_\phi(i) = i, \quad 1 \leq i \leq p$$

$$l_\theta(j) = j, \quad 1 \leq j \leq q$$

The Box-Jenkins forecast at origin t for lead time l of Z_{t+l} is defined in terms of the difference equation:

$$\hat{Z}_t(l) = \theta_0 + \phi_1 [Z_{t+l-l_\phi(1)}] + \dots + \phi_p [Z_{t+l-l_\phi(p)}] + [A_{t+l}] - \dots$$

$$\theta_1 [A_{t+l-l_\theta(1)}] - [A_{t+l}] - \theta_1 [A_{t+l-l_\theta(1)}] - \dots - \theta_q [A_{t+l-l_\theta(q)}]$$

where the following is true:

$$[Z_{t+k}] = \begin{cases} Z_{t+k} & \text{for } k = 0, -1, -2, \dots \\ \hat{Z}_t(k) & \text{for } k = 1, 2, \dots \end{cases}$$

$$[A_{t+k}] = \begin{cases} Z_{t+k} - \hat{Z}_{t+k-1}(1) & \text{for } k = 0, -1, -2, \dots \\ 0 & \text{for } k = 1, 2, \dots \end{cases}$$

The $100(1 - \alpha)$ -percent confidence interval for Z_{t+l} is given by:

$$\hat{Z}_t(l) \pm z_{(1-\alpha/2)} \left\{ \sum_{j=0}^{l-1} \Psi_j^2 \right\}^{1/2} \sigma_A$$

where $Z_{(1-\alpha/2)}$

is the 100 $(1 - \alpha / 2)$ -percentile of the standard normal distribution, σ_A is the standard deviation of the random error, and ψ_j is defined as follows:

$$\psi_j = \begin{cases} 1 & \text{for } j = 0 \\ \sum_{i=1}^j \phi_i \psi_{j-i} - \theta_j & \text{for } j > 0 \end{cases}$$

In this equation, $\phi_i = 0$ for $i > p$ and $\theta_j = 0$ for $j > q$. Note that the forecasts are computed for lead times $l = 1, 2, \dots, L$ at origins $t = (n - b), (n - b + 1), \dots, n$, where $L = N\_Predict$ and $b = Backward\_Origin$.

The Box-Jenkins forecasts minimize the mean-square error:

$$E[Z_{t+1} - \hat{Z}_t(l)]^2$$

Also, the forecasts are easily updated according to the following equation:

$$\hat{Z}_{t+1}(l) = \hat{Z}_t(l+1) + \psi_l A_{t+1}$$

This approach and others are discussed in Chapter 5 of Box and Jenkins (1976).

Examples

Example 1

Consider the Wolfer Sunspot Data (Anderson 1971, p. 660) consisting of the number of sunspots observed each year from 1749 through 1924. The data set for this example consists of the number of sunspots observed from 1770 through 1869 and is shown in [Figure 20-1](#). The method-of-moments estimates:

$$\hat{\theta}_0, \hat{\phi}_1, \hat{\phi}_2 \text{ and } \hat{\theta}_1$$

for the IMSL_ARMA(2,1) model are:

$$Z_t = \theta_0 + \phi_1 Z_{t-1} + \phi_2 Z_{t-2} - \theta_1 A_{t-1} + A_t$$

where Z_t is “raw” data and the errors A_t are independently and identically normally distributed with mean zero and variance σ_A^2 .

```
temp = IMSL_STATDATA(2)
; Get the Wolfer Sunspot Data.
z = TEMP(21:120, 1)
; Use only 100 observations, 1770-1869.
years = FINDGEN(100) + 1770
PLOT, years, z, XStyle = 1, Psym = -6, $
Title = 'Wolfer Sunspot Data', XTitle = 'Year', $
```



```

      YTitle = 'Number of Sunspots'
; Plot the data.
p = 2
q = 1
parameters = IMSL_ARMA(z, p, q)
; Perform time-series analysis.
PRINT, 'AR estimates:', parameters(1), parameters(2)
PRINT, 'MA estimate :', parameters(3)

AR estimates: 1.24426 -0.575149
MA estimate  : -0.124094

```

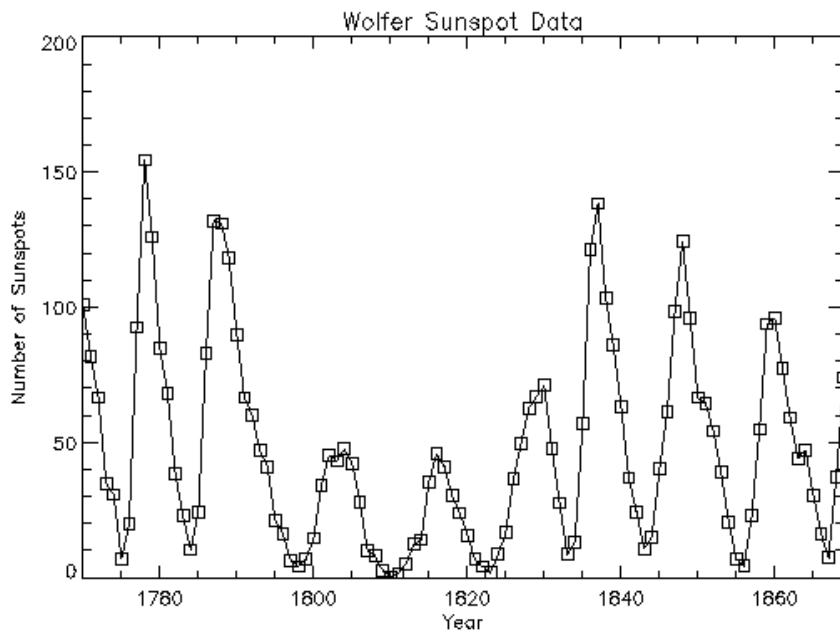


Figure 20-1: Wolfer Sunspot Data Plot

Example 2

The data for this example are the same as that for the initial example. Preliminary method-of-moments estimates are computed by default, and the method of least squares is used to find the final estimates.

```

temp = IMSL_STATDATA(2)
; Get the Wolfer Sunspot Data.

```

```

z = TEMP(21:120, 1)
; Use only 100 observations, 1770-1869.
parameters = IMSL_ARMA(z, 2, 1, /Lsq, Tol_Convergence = .125)
; Perform time-series analysis using method of moments. The
; warning error can be ignored in this case.
PRINT, 'AR estimates:', parameters(1), parameters(2)
PRINT, 'MA estimate :', parameters(3)

AR estimates: 1.39257 -0.732948
MA estimate : -0.137512

```

Example 3

Consider the Wolfer Sunspot Data (Anderson 1971, p. 660) consisting of the number of sunspots observed each year from 1749 through 1924. The data set for this example consists of the number of sunspots observed from 1770 through 1869. IMSL_ARMA computes forecasts and 95-percent confidence limits for the forecasts for an IMSL_ARMA(2, 1) model fit using IMSL_ARMA with the method-of-moments option. With *Backward-Origin* = 3, columns zero through three of *Forecast* provide forecasts given the data through 1866, 1867, 1868, and 1869. Column five gives the deviations from the forecast for computing confidence limits, and column six gives the psi weights, which can be used to update forecasts when more data is available. For example, the forecast for the 102-nd observation (year 1871) given the data through the 100-th observation (year 1869) is 77.21; 95-percent confidence limits are given by:

$$77.21 \mp 56.30$$

After observation 101 (Z_{101} for year 1870) is available, the forecast can be updated by using:

$$\hat{Z}_{t+1}(l) = \hat{Z}_t(l+1) + \psi_l[Z_{t+1} - \hat{Z}_t(1)]$$

with the psi weight ($\psi_1 = 1.37$) and the one-step-ahead forecast error for observation 101 ($Z_{101} - 83.72$) to give the following:

$$77.21 + 1.37 \times (Z_{101} - 83.72)$$

Since this updated forecast is one step ahead, the 95-percent confidence limits are now given by the forecast:

$$\mp 33.22$$

First, define a procedure to output the results:

```

.RUN
PRO print_results, parameters, forecast
  PRINT, 'Method-of-moments initial estimates:'
  PRINT, 'AR estimates:', parameters(1), parameters(2)

```

```

PRINT, 'MA estimate :', parameters(3)
PRINT
lead_time = INDGEN(12) + 1
forecast = [[lead_time], [forecast]]
PRINT, 'Forecasts from ...'
PRINT, 'lead time', ' 1866', ' 1867', $
      ' 1868', ' 1869', ' Deviat.', ' Psi'
PM, forecast, FORMAT = '(i6, 3x, 6f9.4)'
END

temp = IMSL_STATDATA(2)
; Get the Wolfer Sunspot Data.
z = TEMP(21:120, 1)
; Use only 100 observations, 1770-1869.
parameters = IMSL_ARMA(z, 2, 1, Itmax = 0, Err_Rel = 0.0, $
      Forecast = forecast, N_Predict = 12, Backward_Origin = 3)
; Perform time-series analysis using method-of-moments.
print_results, parameters, forecast
years = INDGEN(100) + 1770
PLOT, years, z, $
      Psym = -6, Symsize = .5, XStyle = 1, XRange = [1770, 1885], $
      YRange = [-50, 175], Title = 'Wolfer Sunspot Data', $
      XTitle = 'Year', YTitle = 'Number of Sunspots'
; Plot the data along with the forecasted values with confidence
; intervals.
OPLOT, INDGEN(10) + 1870, forecast(*, 3), Psym = 4, Symsize = .5
ERRPLOT, indgen(10) + 1870, forecast(*, 3) - forecast(*, 4), $
      forecast(*, 3) + forecast(*, 4), Width = .005

Method-of-moments initial estimates:
AR estimates:      1.24426      -0.575149
MA estimate :      -0.124094
Forecasts from ...
lead time 1866      1867      1868      1869      Deviat.  Psi
1      18.2833  16.6151  55.1893  83.7196  33.2179  1.3684
2      28.9182  32.0189  62.7606  77.2092  56.2980  1.1274
3      41.0101  45.8275  61.8922  63.4608  67.6168  0.6158
4      49.9387  54.1496  56.4571  50.0987  70.6432  0.1178
5      54.0937  56.5623  50.1939  41.3803  70.7515 -0.2076
6      54.1282  54.7780  45.5268  38.2174  71.0869 -0.3261
7      51.7815  51.1701  43.3221  39.2965  71.9074 -0.2863
8      48.8417  47.7072  43.2631  42.4582  72.5337 -0.1687
9      46.5335  45.4736  44.4577  45.7715  72.7498 -0.0452
10     45.3524  44.6861  45.9781  48.0758  72.7653  0.0407
11     45.2103  44.9909  47.1827  49.0371  72.7779  0.0767
12     45.7128  45.8230  47.8072  48.9080  72.8225  0.0720

```

The plot of the forecasts and the confidence limits from year 1869 are shown in [Figure 20-2](#).

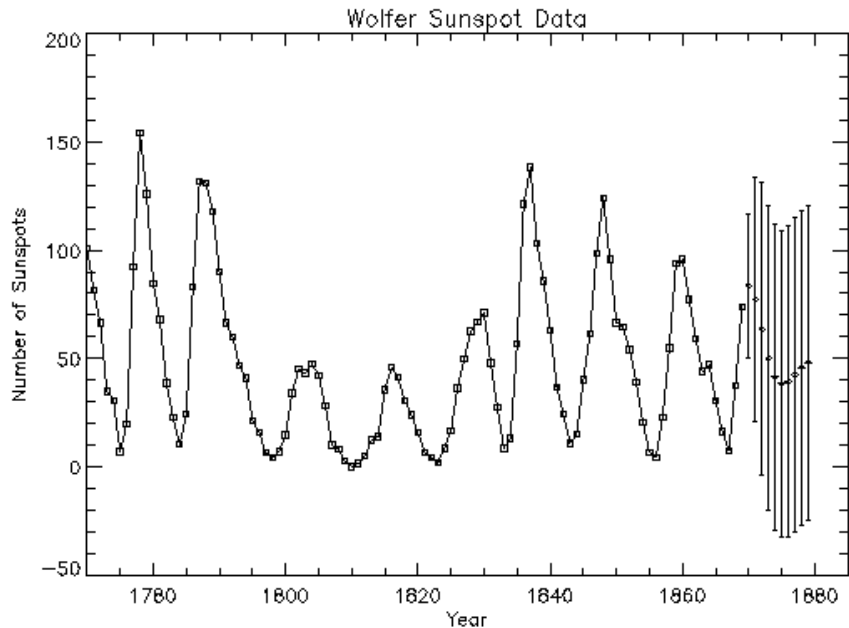


Figure 20-2: Sunspot Data with Predicted Values and Confidence Bands

Version History

6.4	Introduced

IMSL_DIFFERENCE

The IMSL_DIFFERENCE function differences a seasonal or nonseasonal time series.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_DIFFERENCE(z, periods [, /DOUBLE] [, /EXCLUDE_FIRST]  
[, /FIRST_TO_NAN] [, NUM_LOST=variable] [, ORDERS=array] )
```

Return Value

One-dimensional array of length N_ELEMENTS (z) containing the differenced series.

Arguments

z

One-dimensional array containing the time series.

periods

One-dimensional array containing the periods at which z is to be differenced.

Keywords

DOUBLE

If present and nonzero, double precision is used.

EXCLUDE_FIRST

If *Exclude_First* is present and nonzero, the first *Num_Lost* observations are excluded from the solution due to differencing. The differenced series is of length N_ELEMENTS(*periods*) – *Num_Lost*. If *First_To_Nan* is specified, the first

Num_Lost observations are set to NaN (Not a Number). This is the default if neither *Exclude_First* nor *First_To_Nan* is specified. Default: *First_To_Nan*

FIRST_TO_NAN

If *Exclude_First* is present and nonzero, the first *Num_Lost* observations are excluded from the solution due to differencing. The differenced series is of length $N\_ELEMENTS(periods) - Num\_Lost$. If *First_To_Nan* is specified, the first *Num_Lost* observations are set to NaN (Not a Number). This is the default if neither *Exclude_First* nor *First_To_Nan* is specified. Default: *First_To_Nan*

NUM_LOST

Named variable into which the number of observations “lost” because of differencing the time series *z* is stored.

ORDERS

One-dimensional array of length $N\_ELEMENTS(periods)$ containing the order of each difference given in periods. The elements of *Orders* must be greater than or equal to 0. Default: all the elements equal 1

Discussion

The `IMSL_DIFFERENCE` function performs $m = N\_ELEMENTS(periods)$ successive backward differences of period $s_i = periods(i - 1)$ and $d_i = Orders(i - 1)$ for $i = 1, \dots, m$ on the $n = N\_ELEMENTS(x)$ observations $\{Z_t\}$ for $t = 1, 2, \dots, n$. Consider the backward shift operator B given by:

$$B^k Z_t = Z_{t-k}$$

for all k . Then, the *backward difference operator* with period s is defined by the following:

$$\Delta_s Z_t = (1 - B^s) Z_t = Z_t - Z_{t-s} \quad \text{for } s \geq 0$$

Note that $B_s Z_t$ and $\Delta_s Z_t$ are defined only for $t = (s + 1), \dots, n$. Repeated differencing with period s is simply:

$$\Delta_s^d Z_t = (1 - B^s)^d Z_t = \sum_{j=0}^d \frac{d!}{j!(d-j)!} (-1)^j B^{sj} Z_t$$

where $d \geq 0$ is the order of differencing. Note that $\Delta_s^d Z_t$ is defined only for $t = (sd + 1), \dots, n$.

The general difference formula used in `IMSL_DIFFERENCE` is given by:

$$W_t = \begin{cases} \text{NaN} & \text{for } t = 1, \dots, n_L \\ \Delta_{s_1}^{d_1} \Delta_{s_2}^{d_2} \dots \Delta_{s_m}^{d_m} Z_t & \text{for } t = n_L + 1, \dots, n \end{cases}$$

where n_L represents the number of observations “lost” because of differencing and NaN represents the missing value code. See `IMSL_MACHINE` to retrieve missing values. Note that:

$$n_L = \sum_j s_j d_j$$

A homogeneous, stationary time series can be arrived at by appropriately differencing a homogeneous, nonstationary time series (Box and Jenkins 1976, p. 85). Preliminary application of an appropriate transformation followed by differencing of a series enables model identification and parameter estimation in the class of homogeneous stationary `IMSL_ARMA`.

Examples

Example 1

Consider the Airline Data (Box and Jenkins 1976, p. 531) consisting of the monthly total number of international airline passengers from January 1949 through December 1960. The entire data, after taking a natural logarithm, are shown in [Figure 20-3](#). The plot shows a linear trend and a seasonal pattern with a period of 12 months. This suggests that the data needs a nonseasonal difference operator, Δ_1 , and a seasonal difference operator, Δ_{12} , to make the series stationary. The `IMSL_DIFFERENCE` function is used to compute:

$$W_t = \Delta_1 \Delta_{12} Z_t = (Z_t - Z_{t-12}) - (Z_{t-1} - Z_{t-13})$$

for $t = 14, 15, \dots, 24$.

```
ztemp = ALOG(IMSL_STATDATA(4))
; Get the data set.
PLOT, INDGEN(144), ztemp, Psym = -6, Symsize = .5, $
  YStyle = 1, Title = 'Complete Airline Data', $
  XTitle = 'Month (beginning 1949)', $
  YTitle = '!8ln!3(thousands of Passengers)'
; Plot the complete data set.
z = ztemp(0:23)
periods = [1, 12]
difference = IMSL_DIFFERENCE(z, periods)
```

```

; Call IMSL_DIFFERENCE.
matrix = [[INDGEN(24)], [z], [difference]]
; Create a matrix of the data to make the output easier.
PM, matrix, FORMAT = '(i4, x, 2f7.1)', $
    Title = '      I      z(i)      difference(i)'

; Output the results.

```

I	z(i)	difference(i)
0	4.7	NaN
1	4.8	NaN
2	4.9	NaN
3	4.9	NaN
4	4.8	NaN
5	4.9	NaN
6	5.0	NaN
7	5.0	NaN
8	4.9	NaN
9	4.8	NaN
10	4.6	NaN
11	4.8	NaN
12	4.7	NaN
13	4.8	0.0
14	4.9	0.0
15	4.9	-0.0
16	4.8	-0.0
17	5.0	0.1
18	5.1	0.0
19	5.1	0.0
20	5.1	0.0
21	4.9	-0.0
22	4.7	-0.0
23	4.9	0.1

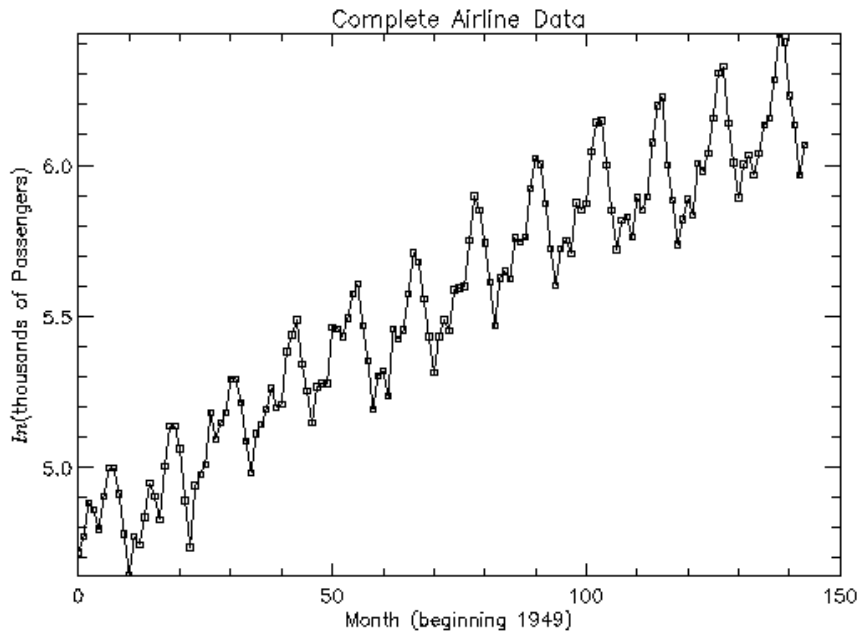


Figure 20-3: Complete Airline Data Plot

Example 2

The data for this example is the same as that for the initial example. The first *Num_Lost* observations are excluded from *W* due to differencing, and *Num_Lost* also is output.

```
ztemp = ALOG(IMSL_STATDATA(4))
z = ztemp(0:23)
periods = [1, 12]
diff = IMSL_DIFFERENCE(z, periods, $
    /EXCLUDE_FIRST, NUM_LOST = num_lost)
num_valid = N_ELEMENTS(z) - num_lost
; Use Num_Lost to compute the number of rows in the result
; that have valid values.
matrix = [[INDGEN(num_valid)], [z(0:num_valid-1)], $
    [DIFF(0:num_valid-1)]]
; Put the data in one matrix to make printing easier.
PM, matrix, FORMAT = '(i4, x, 2f7.1)', $
    TITLE = '    i      z(i)    IMSL_DIFFERENCE(i)'
```

i	z (i)	IMSL_DIFFERENCE (i)
0	4.7	0.0
1	4.8	0.0
2	4.9	-0.0
3	4.9	-0.0
4	4.8	0.1
5	4.9	0.0
6	5.0	0.0
7	5.0	0.0
8	4.9	-0.0
9	4.8	-0.0
10	4.6	0.1

Errors

Fatal Errors

STAT_PERIODS_LT_ZERO—Parameter *periods* (#) = #. All elements of *Periods* must be greater than zero.

STAT_ORDER_NEGATIVE—Parameter *order* (#) = #. All elements of *order* must be nonnegative.

STAT_Z_CONTAINS_NAN—Parameter *z* (#) = NaN; *z* cannot contain missing values. Other elements of *z* may be equal to NaN.

Version History

6.4	Introduced

IMSL_BOXCOXTRANS

The IMSL_BOXCOXTRANS function performs a forward or an inverse Box-Cox (power) transformation.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_BOXCOXTRANS(z, power [, /DOUBLE] [, /INVERSE]  
[, S=parameter] )
```

Return Value

One-dimensional array containing the transformed data.

Arguments

power

Exponent parameter in the Box-Cox (power) transformation.

z

One-dimensional array containing the observations.

Keywords

DOUBLE

If present and nonzero, double precision is used.

INVERSE

If present and nonzero, the inverse transform is performed.

S

Shift parameter in the Box-Cox (power) transformation. Parameter shift must satisfy the relation $\min(z(i)) + S > 0$. Default: $S = 0.0$.

Discussion

The IMSL_BOXCOXTRANS function performs a forward or an inverse Box-Cox (power) transformation of $n = N\_ELEMENTS(z)$ observations $\{Z_t\}$ for $t = 0, 1, \dots, n-1$.

The forward transformation is useful in the analysis of linear models or models with non-normal errors or non-constant variance (Draper and Smith 1981, p. 222). In the time series setting, application of the appropriate transformation and subsequent differencing of a series can enable model identification and parameter estimation in the class of homogeneous stationary autoregressive-moving average models. The inverse transformation can later be applied to certain results of the analysis, such as forecasts and prediction limits of forecasts, in order to express the results in the scale of the original data. A brief note concerning the choice of transformations in the time series models is given in Box and Jenkins (1976, p. 328).

The class of power transformations discussed by Box and Cox (1964) is defined by:

$$X_t = \begin{cases} \frac{(Z_t + \xi)^\lambda - 1}{\lambda} & \lambda \neq 0 \\ \ln(Z_t + \xi) & \lambda = 0 \end{cases}$$

where $Z_t + \xi > 0$ for all t . Since:

$$\lim_{\lambda \rightarrow 0} \frac{(Z_t + \xi)^\lambda - 1}{\lambda} = \ln(Z_t + \xi)$$

the family of power transformations is continuous.

Let $\lambda = \text{power}$ and $\xi = S$; then, the computational formula used by IMSL_BOXCOXTRANS is given by:

$$X_t = \begin{cases} (Z_t + \xi)^\lambda & \lambda \neq 0 \\ \ln(Z_t + \xi) & \lambda = 0 \end{cases}$$

where $Z_t + \xi > 0$ for all t . The computational and Box-Cox formulas differ only in the scale and origin of the transformed data. Consequently, the general analysis of the data is unaffected (Draper and Smith 1981, p. 225).

The inverse transformation is computed by:

$$X_t = \begin{cases} Z_t^{1/\lambda} - \xi & \lambda \neq 0 \\ \exp(Z_t) - \xi & \lambda = 0 \end{cases}$$

where $\{Z_t\}$ now represents the result computed by IMSL_BOXCOXTRANS for a forward transformation of the original data using parameters λ and ξ .

Examples

Example 1

The following example performs a Box-Cox transformation with *power* = 2.0 on 10 data points.

```
power = 2.0
z = [1.0, 2.0, 3.0, 4.0, 5.0, 5.5, 6.5, 7.5, 8.0, 10.0]
; Transform Data using Box Cox Transform
x = IMSL_BOXCOXTRANS(z, power)
PM, x, Title = 'Transformed Data'
```

```
Transformed Data
1.00000
4.00000
9.00000
16.0000
25.0000
30.2500
42.2500
56.2500
64.0000
100.000
```

Example 2

This example extends the first example—an inverse transformation is applied to the transformed data to return to the original data values.

```
power = 2.0
z = [1.0, 2.0, 3.0, 4.0, 5.0, 5.5, 6.5, 7.5, 8.0, 10.0]
x = IMSL_BOXCOXTRANS(z, power)
PM, x, Title = 'Transformed Data'
```

```
Transformed Data
1.00000
4.00000
9.00000
```

```

16.0000
25.0000
30.2500
42.2500
56.2500
64.0000
100.000
; Perform an Inverse Transform on the Transformed Data
y = IMSL_BOXCOXTRANS(x, power, /inverse)
PM, y, Title = 'Inverse Transformed Data'

Inverse Transformed Data
1.00000
2.00000
3.00000
4.00000
5.00000
5.50000
6.50000
7.50000
8.00000
10.0000

```

Errors

Fatal Errors

STAT_ILLEGAL_SHIFT— $S = \#$ and the smallest element of z is $z(\#) = \#$. S plus $z(\#) = \#$. $S + z(i)$ must be greater than 0 for $i = 1, \dots, N\_ELEMENTS(z)$.
 $N\_ELEMENTS(z) = \#$.

STAT_BCTR_CONTAINS_NAN—One or more elements of z is equal to NaN (Not a number). No missing values are allowed. The smallest index of an element of z that is equal to NaN is $\#$.

STAT_BCTR_F_UNDERFLOW—Forward transform. $power = \#$. $S = \#$. The minimum element of z is $z(\#) = \#$. $(z(\#) + S) ^ power$ will underflow.

STAT_BCTR_F_OVERFLOW—Forward transformation. $power = \#$. $S = \#$. The maximum element of z is $z(\#) = \#$. $(z(\#) + S) ^ power$ will overflow.

STAT_BCTR_I_UNDERFLOW—Inverse transformation. $power = \#$. The minimum element of z is $z(\#) = \#$. $\exp(z(\#))$ will underflow.

STAT_BCTR_I_OVERFLOW—Inverse transformation. $power = \#$. The maximum element of $z(\#) = \#$. $\exp(z(\#))$ will overflow.

STAT_BCTR_I_ABS_UNDERFLOW—Inverse transformation. *power* = #. The element of *z* with the smallest absolute value is $z(\#) = \#$. $z(\#)^{(1/power)}$ will underflow.

STAT_BCTR_I_ABS_OVERFLOW—Inverse transformation. *power* = #. The element of *z* with the largest absolute value is $z(\#) = \#$. $z(\#)^{(1/power)}$ will overflow.

Version History

6.4	Introduced

IMSL_AUTOCORRELATION

The IMSL_AUTOCORRELATION function computes the sample autocorrelation function of a stationary time series.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_AUTOCORRELATION(x, lagmax [, ACV=variable] [ /DOUBLE]
    [, SE_OPTION=value] [, SEAC=variable] [, XMEAN_IN=value]
    [, XMEAN_OUT=variable])
```

Return Value

One-dimensional array of length $lagmax + 1$ containing the auto-correlations of the time series x . The 0-th element of this array is 1. The k -th element of this array contains the autocorrelation of lag k where $k = 1, \dots, lagmax$.

Arguments

lagmax

Scalar integer containing the maximum lag of autocovariance, auto-correlations, and standard errors of auto-correlations to be computed. $lagmax$ must be greater than or equal to 1 and less than $N\_ELEMENTS(x)$.

x

One-dimensional array containing the time series. $N\_ELEMENTS(x)$ must be greater than or equal to 2.

Keywords

ACV

Named variable into which an array of length $lagmax + 1$ containing the variance and auto-covariances of the time series x is stored. The 0-th element of this array is the

variance of the time series x . The k -th element contains the autocovariance of lag k where $k = 1, \dots, \text{lagmax}$.

DOUBLE

If present and nonzero, double precision is used.

SE_OPTION

Method of computation for standard errors of the auto-correlations. Keywords *Se_Option* and *Seac* must be used together.

- 1—Compute the standard errors of autocorrelation using Barlett's formula.
- 2—Compute the standard errors of autocorrelation using Moran's formula.

SEAC

Named variable into which an array of length *lagmax* containing the standard errors of the auto-correlations of the time series x is stored. Keywords *Seac* and *Se_Option* must be used together.

XMEAN_IN

The estimate of the mean of the time series x .

XMEAN_OUT

Named variable into which the estimate of the mean of the time series x is stored.

Discussion

The IMSL_AUTOCORRELATION function estimates the autocorrelation function of a stationary time series given a sample of $n = \text{N\_ELEMENTS}(x)$ observations $\{X_t\}$ for $t = 1, 2, \dots, n$.

Let:

$$\mu = x\_mean$$

be the estimate of the mean μ of the time series $\{X_t\}$ where:

$$\hat{\mu} = \begin{cases} \mu, & \mu \text{ known} \\ \frac{1}{n} \sum_{t=1}^n X_t & \mu \text{ unknown} \end{cases}$$

The autocovariance function $\sigma(k)$ is estimated by:

$$\hat{\sigma}(k) = \frac{1}{n} \sum_{t=1}^{n-k} (X_t - \hat{\mu})(X_{t+k} - \hat{\mu}), \quad k = 0, 1, \dots, K$$

where $K = \text{lagmax}$. Note that:

$$\hat{\sigma}(0)$$

is an estimate of the sample variance. The autocorrelation function $\rho(k)$ is estimated by:

$$\hat{\rho}(k) = \frac{\hat{\sigma}(k)}{\hat{\sigma}(0)}, \quad k = 0, 1, \dots, K$$

Note that:

$$\hat{\rho}(0) \equiv 1$$

by definition.

The standard errors of the sample auto-correlations may be optionally computed according to the keyword *Se_Option* for the output keyword *Seac*. One method (Bartlett 1946) is based on a general asymptotic expression for the variance of the sample autocorrelation coefficient of a stationary time series with independent, identically distributed normal errors. The theoretical formula is:

$$\text{var}\{\hat{\rho}(k)\} = \frac{1}{n} \sum_{i=-\infty}^{\infty} [\rho^2(i) + \rho(i-k)\rho(i+k) - 4\rho(i)\rho(k)\rho(i-k) + 2\rho^2(i)\rho^2(k)]$$

where:

$$\hat{\rho}(k)$$

assumes μ is unknown. For computational purposes, the auto-correlations $\rho(k)$ are replaced by their estimates:

$$\hat{\rho}(k)$$

for $|k| \leq K$, and the limits of summation are bounded because of the assumption that $\rho(k) = 0$ for all k such that $|k| > K$.

A second method (Moran 1947) utilizes an exact formula for the variance of the sample autocorrelation coefficient of a random process with independent, identically distributed normal errors. The theoretical formula is:

$$\text{var}\{\hat{\rho}(k)\} = \frac{n-k}{n(n+2)}$$

where μ is assumed to be equal to zero. Note that this formula does not depend on the autocorrelation function.

Example

Consider the Wolfer Sunspot Data (Anderson 1971, page 660) consisting of the number of sunspots observed each year from 1749 through 1924. The data set for this example consists of the number of sunspots observed from 1770 through 1869. The `IMSL_AUTOCORRELATION` function computes the estimated auto-covariances, estimated auto-correlations, and estimated standard errors of the auto-correlations.

```
.RUN
PRO print_results, xm, acv, result, seac
  PRINT, 'Mean =', xm
  PRINT, 'Variance =', acv(0)
  PRINT, '      Lag      ACV      AC      SEAC'
  PRINT, '      0', acv(0), result(0)
  FOR j = 1, 20 DO $
    PRINT, j, acv(j), result(j), seac(j - 1)
  END
```

```
lagmax = 20
data = IMSL_STATDATA(2)
x = data(21:120,1)
result = IMSL_AUTOCORRELATION(x, lagmax, ACV = acv, $
  SE_OPTION = 1, SEAC = seac, XMEAN_OUT = xm)
print_results, xm, acv, result, seac
```

```
Mean =      46.9760
Variance =    1382.91
      Lag      ACV      AC      SEAC
0      1382.91      1.00000
1      1115.03      0.806293    0.0347834
2      592.004      0.428087    0.0962420
3      95.2974      0.0689109    0.156783
4     -235.952     -0.170620    0.205767
5     -370.011     -0.267560    0.230956
6     -294.255     -0.212780    0.228995
7     -60.4423     -0.0437067    0.208622
8      227.633      0.164604    0.178476
9      458.381      0.331462    0.145727
10     567.841      0.410613    0.134406
11     546.122      0.394908    0.150676
12     398.937      0.288477    0.174348
13     197.757      0.143001    0.190619
14      26.8911      0.0194453    0.195490
15     -77.2807     -0.0558828    0.195893
```

16	-143.733	-0.103935	0.196285
17	-202.048	-0.146104	0.196021
18	-245.372	-0.177432	0.198716
19	-230.816	-0.166906	0.205359
20	-142.879	-0.103318	0.209387

Version History

6.4	Introduced

IMSL_PARTIAL_AC

The IMSL_PARTIAL_AC function computes the sample partial autocorrelation function of a stationary time series.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_PARTIAL_AC(*cf* [, /DOUBLE])

Return Value

One-dimensional array containing the partial auto-correlations of the time series *x*.

Arguments

cf

One-dimensional array containing the auto-correlations of the time series *x*.

Keywords

DOUBLE

If present and nonzero, double precision is used.

Discussion

IMSL_PARTIAL_AC estimates the partial auto-correlations of a stationary time series given the $K = (N\_ELEMENTS(cf) - 1)$ sample auto-correlations:

$$\hat{\rho}(k)$$

for $k = 0, 1, \dots, K$. Consider the AR(k) process defined by:

$$X_t = \phi_{k1}X_{t-1} + \phi_{k2}X_{t-2} + \dots + \phi_{kk}X_{t-k} + A_t$$

where ϕ_{kj} denotes the j -th coefficient in the process. The set of estimates:

$$\{\hat{\phi}_{kk}\}$$

for $k = 1, \dots, K$ is the sample partial autocorrelation function. The autoregressive parameters:

$$\{\hat{\phi}_{kj}\}$$

for $j = 1, \dots, k$ are approximated by Yule-Walker estimates for successive $AR(k)$ models where $k = 1, \dots, K$. Based on the sample Yule-Walker equations:

$$\hat{\rho}(j) = \hat{\phi}_{k1}\hat{\rho}(j-1) + \hat{\phi}_{k2}\hat{\rho}(j-2) + \dots + \hat{\phi}_{kk}\hat{\rho}(j-k) \quad j = 1, 2, \dots, k$$

a recursive relationship for $k = 1, \dots, K$ was developed by Durbin (1960). The equations are given by:

$$\hat{\phi}_{kk} = \begin{cases} \hat{\rho}(1) & k = 1 \\ \frac{\hat{\rho}(k) - \sum_{j=1}^{k-1} \hat{\phi}_{k-1,j} \hat{\rho}(k-j)}{1 - \sum_{j=1}^{k-1} \hat{\phi}_{k-1,j} \hat{\rho}(j)} & k = 2, \dots, K \end{cases}$$

and:

$$\hat{\phi}_{kj} = \begin{cases} \hat{\phi}_{k-1,j} - \hat{\phi}_{kk} \hat{\phi}_{k-1,k-j} & j = 1, 2, \dots, k-1 \\ \hat{\phi}_{kk} & j = k \end{cases}$$

This procedure is sensitive to rounding error and should not be used if the parameters are near the non-stationary boundary. A possible alternative would be to estimate $\{\phi_{kk}\}$ for successive $AR(k)$ models using least or maximum likelihood. Based on the hypothesis that the true process is $AR(p)$, Box and Jenkins (1976, page 65) note:

$$\text{var}\{\hat{\phi}_{kk}\} = \frac{1}{n} \quad k \geq p + 1$$

See Box and Jenkins (1976, pages 82–84) for more information concerning the partial autocorrelation function.

Example

Consider the Wolfer Sunspot Data (Anderson 1971, page 660) consisting of the number of sunspots observed each year from 1749 through 1924. The data set for this

example consists of the number of sunspots observed from 1770 through 1869. Routine `IMSL_PARTIAL_AC` is used to compute the estimated partial auto-correlations.

```
data = IMSL_STATDATA(2)
x = data(21:120,1)
result = IMSL_AUTOCORRELATION(x, 20)
partial = IMSL_PARTIAL_AC(result)
PRINT, 'LAG      PACF'
FOR i = 0, 19 DO PM, i + 1, partial(i), FORMAT = '(I2, F11.3)'
```

```
LAG      PACF
 1      0.806
 2     -0.635
 3      0.078
 4     -0.059
 5     -0.001
 6      0.172
 7      0.109
 8      0.110
 9      0.079
10      0.079
11      0.069
12     -0.038
13      0.081
14      0.033
15     -0.035
16     -0.131
17     -0.155
18     -0.119
19     -0.016
20     -0.004
```

Version History

6.4	Introduced

IMSL_LACK_OF_FIT

The IMSL_LACK_OF_FIT function performs lack-of-fit test for a univariate time series or transfer function given the appropriate correlation function.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_LACK_OF_FIT(*nobs*, *cf*, *npfree* [, /DOUBLE] [, LAGMIN=*value*])

Return Value

One-dimensional array of length 2 with the test statistic, *Q*, and its *p*-value, *p*. Under the null hypothesis, *Q* has an approximate chi-squared distribution with $\text{lagmax} - \text{Lagmin} + 1 - \text{npfree}$ degrees of freedom.

Arguments

cf

One-dimensional array containing the correlation function.

nobs

Number of observations of the stationary time series.

npfree

Number of free parameters in the formulation of the time series model. *npfree* must be greater than or equal to zero and less than lagmax where $\text{lagmax} = (\text{N\_ELEMENTS}(\text{cf}) - 1)$. Woodfield (1990) recommends $\text{npfree} = p + q$.

Keywords

DOUBLE

If present and nonzero, double precision is used.

LAGMIN

Minimum lag of the correlation function. *Lagmin* corresponds to the lower bound of summation in the lack of fit test statistic. Default: *Lagmin* = 1.

Discussion

Routine IMSL_LACK_OF_FIT may be used to diagnose lack of fit in both IMSL_ARMA and transfer function models. Table 20-2 shows typical arguments for these situations:

Model	LAGMIN	LAGMAX	NPFREE
IMSL_ARMA (<i>p</i> , <i>q</i>)	1	$\sqrt{NOBS}$	<i>p</i> + <i>q</i>
Transfer function	0	$\sqrt{NOBS}$	<i>r</i> + <i>s</i>

Table 20-2: Max, Min, and Free Arguments

The IMSL_LACK_OF_FIT function performs a portmanteau lack of fit test for a time series or transfer function containing *n* observations given the appropriate sample correlation function:

$$\hat{\rho}(k)$$

for $k = L, L + 1, \dots, K$ where $L = \text{Lagmin}$ and $K = \text{lagmax}$.

The basic form of the test statistic *Q* is:

$$Q = n(n + 2) \sum_{k=L}^K (n - k)^{-1} \hat{\rho}(k)$$

with $L = 1$ if:

$$\hat{\rho}(k)$$

is an autocorrelation function. Given that the model is adequate, *Q* has a chi-squared distribution with $K - L + 1 - m$ degrees of freedom where $m = \text{npfree}$ is the number of parameters estimated in the model. If the mean of the time series is estimated, Woodfield (1990) recommends not including this in the count of the parameters estimated in the model. Thus, for an IMSL_ARMA(*p*, *q*) model set $\text{npfree} = p + q$

regardless of whether the mean is estimated or not. The original derivation for time series models is due to Box and Pierce (1970) with the above modified version discussed by Ljung and Box (1978). The extension of the test to transfer function models is discussed by Box and Jenkins (1976, pages 394–395).

Example

Consider the Wölfer Sunspot Data (Anderson 1971, page 660) consisting of the number of sunspots observed each year from 1749 through 1924. The data set for this example consists of the number of sunspots observed from 1770 through 1869. An `IMSL_ARMA(2,1)` with nonzero mean is fitted using the “[IMSL_ARMA](#)” on page 913. The auto-correlations of the residuals are estimated using the “[IMSL_AUTOCORRELATION](#)” on page 940. A portmanteau lack of fit test is computed using 10 lags with `IMSL_LACK_OF_FIT`.

The warning message from `IMSL_ARMA` in the output can be ignored. (See the example for routine `IMSL_ARMA` for a full explanation of the warning message.)

```
p = 2
q = 1
tc = 0.125
lagmax = 10
npfree = 4
; Get sunspot data for 1770 through 1869, store it in x()
data = IMSL_STATDATA(2)
x = data(21:120,1)
; Get residuals for IMSL_ARMA(2, 1) for autocorrelation/lack
; of fit
params = IMSL_ARMA(x, p, q, /Lsq, TOL_CONVERGENCE = tc, $
    RESIDUAL = r)

; Get autocorrelations from residuals for lack of fit test
; NOTE: number of observations is equal to number of residuals
corrs = IMSL_AUTOCORRELATION(r, lagmax)
; Get lack of fit test statistic and p-value
; NOTE: number of observations is equal to original number of data
result = IMSL_LACK_OF_FIT(N_ELEMENTS(x), corrs, npfree)
; Print parameter estimates, test statistic, and p_value
; NOTE: Test Statistic Q follows a Chi-squared dist.
PRINT, 'Lack of Fit Statistic (Q) =', result(0), $
    FORMAT = '(A28, F8.3)'
PRINT, 'P-value (PVALUE) =', result(1), FORMAT = '(A28, F8.4)'

Lack of Fit Statistic (Q) = 14.572
P-value (PVALUE) = 0.9761
```

Version History

6.4	Introduced

IMSL_GARCH

The IMSL_GARCH function computes estimates of the parameters of a $\text{IMSL\_GARCH}(p,q)$ model.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_GARCH(p, q, y, xguess [, AIC=variable] [, /DOUBLE]
[, LOG_LIKELIHOOD=variable] [, MAX_SIGMA=value] [, VAR=variable] )
```

Return Value

One-dimensional array of length $p + q + 1$ containing the estimated values of sigma squared, the AR parameters, and the MA parameters.

Arguments

p

Number of autoregressive (AR) parameters.

q

Number of moving average (MA) parameters.

xguess

One-dimensional array of length $p + q + 1$ containing the initial values for the parameter array x .

y

One-dimensional array containing the observed time series data.

Keywords

AIC

Named variable into which the value of Akaike Information Criterion evaluated at the estimated parameter array x is stored.

DOUBLE

If present and nonzero, double precision is used.

LOG_LIKELIHOOD

Named variable into which the value of Log-likelihood function evaluated at the estimated parameter array x is stored.

MAX_SIGMA

Value of the upperbound on the first element (sigma) of the array of returned estimated coefficients. Default: *Max_Sigma* = 10.

VAR

Named variable into which an array of size $(p + q + 1)$ by $(p + q + 1)$ containing the variance-covariance matrix is stored.

Discussion

The Generalized Autoregressive Conditional Heteroskedastic (IMSL_GARCH) model is defined as:

$$y_t = z_t \sigma_t$$

$$\sigma_t^2 = \sigma^2 + \sum_{i=1}^p \beta_i \sigma_{t-i}^2 + \sum_{i=1}^q \alpha_i y_{t-i}^2,$$

where z_t 's are independent and identically distributed standard normal random variables:

$$\sigma > 0, \beta_i \geq 0, \alpha_i \geq 0 \text{ and}$$

$$\sum_{i=1}^p \beta_i + \sum_{i=1}^q \alpha_i < 1.$$

The above model is denoted as IMSL_GARCH(p, q). The p is the autoregressive lag and the q is the moving average lag. When $\beta_i = 0$, $i = 1, 2, \dots, p$, the above model

reduces to ARCH(q) which was proposed by Engle (1982). The non-negativity conditions on the parameters implied a nonnegative variance and the condition on the sum of the β_i 's and α_i 's is required for wide sense stationarity.

In the empirical analysis of observed data, IMSL_GARCH(1,1) or IMSL_GARCH(1,2) models have often found to appropriately account for conditional heteroskedasticity (Palm 1996). This finding is similar to linear time series analysis based on IMSL_ARMA models.

It is important to notice that for the above models positive and negative past values have a symmetric impact on the conditional variance. In practice, many series may have strong asymmetric influence on the conditional variance. To take into account this phenomena, Nelson (1991) put forward Exponential IMSL_GARCH (EGARCH). Lai (1998) proposed and studied some properties of a general class of models that extended linear relationship of the conditional variance in ARCH and IMSL_GARCH into nonlinear fashion.

The maximal likelihood method is used in estimating the parameters in IMSL_GARCH(p,q). The log-likelihood of the model for the observed series $\{Y_t\}$ with length m is:

$$\log(L) = \frac{m}{2} \log(2\pi) - \frac{1}{2} \sum_{t=1}^m y_t^2 / \sigma_t^2 - \frac{1}{2} \sum_{t=1}^m \log \sigma_t^2,$$

$$\text{where } \sigma_t^2 = \sigma^2 + \sum_{i=1}^p \beta_i \sigma_{t-i}^2 + \sum_{i=1}^q \alpha_i y_{t-i}^2.$$

In the model, if $q = 0$, the model IMSL_GARCH is singular such that the estimated Hessian matrix H is singular.

The initial values of the parameter array x entered in array $xguess$ must satisfy certain constraints. The first element of $xguess$ refers to sigma and must be greater than zero and less than *Max_Sigma*. The remaining $p + q$ initial values must each be greater than or equal to zero but less than one.

To guarantee stationarity in model fitting:

$$\sum_{i=1}^{p+q} x(i) < 1,$$

is checked internally. The initial values should be selected from the values between zero and one. The Aic is computed by:

$$2 * \log(L) + 2 * (p+q+1)$$

where $\log(L)$ is the value of the log-likelihood function at the estimated parameters.

In fitting the optimal model, the subroutine IMSL_MINCONGEN as well as its associated subroutines are modified to find the maximal likelihood estimates of the parameters in the model. Statistical inferences can be performed outside the routine IMSL_GARCH based on the output of the log-likelihood function (*Log_Likelihood*), the Akaike Information Criterion (*Aic*), and the variance-covariance matrix (*Var*).

Example

The data for this example are generated to follow a IMSL_GARCH(p,q) process by using a random number generation function SGARCH. The data set is analyzed and estimates of sigma, the AR parameters, and the MA parameters are returned. The values of the Log-likelihood function and the Akaike Information Criterion are returned from the output keywords *Log_Likelihood* and *Aic* respectively.

```
.RUN
FUNCTION SGARCH, p, q, m, x
  z = FLTARR(m + 1000)
  y0 = FLTARR(m + 1000)
  sigma = FLTARR(m + 1000)
  z = IMSL_RANDOM(m + 1000, /Normal)
  l = ((p > q) > 1)
  y0(0:l - 1) = z(0:l - 1)*x(0)
  ; Compute the Initial Value Of Sigma
  s3 = 0.0
  IF ((p > q) GE 1) THEN s3 = TOTAL(x(1:p + q))
  sigma(0:l - 1) = x(0)/(1.0 - s3)
  FOR i = 1, (m + 1000 - 1) DO BEGIN
    s1 = 0.0
    s2 = 0.0
    IF (q GE 1) THEN BEGIN
      FOR j = 0, q - 1 DO s1 = s1 + x(j + 1) * $
        (y0(i - j - 1)^2)
    END
    IF (p GE 1) THEN BEGIN
      FOR j = 0, p - 1 DO s2 = s2 + x(q + 1 + j) $
        * sigma(i - j - 1)
    END
    sigma(i) = x(0) + s1 + s2
    y0(i) = z(i)*SQRT(sigma(i))
  END
  ; Discard the first 1000 Simulated Observations
  RETURN, y0(1000:*)
  ; End of function
END

IMSL_RANDOMOPT, Set = 182198625
p = 2
```

```
q = 1
m = 1000
x = [1.3, 0.2, 0.3, 0.4]
xguess = [1.0, 0.1, 0.2, 0.3]
y = SGARCH(p, q, m, x)
result = IMSL_GARCH(p, q, y, xguess, LOG_LIKELIHOOD = a, $
    AIC = aic)
PRINT, 'Sigma estimate is', result(0)
PRINT, 'AR(1) estimate is', result(1)
PRINT, 'AR(2) estimate is', result(2)
PRINT, 'MA(1) estimate is', result(3)
PRINT, 'Log-likelihood function value is', a
PRINT, 'Akaike Information Criterion value is', aic

Sigma estimate is      1.27742
AR(1) estimate is      0.230132
AR(2) estimate is      0.375924
MA(1) estimate is      0.312843
Log-likelihood function value is      -2707.53
Akaike Information Criterion value is      5423.06
```

Version History

6.4	Introduced

IMSL_KALMAN

The IMSL_KALMAN procedure performs Kalman filtering and evaluates the likelihood function for the state-space model.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
IMSL_KALMAN, b, covb, n, ss, alndet [, COVV=array] [, Q_MATRIX=array]
[, R=array] [, T_MATRIX=array] [, TOLERANCE=value] [, V=array]
[, Y=array]
```

Arguments

alndet

Named variable containing the natural log of the product of the nonzero eigenvalues of P where $P * \sigma^2$ is the variance-covariance matrix of the observations. Although *alndet* is computed, IMSL_KALMAN avoids the explicit computation of P . *alndet* must be initialized to zero before the first call to IMSL_KALMAN. In the usual case when P is non-singular, *alndet* is the natural log of the determinant of P .

b

One dimensional array of containing the estimated state vector. The input is the estimated state vector at time k given the observations through time $k - 1$. The output is the estimated state vector at time $k + 1$ given the observations through time k . On the first call to IMSL_KALMAN, the input b must be the prior mean of the state vector at time.

covb

Two dimensional array of size $N\_ELEMENTS(b)$ by $N\_ELEMENTS(b)$ such that $covb * \sigma^2$ is the mean squared error matrix for b . Before the first call to IMSL_KALMAN, $covb * \sigma^2$ must equal the variance-covariance matrix of the state vector.

n

Named variable containing the rank of the variance-covariance matrix for all the observations. n must be initialized to zero before the first call to IMSL_KALMAN. In the usual case when the variance-covariance matrix is non-singular, n equals the sum of the N_ELEMENTS(Y) from the invocations to IMSL_KALMAN. See the keyword section below for the definition of Y .

ss

Named variable containing the generalized sum of squares. ss must be initialized to zero before the first call to IMSL_KALMAN. The estimate of σ^2 is given by:

$$\frac{ss}{n}$$

Keywords**COVV**

Two dimensional array if size N_ELEMENTS(Y) by N_ELEMENTS(Y) containing a matrix such that $Covv * \sigma^2$ is the variance-covariance matrix of v .

Q_MATRIX

Two dimensional array if size N_ELEMENTS(b) by N_ELEMENTS(b) matrix such that $Q\_matrix * \sigma^2$ is the variance-covariance matrix of the error vector in the state equation. Default: There is no error term in the state equation

R

Two dimensional array if size N_ELEMENTS(Y) by N_ELEMENTS(Y) containing the matrix such that $R * \sigma^2$ is the variance-covariance matrix of errors in the observation equation. Keywords Y , Z and R indicate an update step and must be used together.

T_MATRIX

Two dimensional array if size N_ELEMENTS(b) by N_ELEMENTS(b) containing the transition matrix in the state equation. Default: T_matrix = identity matrix

TOLERANCE

Tolerance used in determining linear dependence. Default: $Tolerance = 100 * eps$ where eps is machine precision.

V

One dimensional array of length N_ELEMENTS(Y) containing the one-step-ahead prediction error.

Y

One dimensional array containing the observations. Keywords Y, Z and R indicate an update step and must be used together

Discussion

Routine IMSL_KALMAN is based on a recursive algorithm given by Kalman (1960), which has come to be known as the Kalman filter. The underlying model is known as the state-space model. The model is specified stage by stage where the stages generally correspond to time points at which the observations become available. The routine IMSL_KALMAN avoids many of the computations and storage requirements that would be necessary if one were to process all the data at the end of each stage in order to estimate the state vector. This is accomplished by using previous computations and retaining in storage only those items essential for processing of future observations.

The notation used here follows that of Sallas and Harville (1981). Let y_k (input in keyword Y) be the $n_k \times 1$ vector of observations that become available at time k . The subscript k is used here rather than t , which is more customary in time series, to emphasize that the model is expressed in stages $k = 1, 2, \dots$ and that these stages need not correspond to equally spaced time points. In fact, they need not correspond to time points of any kind. The observation equation for the state-space model is:

$$y_k = Z_k b_k + e_k \quad k = 1, 2, \dots$$

Here, Z_k is an $n_k \times q$ known matrix and b_k is the $q \times 1$ state vector. The state vector b_k is allowed to change with time in accordance with the state equation:

$$b_{k+1} = T_{k+1} b_k + w_{k+1} \quad k = 1, 2, \dots$$

starting with $b_1 = \mu_1 + w_1$.

The change in the state vector from time k to $k + 1$ is explained in part by the transition matrix T_{k+1} (the identity matrix by default, or optionally input using keyword *T_matrix*), which is assumed known. It is assumed that the q -dimensional w_k s ($k = 1, 2, \dots, K$) are independently distributed multivariate normal with mean vector 0 and variance-covariance matrix $\sigma^2 Q_k$, that the n_k -dimensional e_k s ($k = 1, 2, \dots, K$) are independently distributed multivariate normal with mean vector 0 and variance-covariance matrix $\sigma^2 R_k$, and that the w_k s and e_k s are independent of each

other. Here, μ_1 is the mean of b_1 and is assumed known, σ^2 is an unknown positive scalar. Q_{k+1} (input in Q) and R_k (input in keyword R) are assumed known.

Denote the estimator of the realization of the state vector b_k given the observations $y_1, y_2, \dots, y_j$ by:

$$\hat{\beta}_{k|j}$$

By definition, the mean squared error matrix for:

$$\hat{\beta}_{k|j}$$

is:

$$\hat{\sigma}^2 C_{k|j} = E(\hat{\beta}_{k|j} - b_k)(\hat{\beta}_{k|j} - b_k)^T$$

At the time of the k -th invocation, we have:

$$\hat{\beta}_{k|k-1}$$

and:

$$C_{k|k-1}$$

which were computed from the $(k-1)$ -st invocation, input in b and $covb$, respectively. During the k -th invocation, routine IMSL_KALMAN computes the filtered estimate:

$$\hat{\beta}_{k|k}$$

along with $C_{k|k}$. These quantities are given by the update equations:

$$\hat{\beta}_{k|k} = \hat{\beta}_{k|k-1} + C_{k|k-1} Z_k^T H_k^{-1} v_k$$

$$C_{k|k} = C_{k|k-1} - C_{k|k-1} Z_k^T H_k^{-1} Z_k C_{k|k-1}$$

where:

$$v_k = y_k - Z_k \hat{\beta}_{k|k-1}$$

and where:

$$H_k = R_k + Z_k C_{k|k-1} Z_k^T$$

Here, v_k (stored in v) is the one-step-ahead prediction error, and $\sigma^2 H_k$ is the variance-covariance matrix for v_k . H_k is stored in $covv$. The “start-up values” needed on the first invocation of IMSL_KALMAN are:

$$\hat{\beta}_{1|0} = \mu_1$$

and $C_{1|0} = Q_1$ input via b and $covb$, respectively. Computations for the k -th invocation are completed by IMSL_KALMAN computing the one-step-ahead estimate:

$$\hat{\beta}_{k+1|k}$$

along with $C_{k+1|k}$ given by the prediction equations:

$$\hat{\beta}_{k+1|k} = T_{k+1} \hat{\beta}_{k|k}$$

$$C_{k+1|k} = T_{k+1} C_{k|k} T_{k+1}^T + Q_{k+1}$$

If both the filtered estimates and one-step-ahead estimates are needed at each time point, IMSL_KALMAN can be invoked twice for each time point—first without *T_matrix* and *Q_matrix* to produce:

$$\hat{\beta}_{k|k}$$

and $C_{k|k}$, and second without keywords *Y*, *Z*, and *R* to produce:

$$\hat{\beta}_{k+1|k}$$

and $C_{k+1|k}$ (Without *T_matrix* and *Q_matrix*, prediction equations are skipped. Without keywords *Y*, *Z*, and *R*, update equations are skipped.).

Often, one desires the estimate of the state vector more than one-step-ahead, i.e., an estimate of:

$$\hat{\beta}_{k|j}$$

is needed where $k > j + 1$. At time j , IMSL_KALMAN is invoked with keywords *Y*, *Z*, and *R* to compute:

$$\hat{\beta}_{j+1|j}$$

Subsequent invocations of IMSL_KALMAN without *Y*, *Z*, and *R* can compute:

$$\hat{\beta}_{j+2|j}, \hat{\beta}_{j+3|j}, \dots, \hat{\beta}_{k|j}$$

Computations for:

$$\hat{\beta}_{k|j}$$

and $C_{k|j}$ assume the variance-covariance matrices of the errors in the observation equation and state equation are known up to an unknown positive scalar multiplier, σ^2 . The maximum likelihood estimate of σ^2 based on the observations $y_1, y_2, \dots, y_m$, is given by:

$$\hat{\sigma}^2 = (SS)/N$$

where:

$$N = \sum_{k=1}^m n_k \text{ and } SS = \sum_{k=1}^m v_k^T H_k^{-1} v_k$$

N and SS are the input/output arguments n and ss .

If σ^2 is known, the R_k s and Q_k s can be input as the variance-covariance matrices exactly. The earlier discussion is then simplified by letting $\sigma^2 = 1$.

In practice, the matrices T_k , Q_k , and R_k are generally not completely known. They may be known functions of an unknown parameter vector θ . In this case, IMSL_KALMAN can be used in conjunction with an optimization program (see [IMSL_FMINV](#)) to obtain a maximum likelihood estimate of θ . The natural logarithm of the likelihood function for $y_1, y_2, \dots, y_m$ differs by no more than an additive constant from:

$$L_c(\theta, \sigma^2; y_1, y_2, \dots, y_m) = -\frac{1}{2}N \ln \sigma^2 - \frac{1}{2} \sum_{k=1}^m \ln[\det(H_k)] - \frac{1}{2} \sigma^2 \sum_{k=1}^m v_k^T H_k^{-1} v_k$$

(Harvey 1981, page 14, equation 2.21).

Here:

$$\sum_{k=1}^m \ln[\det(H_k)]$$

(stored in *alndet*) is the natural logarithm of the determinant of V where $\sigma^2 V$ is the variance-covariance matrix of the observations.

Minimization of $-2L(\theta, \sigma^2; y_1, y_2, \dots, y_m)$ over all θ and σ^2 produces maximum likelihood estimates. Equivalently, minimization of $-2L_c(\theta; y_1, y_2, \dots, y_m)$ where:

$$L_c(\theta; y_1, y_2, \dots, y_m) = -\frac{1}{2}N \ln\left(\frac{SS}{N}\right) - \frac{1}{2} \sum_{k=1}^m \ln[\det(H_k)]$$

produces maximum likelihood estimates:

$$\hat{\theta} \text{ and } \hat{\sigma}^2 = SS/N$$

The minimization of $-2L_c(\theta; y_1, y_2, \dots, y_m)$ instead of $-2L(\theta, \sigma^2; y_1, y_2, \dots, y_m)$, reduces the dimension of the minimization problem by one. The two optimization problems are equivalent since:

$$\hat{\sigma}^2(\theta) = SS(\theta)/N$$

minimizes $-2L(\theta, \sigma^2; y_1, y_2, \dots, y_m)$ for all θ , consequently:

$$\hat{\sigma}^2(\theta)$$

can be substituted for σ^2 in $L(\theta, \sigma^2; y_1, y_2, \dots, y_m)$ to give a function that differs by no more than an additive constant from $L_c(\theta; y_1, y_2, \dots, y_m)$.

The earlier discussion assumed H_k to be non-singular. If H_k is singular, a modification for singular distributions described by Rao (1973, pages 527–528) is used. The changes in the preceding discussion are as follows:

1. Replace H_k^{-1} by a generalized inverse.
2. Replace $\det(H_k)$ by the product of the nonzero eigenvalues of H_k .
3. Replace N by:

$$\sum_{k=1}^m \text{rank}(H_k)$$

Maximum likelihood estimation of parameters in the Kalman filter is discussed by Sallas and Harville (1988) and Harvey (1981, pages 111–113).

Example

Routine IMSL_KALMAN is used to compute the filtered estimates and one-step-ahead estimates for a scalar problem discussed by Harvey (1981, pages 116–117). The observation equation and state equation are given by:

$$y_k = b_k + e_k$$

$$b_{k+1} = b_k + w_{k+1} \quad k = 1, 2, 3, 4$$

where the e_k s are identically and independently distributed normal with mean 0 and variance σ^2 , the w_k s are identically and independently distributed normal with mean 0 and variance $4\sigma^2$, and b_1 is distributed normal with mean 4 and variance $16\sigma^2$. Two invocations of IMSL_KALMAN are needed for each time point in order to compute the filtered estimate and the one-step-ahead estimate. The first invocation does not use the keywords *T_matrix* and *Q_matrix* so that the prediction equations are skipped in the computations. The update equations are skipped in the computations in the second invocation.

This example also computes the one-step-ahead prediction errors. Harvey (1981, page 117) contains a misprint for the value v_4 that he gives as 1.197. The correct value of $v_4 = 1.003$ is computed by IMSL_KALMAN.

Note that this example is in the form of an IDL Advanced Math and Stats procedure, with the output following the procedure.

```

.RUN
PRO EX_KALMAN
  z = 1
  r = 1
  q = 4
  t = 1

  b = 4
  covb = 16

  ydata = [4.4, 4, 3.5, 4.6]

  n = 0
  ss = 0
  alndet = 0
  FORMAT = '(2I4, 2F8.3, I4, 4F8.3)'
  PRINT, '    k    j        b        covb    n    ss        alndet    v
covv'
  FOR i = 0, 3 DO BEGIN
    y = ydata(i)
    ; Update
    IMSL_KALMAN, b, covb, n, ss, alndet, Y = y, Z = Z, R = r, $
      v = v, covv = covv
    PRINT, i, i, b, covb, n, ss, alndet, v, covv, $
      FORMAT = format

    ; Predict
    IMSL_KALMAN, b, covb, n, ss, alndet, t_matrix = t, q = q
    PRINT, i+1, i, b, covb, n, ss, alndet, v, covv, $
      FORMAT = format
  END

END

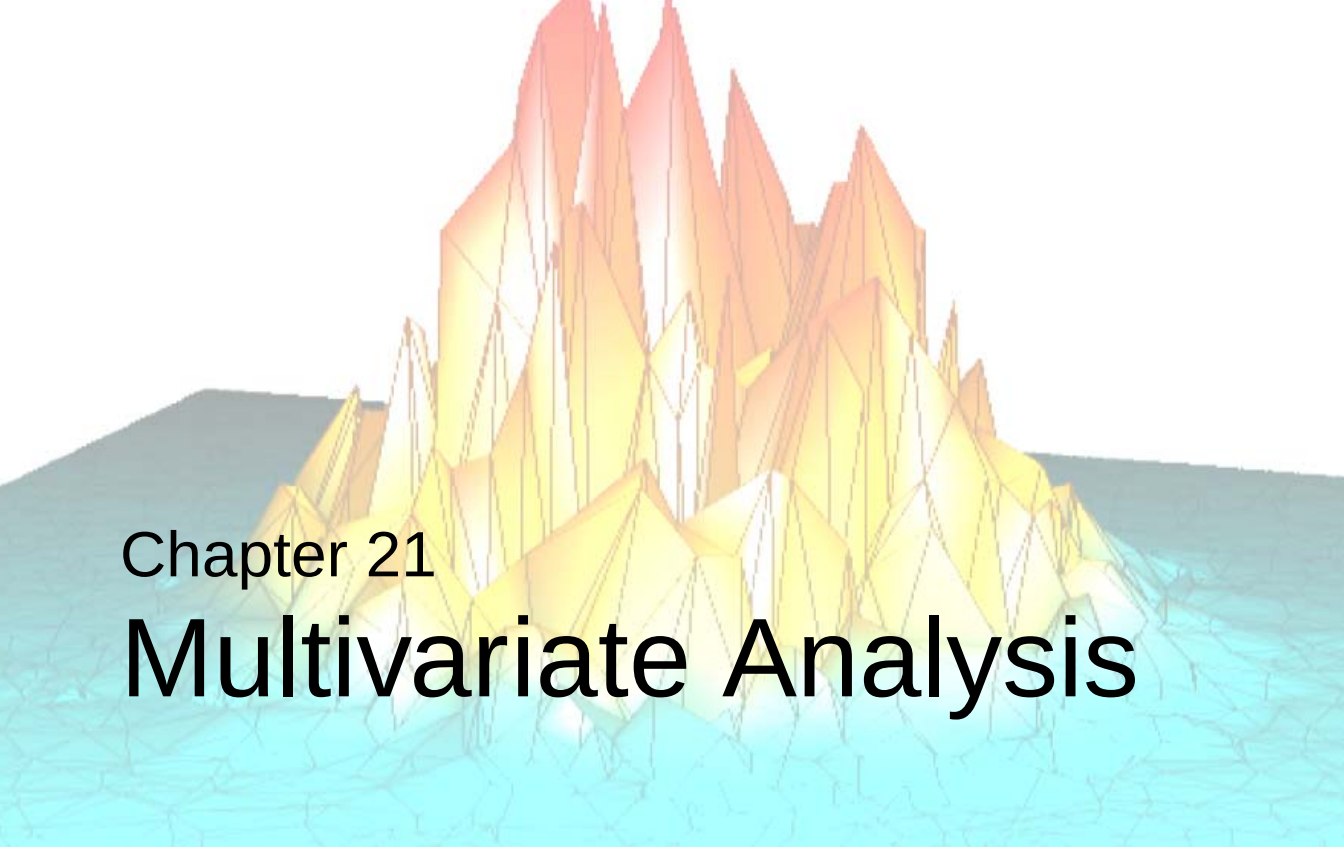
```

Output

k	j	b	covb	n	ss	alndet	v	covv
0	0	4.376	0.941	1	0.009	2.833	0.400	17.000
1	0	4.376	4.941	1	0.009	2.833	0.400	17.000
1	1	4.063	0.832	2	0.033	4.615	-0.376	5.941
2	1	4.063	4.832	2	0.033	4.615	-0.376	5.941
2	2	3.597	0.829	3	0.088	6.378	-0.563	5.832
3	2	3.597	4.829	3	0.088	6.378	-0.563	5.832
3	3	4.428	0.828	4	0.260	8.141	1.003	5.829
4	3	4.428	4.828	4	0.260	8.141	1.003	5.829

Version History

6.4	Introduced



Chapter 21

Multivariate Analysis

This section contains the following topics:

Overview: Multivariate Analysis	968	Multivariate Analysis Routines	970

Overview: Multivariate Analysis

This section describes cluster analysis, principal components, and factor analysis.

Cluster Analysis

The `IMSL_K_MEANS` function performs a K -means cluster analysis. Basic K -means clustering attempts to find a clustering that minimizes the within-cluster sums-of-squares. In this method of clustering the data, matrix X is grouped so that each observation (row in X) is assigned to one of a fixed number, K , of clusters. The sum of the squared difference of each observation about its assigned cluster's mean is used as the criterion for assignment. In the basic algorithm, observations are transferred from one cluster or another when doing so decreases the within-cluster sums-of-squared differences. When no transfer occurs in a pass through the entire data set, the algorithm stops. The `IMSL_K_MEANS` function is one implementation of the basic algorithm.

The usual course of events in K -means cluster analysis is to use `IMSL_K_MEANS` to obtain the optimal clustering. The clustering is then evaluated by functions described in [Chapter 13, “Basic Statistics”](#), and other chapters in this manual. Often, K -means clustering with more than one value of K is performed, and the value of K that best fits the data is used.

Clustering can be performed either on observations or variables. The discussion of `IMSL_K_MEANS` assumes the clustering is to be performed on the observations, which correspond to the rows of the input data matrix. If variables, rather than observations, are to be clustered, the data matrix should first be transposed. In the documentation for `IMSL_K_MEANS`, the words “observation” and “variable” can be interchanged.

Principal Components

The idea in principal components is to find a small number of linear combinations of the original variables that maximize the variance accounted for in the original data. This amounts to an eigensystem analysis of the covariance (or correlation) matrix. In addition to the eigensystem analysis, `IMSL_PRINC_COMP` computes standard errors for the eigenvalues. Correlations of the original variables with the principal component scores also are computed.

Factor Analysis

Factor analysis and principal component analysis, while different in assumptions, often serve the same purpose. Unlike principal components in which linear combinations yielding the highest possible variances are obtained, factor analysis generally obtains linear combinations of the observed variables according to a model relating the observed variable to hypothesized underlying factors, plus a random error term called the unique error or uniqueness. In factor analysis, the unique errors associated with each variable are usually assumed to be independent of the factors. Additionally, in the common factor model, the unique errors are assumed to be mutually independent. The factor analysis model is expressed in the following equation:

$$x - \mu = \Lambda f + e$$

where x is the p vector of observed values, μ is the p vector of variable means, Λ is the $p \times k$ matrix of factor loadings, f is the k vector of hypothesized underlying random factors, e is the p vector of hypothesized unique random errors, p is the number of variables in the observed variables, and k is the number of factors.

Because much of the computation in factor analysis was originally done by hand or was expensive on early computers, quick (but “dirty”) algorithms that made the calculations possible were developed. One result is the many factor extraction methods available today. Generally speaking, in the exploratory or model-building phase of a factor analysis, a method of factor extraction that is not computationally intensive (such as principal components, principal factor, or image analysis) is used. If desired, a computationally intensive method is then used to obtain the final factors.

Multivariate Analysis Routines

- [IMSL_K_MEANS](#)—Performs a K -means (centroid) cluster analysis.
- [IMSL_PRINC_COMP](#)—Computes principal components.
- [IMSL_FACTOR_ANALYSIS](#)—Extracts factor-loading estimates.
- [IMSL_DISCR_ANALYSIS](#)—Perform discriminant function analysis.

IMSL_K_MEANS

The IMSL_K_MEANS function performs a K -means (centroid) cluster analysis.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_K_MEANS(x, seeds [, COUNTS_CLUSTER=variable]  
    [, /DOUBLE] [, FREQUENCIES=array] [, ITMAX=value]  
    [, MEANS_CLUSTER=variable] [, SSQ_CLUSTER=variable]  
    [, VAR_COLUMNS=array] [, WEIGHTS=array] )
```

Return Value

The cluster membership for each observation is returned.

Arguments

seeds

Two-dimensional array containing the cluster seeds, i.e., estimates for the cluster centers. The seed value for the j -th variable of the i -th seed should be in *seeds* (i, j).

x

Two-dimensional array containing observations to be clustered. The data value for the i -th observation of the j -th variable should be in *x*(i, j).

Keywords

COUNTS_CLUSTER

Named variable into which an array containing the number of observations in each cluster is stored.

DOUBLE

If present and nonzero, double precision is used.

FREQUENCIES

One-dimensional array containing the frequency of each observation of matrix x .

Default: *Frequencies*(*) = 1

ITMAX

Maximum number of iterations. Default: *Itmax* = 30

MEANS_CLUSTER

Named variable into which a two-dimensional array containing the cluster means is stored.

SSQ_CLUSTER

Named variable into which a one-dimensional array containing the within sum-of-squares for each cluster is stored.

VAR_COLUMNS

One-dimensional array containing the columns of x to be used in computing the metric. Columns are numbered 0, 1, 2, ..., $N\_ELEMENTS(x(0, *))$. Default:

Vars_Columns(*) = 0, 1, 2, ..., $N\_ELEMENTS(x(0, *)) - 1$

WEIGHTS

One-dimensional array containing the weight of each observation of matrix x .

Default: *Weights*(*) = 1

Discussion

The `IMSL_K_MEANS` function is an implementation of Algorithm AS 136 by Hartigan and Wong (1979). This function computes K -means (centroid) Euclidean metric clusters for an input matrix starting with initial estimates of the K -cluster means. The `IMSL_K_MEANS` function allows for missing values coded as NaN (Not a Number) and for weights and frequencies.

Let $p = N\_ELEMENTS(x(0, *))$ be the number of variables to be used in computing the Euclidean distance between observations. The idea in K -means cluster analysis is to find a clustering (or grouping) of the observations so as to minimize the total

within-cluster sums-of-squares. In this case, the total sums-of-squares within each cluster is computed as the sum of the centered sum-of-squares over all non-missing values of each variable. That is:

$$\phi = \sum_{i=1}^K \sum_{j=1}^p \sum_{m=1}^{n_i} f_{v_{im}} w_{v_{im}} \delta_{v_{im},j} (x_{v_{im},j} - \bar{x}_{ij})^2$$

where v_{im} denotes the row index of the m -th observation in the i -th cluster in the matrix X ; n_i is the number of rows of X assigned to group i ; f denotes the frequency of the observation; w denotes its weight; δ is 0 if the j -th variable on observation v_{im} is missing, otherwise δ is 1; and:

$$\bar{x}_{ij}$$

is the average of the non-missing observations for variable j in group i . This method sequentially processes each observation and reassigns it to another cluster if doing so results in a decrease of the total within-cluster sums-of-squares. See Hartigan and Wong (1979) or Hartigan (1975) for details.

Example

This example performs K -means cluster analysis on Fisher's iris data, which is obtained by `IMSL_STATDATA`. The initial cluster seed for each iris type is an observation known to be in the iris type.

```
seeds = MAKE_ARRAY(3,4)
x = IMSL_STATDATA(3)
seeds(0, *) = x(0, 1:4)
seeds(1, *) = x(50, 1:4)
seeds(2, *) = x(100, 1:4)
; Use Columns 1, 2, 3, and 4 of data matrix x, only.
cluster_group = IMSL_K_MEANS(x(*, 1:4), seeds, $
    Means_Cluster = means_cluster, Ssq_Cluster= ssq_cluster, $
    Counts_Cluster = counts_cluster)
FORMAT = '(a, 10i4)'
FOR i = 0, 140, 10 DO BEGIN &$
    PRINT, 'observation: ', i + INDGEN(10)+1, $
    FORMAT = format &$
    PRINT, 'cluster: ', cluster_group(i:i+9), $
    FORMAT = format &$
    PRINT &$
END
; Print cluster membership in groups of 10.
```

observation:	1	2	3	4	5	6	7	8	9	10
cluster	: 1	1	1	1	1	1	1	1	1	1

```

observation: 11 12 13 14 15 16 17 18 19 20
cluster      : 1 1 1 1 1 1 1 1 1 1
observation: 21 22 23 24 25 26 27 28 29 30
cluster      : 1 1 1 1 1 1 1 1 1 1
observation: 31 32 33 34 35 36 37 38 39 40
cluster      : 1 1 1 1 1 1 1 1 1 1
observation: 41 42 43 44 45 46 47 48 49 50
cluster      : 1 1 1 1 1 1 1 1 1 1
observation: 51 52 53 54 55 56 57 58 59 60
cluster      : 2 2 3 2 2 2 2 2 2 2
observation: 61 62 63 64 65 66 67 68 69 70
cluster      : 2 2 2 2 2 2 2 2 2 2
observation: 71 72 73 74 75 76 77 78 79 80
cluster      : 2 2 2 2 2 2 2 3 2 2
observation: 81 82 83 84 85 86 87 88 89 90
cluster      : 2 2 2 2 2 2 2 2 2 2
observation: 91 92 93 94 95 96 97 98 99 100
cluster      : 2 2 2 2 2 2 2 2 2 2
observation: 101 102 103 104 105 106 107 108 109 110
cluster      : 3 2 3 3 3 3 2 3 3 3
observation: 111 112 113 114 115 116 117 118 119 120
cluster      : 3 3 3 2 2 3 3 3 3 2
observation: 121 122 123 124 125 126 127 128 129 130
cluster      : 3 2 3 2 3 3 2 2 3 3
observation: 131 132 133 134 135 136 137 138 139 140
cluster      : 3 3 3 2 3 3 3 3 2 3
observation: 141 142 143 144 145 146 147 148 149 150
cluster      : 3 3 2 3 3 3 2 3 3 2

```

```

PM, [[INDGEN(3) + 1],[means_cluster]], Title = 'Cluster Means:',$
  FORMAT = '(i3, 5x, 4f8.4)'

```

Cluster Means:

```

1      5.0060  3.4280  1.4620  0.2460
2      5.9016  2.7484  4.3935  1.4339
3      6.8500  3.0737  5.7421  2.0711

```

```

PM, [[INDGEN(3) + 1],[ssq_cluster]], $
  Title = 'Cluster Sums of Squares:', FORMAT = '(i3, 5x, f8.4)'

```

Cluster Sums of Squares:

```

1      15.1510
2      39.8210
3      23.8795

```

```

PM, [[INDGEN(3) + 1],[counts_cluster]], Title = $
  'Number of Observations per Cluster:'

```

Number of Observations per Cluster:

1	50
2	62
3	38

Errors

Warning Errors

STAT_NO_CONVERGENCE—Convergence did not occur.

Version History

6.4	Introduced

IMSL_PRINC_COMP

The IMSL_PRINC_COMP function computes principal components.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_PRINC_COMP(covariances [, /COV_MATRIX]  
    [, /CORR_MATRIX] [, CORRELATIONS=variable]  
    [, CUM_PERCENT=variable] [, DF=variable] [, /DOUBLE]  
    [, EIGENVECTORS=variable] [, STDEV=variable] )
```

Return Value

One-dimensional array containing the eigenvalues of *covariances* ordered from largest to smallest.

Arguments

covariances

Two-dimensional square matrix containing the covariance or correlation matrix.

Keywords

COV_MATRIX

If present and nonzero, treats the input matrix *covariances* as a covariance matrix. Keywords *Cov_Matrix* and *Corr_Matrix* cannot be used together. Default: *Cov_Matrix* = 1

CORR_MATRIX

If present and nonzero, treats the input matrix *covariances* as a correlation matrix.

CORRELATIONS

Named variable into which the one-dimensional array of length containing the correlations of the principal components (the columns) with the observed/standardized variables (the rows) is stored. If *Cov_Matrix* is present and nonzero, the correlations are with the observed variables; otherwise, the correlations are with the standardized variables (to a variance of 1.0). In the principal component model for factor analysis, matrix *Correlations* is the matrix of unrotated factor loadings.

CUM_PERCENT

Named variable into which the one-dimensional array containing the cumulative percent of the total variances explained by each principal component is stored.

DF

Named variable into which the number of degrees of freedom in *covariances* is stored. Keywords *Df* and *Stdev* must be used together.

DOUBLE

If present and nonzero, double precision is used.

EIGENVECTORS

Named variable into which the two-dimensional array containing the eigenvectors of *covariances*, stored columnwise, is stored. Each vector is normalized to have Euclidean length equal to the value 1. Also, the sign of each vector is set so that the largest component in magnitude (the first of the largest if ties exist) is made positive.

STDEV

Named variable into which the one-dimensional array containing the estimated asymptotic standard errors of the eigenvalues is stored. Keywords *Df* and *Stdev* must be used together.

Discussion

The `IMSL_PRINC_COMP` function finds the principal components of a set of variables from a sample covariance or correlation matrix. The characteristic roots, characteristic vectors, standard errors for the characteristic roots, and the correlations of the principal component scores with the original variables are computed. Principal

components obtained from correlation matrices are the same as principal components obtained from standardized variables (to unit variance).

The principal component scores are the elements of the vector $y = \Gamma^T x$, where Γ is the matrix whose columns are the characteristic vectors (eigenvectors) of the sample covariance (or correlation) matrix and x is the vector of observed (or standardized) random variables. The variances of the principal component scores are the characteristic roots (eigenvalues) of the covariance (correlation) matrix.

Asymptotic variances for the characteristic roots were first obtained by Girschick (1939) and are given more recently by Kendall et al. (1983, p. 331). These variances are computed either for covariance matrices or for correlation matrices.

The correlations of the principal components with the observed (or standardized) variables are given in the matrix *correlations*. When the principal components are obtained from a correlation matrix, *Correlations* is the same as the matrix of unrotated factor loadings obtained for the principal components model for factor analysis.

Examples

Example 1

In this example, principal components are computed for a nine-variable covariance matrix.

```
covariances = $
[[1.0, 0.523, 0.395, 0.471, 0.346, 0.426, 0.576, 0.434, 0.639], $
 [0.523, 1.0, 0.479, 0.506, 0.418, 0.462, 0.547, 0.283, 0.645], $
 [0.395, 0.479, 1.0, 0.355, 0.27, 0.254, 0.452, 0.219, 0.504], $
 [0.471, 0.506, 0.355, 1.0, 0.691, 0.791, 0.443, 0.285, 0.505], $
 [0.346, 0.418, 0.27, 0.691, 1.0, 0.679, 0.383, 0.149, 0.409], $
 [0.426, 0.462, 0.254, 0.791, 0.679, 1.0, 0.372, 0.314, 0.472], $
 [0.576, 0.547, 0.452, 0.443, 0.383, 0.372, 1.0, 0.385, 0.68], $
 [0.434, 0.283, 0.219, 0.285, 0.149, 0.314, 0.385, 1.0, 0.47], $
 [0.639, 0.645, 0.504, 0.505, 0.409, 0.472, 0.68, 0.47, 1.0]]
values = IMSL_PRINC_COMP(covariances)
PM, values, Title = 'Eigenvalues:'

Eigenvalues:
4.67692
1.26397
0.844450
0.555027
0.447076
0.429125
0.310241
```

```
0.277006
0.196197
```

Example 2

In this example, principal components are computed for a nine-variable correlation matrix. This example uses the same data as the first example.

```
values = IMSL_PRINC_COMP(covariances, /CORR_MATRIX, $
    EIGENVECTORS   = ev, $
    STDEV          = stdev, $
    DF             = 100, $
    CUM_PERCENT    = cp, $
    CORRELATIONS   = a)
PM, [[values],[ev]], TITLE = 'Eigenvalue Eigenvector:', $
    FORMAT = '(f7.2, 2x, 9f7.2)'
```

Eigenvalue Eigenvector:

```
4.68  0.35 -0.24  0.14 -0.33 -0.11  0.80  0.17 -0.12 -0.05
1.26  0.35 -0.11 -0.28 -0.22  0.77 -0.20  0.14 -0.30 -0.01
0.84  0.28 -0.27 -0.56  0.69 -0.15  0.15  0.01 -0.04 -0.10
0.56  0.37  0.40  0.04  0.12  0.00  0.12 -0.40 -0.12  0.71
0.45  0.31  0.50 -0.07 -0.02 -0.28 -0.18  0.73  0.01  0.00
0.43  0.35  0.46  0.18  0.11  0.12  0.07 -0.37  0.09 -0.68
0.31  0.35 -0.27 -0.07 -0.35 -0.52 -0.44 -0.29 -0.34 -0.11
0.28  0.24 -0.32  0.74  0.43  0.09 -0.20  0.19 -0.16  0.05
0.20  0.38 -0.25 -0.01 -0.15  0.05 -0.15 -0.03  0.85  0.12
```

```
PM, a, Title = 'Matrix A:', FORMAT = '(9f7.2)'
```

Matrix A:

```
0.75 -0.26  0.13 -0.25 -0.07  0.52  0.10 -0.07 -0.02
0.76 -0.12 -0.26 -0.16  0.51 -0.13  0.08 -0.16 -0.00
0.60 -0.30 -0.51  0.52 -0.10  0.10  0.01 -0.02 -0.04
0.79  0.45  0.04  0.09  0.00  0.08 -0.22 -0.06  0.31
0.68  0.56 -0.07 -0.02 -0.19 -0.12  0.41  0.00  0.00
0.75  0.51  0.17  0.08  0.08  0.05 -0.21  0.05 -0.30
0.75 -0.31 -0.07 -0.26 -0.35 -0.29 -0.16 -0.18 -0.05
0.52 -0.36  0.68  0.32  0.06 -0.13  0.10 -0.09  0.02
0.83 -0.28 -0.01 -0.11  0.03 -0.10 -0.01  0.45  0.05
```

```
PM, [[values], [stdev], [cp]], Title = 'Eigenvalue STD PCT', $
    FORMAT = '(3(3x,F5.2))'
```

```
Eigenvalue  STD    PCT
4.68        0.65   0.52
1.26        0.18   0.66
0.84        0.10   0.75
0.56        0.09   0.82
```

0.45	0.09	0.87
0.43	0.09	0.91
0.31	0.09	0.95
0.28	0.10	0.98
0.20	0.11	1.00

Errors

Warning Errors

STAT_100_DF—Because the number of degrees of freedom in *Covariances* and *Df* is less than or equal to zero, 100 degrees of freedom will be used.

STAT_COV_NOT_NONNEG_DEF—Keyword *Eigenvalues*(#) = #. One or more eigenvalues much less than zero are computed. The matrix *Covariances* is not nonnegative definite. In order to continue computations of *Eigenvalues* and *Correlations*, these eigenvalues are treated as zero.

STAT_FAILED_TO_CONVERGE—Iteration for the eigenvalue failed to converge in 100 iterations before deflating.

Version History

6.4	Introduced

IMSL_FACTOR_ANALYSIS

The `IMSL_FACTOR_ANALYSIS` function extracts initial factor-loading estimates in factor analysis.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_FACTOR_ANALYSIS(covariances, n_factors [, ALPHA=value]
    [, CHI_SQ_TEST=variable] [, /DOUBLE] [, EIGENVALUES=variable]
    [, EPS=value] [, F_MIN=variable] [, /GEN_LSQ] [, /IMAGE]
    [, ITERS=variable] [, ITMAX=value] [, LAST_STEP=variable]
    [, MAX_LIKELIHOOD=value] [, MAX_STEPS=value]
    [, /PRINC_COMP] [, /PRINC_FACTOR] [, SWITCH_EPS=value]
    [, TUCKER_COEF=variable] [, UNIQUE_VAR_IN=array]
    [, UNIQUE_VAR_OUT=array] [, /UNWGT_LSQ])
```

Return Value

A two-dimensional array containing the matrix of factor loadings.

Arguments

covariances

Two-dimensional array containing the variance-covariance or correlation matrix.

n_factors

Number of factors in the model.

Keywords

ALPHA

The number of degrees of freedom in *covariances*. Using *Alpha* forces the alpha-factor analysis (common factor model) method to be used to obtain the estimates.

Note

Keywords *Max_Likelihood*, *Princ_Comp*, *Princ_Factor*, *Unwgt_Lsq*, *Gen_Lsq*, *Image*, and *Alpha* cannot be used together.

CHI_SQ_TEST

Named variable into which a one-dimensional array of length 3, containing the chi-squared test statistics, is stored. The contents of the array are, in order, the number of degrees of freedom in chi-squared, the chi-squared test statistic for testing that *n_factors* common factors are adequate for the data, and the probability of a greater chi-squared statistic.

DOUBLE

If present and nonzero, double precision is used.

EIGENVALUES

Named variable into which a one-dimensional array of length `N_ELEMENTS(covariances(0, *))` containing the eigenvalues of the matrix from which the factors were extracted is stored.

EPS

Convergence criterion used to terminate the iterations. For the unweighted least squares, generalized least squares, or maximum likelihood methods, convergence is assumed when the relative change in the criterion is less than *Eps*. For alpha-factor analysis, convergence is assumed when the maximum change (relative to the variance) of a uniqueness is less than *Eps*. Default: *Eps* = 0.0001

F_MIN

Named variable into which the value of the function minimum is stored.

GEN_LSQ

If present and nonzero, the generalized least-squares (common factor model) method is used to obtain the estimates.

Note

Keywords *Max_Likelihood*, *Princ_Comp*, *Princ_Factor*, *Unwgt_Lsq*, *Gen_Lsq*, *Image*, and *Alpha* cannot be used together.

IMAGE

If present and nonzero, the image-factor analysis (common factor model) method is used to obtain the estimates.

Note

Keywords *Max_Likelihood*, *Princ_Comp*, *Princ_Factor*, *Unwgt_Lsq*, *Gen_Lsq*, *Image*, and *Alpha* cannot be used together.

ITERS

Named variable into which the number of iterations is stored.

ITMAX

Maximum number of iterations in the iterative procedure. Default: *Itmax* = 60

LAST_STEP

Named variable into which an array of length `N_ELEMENTS(covariances(0, *))` containing the updates of the unique variance estimates when convergence was reached (or the iterations terminated) is stored.

MAX_LIKELIHOOD

The number of degrees of freedom in covariances. Using *Max_Likelihood* forces the maximum likelihood (common factor) model to be used to obtain the estimates.

Note

Keywords *Max_Likelihood*, *Princ_Comp*, *Princ_Factor*, *Unwgt_Lsq*, *Gen_Lsq*, *Image*, and *Alpha* cannot be used together.

MAX_STEPS

Maximum number of step halvings allowed during any one iteration. Default: *Max_Steps* = 10

PRINC_COMP

If present and nonzero, the principal component (principal component model) is used to obtain the estimates.

Note

Keywords *Max_Likelihood*, *Princ_Comp*, *Princ_Factor*, *Unwgt_Lsq*, *Gen_Lsq*, *Image*, and *Alpha* cannot be used together.

PRINC_FACTOR

If present and nonzero, the principal factor (common factor model) is used to obtain the estimates.

Note

Keywords *Max_Likelihood*, *Princ_Comp*, *Princ_Factor*, *Unwgt_Lsq*, *Gen_Lsq*, *Image*, and *Alpha* cannot be used together.

SWITCH_EPS

Convergence criterion used to switch to exact second derivatives. When the largest relative change in the unique standard deviation vector is less than *Switch_Eps*, exact second derivative vectors are used. The value of *Switch_Eps* is not used with the principal component, principal factor, image-factor analysis, or alpha-factor analysis methods. Default: *Switch_Eps* = 0.1

TUCKER_COEF

Named variable into which the Tucker reliability coefficient is stored.

UNIQUE_VAR_IN

One-dimensional array of length $N\_ELEMENTS(covariances(0, *))$ containing the initial estimates of the unique variances. Default: initial estimates are taken as the constant $1 - n\_factors/2 * N\_ELEMENTS(covariances(0, *))$ divided by the diagonal elements of the inverse of *covariances*

UNIQUE_VAR_OUT

One-dimensional array of length $N\_ELEMENTS(covariances(0, *))$ containing the estimated unique variances.

UNWGT_LSQ

If present and nonzero, the unweighted least-squares (common factor model) method is used to obtain the estimates. This option is the default.

Note

Keywords *Max_Likelihood*, *Princ_Comp*, *Princ_Factor*, *Unwgt_Lsq*, *Gen_Lsq*, *Image*, and *Alpha* cannot be used together.

Discussion

Function *S* computes unrotated factor loadings in exploratory factor-analysis models. Models available in *IMSL_FACTOR_ANALYSIS* are the principal component model for factor analysis and the common factor model with additions to the common factor model in alpha-factor analysis and image analysis. Methods of estimation include principal components, principal factor, image analysis, unweighted least squares, generalized least squares, and maximum likelihood.

In the factor-analysis model used for factor extraction, the basic model is given as $\Sigma = \Lambda\Lambda^T + \Psi$, where Σ is the $p \times p$ population covariance matrix, Λ is the $p \times k$ matrix of factor loadings relating the factors f to the observed variables x , and Ψ is the $p \times p$ matrix of covariances of the unique errors e . Here, $p = \text{N\_ELEMENTS}(\text{covariances}(0, *))$ and $k = n\_factors$. The relationship between the factors, the unique errors, and the observed variables is given as $x = \Lambda f + e$, where in addition, the expected values of e , f , and x are assumed to be zero. (The sample means can be subtracted from x if the expected value of x is not zero.) It also is assumed that each factor has unit variance, that the factors are independent of each other, and that the factors and the unique errors are mutually independent. In the common factor model, the elements of unique errors e also are assumed to be independent of one another so that the matrix Ψ is diagonal. This is not the case in the principal component model in which the errors may be correlated.

Further differences between the various methods concern the criterion that is optimized and the amount of computer effort required to obtain estimates. Generally speaking, the least-squares and maximum likelihood methods, which use iterative algorithms, require the most computer time with the principal factor, principal component and the image methods requiring much less time since the algorithms in these methods are not iterative. The algorithm in alpha-factor analysis is also iterative, but the estimates in this method generally require somewhat less computer effort than the least squares and maximum likelihood estimates. In all methods, one eigensystem analysis is required on each iteration.

Principal Component and Principal Factor Methods

Both the principal component and principal factor methods compute the factor-loading estimates as:

$$\hat{\Gamma} \hat{\Delta}^{-1/2}$$

where Γ and the diagonal matrix Δ are the eigenvectors and eigenvalues of a matrix. In the principal component model, the eigensystem analysis is performed on the sample covariance (correlation) matrix S , while in the principal factor model, the matrix $(S + \Psi)$ is used. If the unique error variances Ψ are not known in the principal factor mode, then `IMSL_FACTOR_ANALYSIS` obtains estimates for them.

The basic idea in the principal component method is to find factors that maximize the variance in the original data that is explained by the factors. Because this method allows the unique errors to be correlated, some factor analysts insist that the principal component method is not a factor analytic method. Usually, however, the estimates obtained by the principal component model and factor analysis model are quite similar.

It should be noted that both the principal component and principal factor methods give different results when the correlation matrix is used in place of the covariance matrix. In fact, any rescaling of the sample covariance matrix can lead to different estimates with either of these methods. A further difficulty with the principal factor method is the problem of estimating the unique error variances. Theoretically, these variances must be known in advance and must be passed to `IMSL_FACTOR_ANALYSIS` using the keyword *Unique_Var_In*. In practice, the estimates of these parameters are produced by `IMSL_FACTOR_ANALYSIS` when *Unique_Var_In* is not specified. In either case, the resulting adjusted covariance (correlation) matrix:

$$S - \hat{\Psi}$$

may not yield the $n\_factors$ positive eigenvalues required for $n\_factors$ factors to be obtained. If this occurs, you must either lower the number of factors to be estimated or give new unique error variance values.

Least-squares and Maximum Likelihood Methods

Unlike the previous two methods, the algorithm used to compute estimates in this section is iterative (see Jöreskog 1977). As with the principal factor model, you can either initialize the unique error variances or allow `IMSL_FACTOR_ANALYSIS` to compute initial estimates. Unlike the principal factor method, `IMSL_FACTOR_ANALYSIS` optimizes the criterion function with respect to both Ψ and Γ . (In the principal factor method, Ψ is assumed to be known. Given Ψ , estimates for Λ may be obtained.)

The major difference between the methods discussed in this section is in the criterion function that is optimized. Let S denote the sample covariance (correlation) matrix,

and let Σ denote the covariance matrix that is to be estimated by the factor model. In the unweighted least-squares method, also called the iterated principal factor method or the minres method (see Harman 1976, p. 177), the function minimized is the sum-of-squared differences between S and Σ . This is written as:

$$\Phi_{ul} = 0.5 (\text{trace}(S - \Sigma)^2)$$

Generalized least-squares and maximum-likelihood estimates are asymptotically equivalent methods. Maximum-likelihood estimates maximize the (normal theory) likelihood:

$$\{\phi_{ml} = \text{trace}(\Sigma^{-1}S) - \log(|\Sigma^{-1}S|)\}$$

while generalized least squares optimizes the function:

$$\Phi_{gs} = \text{trace}(\Sigma S^{-1} - I)^2$$

In all three methods, a two-stage optimization procedure is used. This proceeds by first solving the likelihood equations for Λ in terms of Ψ and substituting the solution into the likelihood. This gives a criterion $\phi(\Psi, \Lambda(\Psi))$, which is optimized with respect to Ψ . In the second stage, the estimates:

$$\hat{\Lambda}$$

are obtained from the estimates for Ψ .

The generalized least-squares and maximum-likelihood methods allow for the computation of a statistic (*Chi_Sq_Test*) for testing that $n\_factors$ common factors are adequate to fit the model. This is a chi-squared test that all remaining parameters associated with additional factors are zero. If the probability of a larger chi-squared is so small that the null hypothesis is rejected, then additional factors are needed (although these factors may not be of any practical importance). Failure to reject does not legitimize the model. The statistic *Chi_Sq_Test* is a likelihood ratio statistic in maximum likelihood estimation. As such, it asymptotically follows a chi-squared distribution with degrees of freedom given by *Df*.

The Tucker and Lewis reliability coefficient, ρ , is returned by *Tucker_Coef* when the maximum likelihood or generalized least-squares methods are used. This coefficient is an estimate of the ratio of explained variation to the total variation in the data. It is computed as follows:

$$\rho = \frac{mM_0 - mM_k}{mM_0 - 1}$$

$$M_0 = \frac{-\ln(|S|)}{p(p-1)/2}$$

$$m = d - \frac{2p+5}{6} - \frac{2k}{6}$$

$$M_k = \frac{\phi}{((p-k)^2 - p - k)/2}$$

where:

- $|S|$ is the determinant of covariances
- $p = \text{N\_ELEMENTS}(\text{covariances}(0, *))$
- $k = \text{N\_ELEMENTS}(\text{covariances}(0, *))$
- ϕ is the optimized criterion; and $d = Df$

Image Analysis

The term *image analysis* is used here to denote the noniterative image method of Kaiser (1963), rather than the image analysis discussed by Harman (1976, p. 226). The image method (as well as the alpha-factor analysis method) begins with the notion that only a finite number from an infinite number of possible variables have been measured. The image-factor pattern is calculated under the assumption that the ratio of the number of factors to the number of observed variables is near zero, so that a very good estimate for the unique error variances (for standardized variables) is given as 1 minus the squared multiple correlation of the variable under consideration with all variables in the covariance matrix.

First, the matrix $D^2 = (\text{diag}(S^{-1}))^{-1}$ is computed, where the operator “diag” results in a matrix consisting of the diagonal elements of its argument and S is the sample covariance (correlation) matrix. Then, the eigenvalues Λ and eigenvectors Γ of the matrix $D^{-1}SD^{-1}$ are computed. Finally, the unrotated image-factor pattern is computed as $D\Gamma [(\Lambda - I)^2 \Lambda^{-1}]^{1/2}$.

Alpha-factor Analysis

The alpha-factor analysis method of Kaiser and Caffrey (1965) finds factor-loading estimates to maximize the correlation between the factors and the complete universe of variables of interest. The basic idea in this method is that only a finite number of variables out of a much larger set of possible variables is observed. The population

factors are linearly related to this larger set, while the observed factors are linearly related to the observed variables. Let f denote the factors obtainable from a finite set of observed random variables, and let ξ denote the factors obtainable from the universe of observable variables. Then, the alpha method attempts to find factor-loading estimates so as to maximize the correlation between f and ξ . In order to obtain these estimates, the iterative algorithm of Kaiser and Caffrey (1965) is used.

Comments

1. IMSL_FACTOR_ANALYSIS makes no attempt to solve for $n\_factors$. In general, if $n\_factors$ is not known in advance, several different values of $n\_factors$ should be used and the most reasonable value kept in the final solution.
2. Iterative methods are generally thought to be superior from a theoretical point of view, but in practice, often lead to solutions that differ little from the noniterative methods. For this reason, it is usually suggested that a noniterative method be used in the initial stages of the factor analysis and that the iterative methods be used when issues such as the number of factors have been resolved.
3. Initial estimates for the unique variances can be input. If the iterative methods fail for these values, new initial estimates should be tried. These can be obtained by use of another factoring method. (Use the final estimates from the new method as the initial estimates in the old method.)

Examples

Example 1

In this example, factor analysis is performed for a nine-variable matrix using the default method of unweighted least squares.

```
covariances = $
[[1.0,   0.523, 0.395, 0.471, 0.346, 0.426, 0.576, 0.434, 0.639], $
 [0.523, 1.0,   0.479, 0.506, 0.418, 0.462, 0.547, 0.283, 0.645], $
 [0.395, 0.479, 1.0,   0.355, 0.27,   0.254, 0.452, 0.219, 0.504], $
 [0.471, 0.506, 0.355, 1.0,   0.691, 0.791, 0.443, 0.285, 0.505], $
 [0.346, 0.418, 0.27,   0.691, 1.0,   0.679, 0.383, 0.149, 0.409], $
 [0.426, 0.462, 0.254, 0.791, 0.679, 1.0,   0.372, 0.314, 0.472], $
 [0.576, 0.547, 0.452, 0.443, 0.383, 0.372, 1.0,   0.385, 0.68], $
 [0.434, 0.283, 0.219, 0.285, 0.149, 0.314, 0.385, 1.0,   0.47], $
 [0.639, 0.645, 0.504, 0.505, 0.409, 0.472, 0.68,  0.47,  1.0]]
n_factors = 3
a = IMSL_FACTOR_ANALYSIS(covariances, n_factors)
PM, a, Title = 'Unrotated Loadings:'
```

Unrotated Loadings:

0.701801	-0.231594	0.0795559
0.719964	-0.137227	-0.208225
0.535122	-0.214389	-0.22709
0.790669	0.405017	0.00704257
0.653203	0.422066	-0.104563
0.753915	0.484247	0.160720
0.712674	-0.281911	-0.0700779
0.483540	-0.262720	0.461992
0.819210	-0.313728	-0.0198735

Example 2

The following data were originally analyzed by Emmett (1949). There are 211 observations on nine variables. Following Lawley and Maxwell (1971), three factors are obtained by the method of maximum likelihood. This example uses the same data as the first example.

```
n_factors = 3
a = IMSL_FACTOR_ANALYSIS(covariances, n_factors, $
Max_Likelihood=210, Switch_Eps=0.01, $
Eps=0.000001, Itmax=30, Max_Steps=10)
PM, a, Title = 'Unrotated Loadings:'
```

Unrotated Loadings:

0.664210	-0.320874	0.0735207
0.688833	-0.247138	-0.193280
0.492616	-0.302161	-0.222433
0.837198	0.292427	-0.0353954
0.705002	0.314794	-0.152784
0.818701	0.376672	0.104524
0.661494	-0.396031	-0.0777453
0.457925	-0.295526	0.491347
0.765668	-0.427427	-0.0116992

Errors

Warning Errors

STAT_VARIANCES_INPUT_IGNORED—When using the keyword *Princ_Comp*, the unique variances are assumed to be zero. Input for *Unique_Var_In* is ignored.

STAT_TOO_MANY_ITERATIONS—Too many iterations. Convergence is assumed.

STAT_NO_DEG_FREEDOM—No degrees of freedom for the significance testing.

STAT_TOO_MANY_HALVINGS—Too many step halvings. Convergence is assumed.

Fatal Errors

STAT_HESSIAN_NOT_POS_DEF—Approximate Hessian is not semidefinite on iteration #. The computations cannot proceed. Try using different initial estimates.

STAT_FACTOR_EVAL_NOT_POS—Variable *Eigenvalues*(#) = #. An eigenvalue corresponding to a factor is negative or zero. Either use different initial estimates for *Unique_Var_In* or reduce the number of factors.

STAT_COV_NOT_POS_DEF—Parameter *covariances* is not positive semidefinite. The computations cannot proceed.

STAT_COV_IS_SINGULAR—Matrix *covariances* is singular. The computations cannot continue because variable # is linearly related to the remaining variables.

STAT_COV_EVAL_ERROR—An error occurred in calculating the eigenvalues of the adjusted (inverse) covariance matrix. Check *covariances*.

STAT_ALPHA_FACTOR_EVAL_NEG—In alpha-factor analysis on iteration #, eigenvalue # is #. As all eigenvalues corresponding to the factors must be positive, either the number of factors must be reduced or new initial estimates for *Unique_Var_In* must be given.

Version History

6.4	Introduced

IMSL_DISCR_ANALYSIS

The IMSL_DISCR_ANALYSIS procedure performs a linear or a quadratic discriminant function analysis among several known groups.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
IMSL_DISCR_ANALYSIS, x, n_groups [, CLASS_MEMBER=variable]
[, CLASS_TABLE=variable] [, COEFFICIENTS=variable]
[, COVARIANCES=variable] [, /DOUBLE] [, GROUP_COUNTS=variable]
[, IDX_COLS=array] [, IDX_VARS=array] [, METHOD=value]
[, /PRIOR_EQUAL] [, PRIOR_INPUT=array] [, PRIOR_OUTPUT=variable]
[, /PRIOR_PROP] [, MAHALANOBIS=variable] [, MEANS=variable]
[, NMISSING=variable] [, PROB=variable] [, STATS=variable]
```

Arguments

n_groups

Number of groups in the data.

x

Two-dimensional array of size n_rows by $n\_variables + 1$ containing the data where $n\_rows = N\_ELEMENTS(x(*,0))$, the number of rows to be processed and $n\_variables =$ number of variables to be used in the discrimination. The first $n\_variables$ columns correspond to the variables, and the last column contains the group numbers. The groups must be numbered 1, 2, ..., n_groups .

Keywords

CLASS_MEMBER

Named variable into which an one-dimensional integer array of length n_rows containing the group to which the observation was classified is stored.

If an observation has an invalid group number, frequency, or weight when the leaving-out-one method has been specified, then the observation is not classified and the corresponding elements of *Class_Member* (and *Prob*, see *Prob* below) are set to zero.

CLASS_TABLE

Named variable into which a two-dimensional array of size n_groups by n_groups containing the classification table is stored. Each observation that is classified and has a group number 1.0, 2.0, ..., n_groups is entered into the table. The rows of the table correspond to the known group membership. The columns refer to the group to which the observation was classified.

COEFFICIENTS

Named variable into which a two-dimensional array of size n_groups by $(n\_variables + 1)$ containing the linear discriminant coefficients is stored. The first column of *Coefficients* contains the constant term, and the remaining columns contain the variable coefficients. Row $i - 1$ of *Coefficients* corresponds to group i , for $i = 1, 2, \dots, n\_variables + 1$. Array *Coefficients* are always computed as the linear discriminant function coefficients even when quadratic discrimination is specified.

COVARIANCES

Named variable into which a three-dimensional array of size g by $n\_variables$ by $n\_variables$ containing covariance results is stored. The within-group covariance matrices (*Method* 1, 2, 4, and 5 only) is the first $g-1$ matrices, and the pooled covariance matrix is the g -th matrix.

DOUBLE

If present and nonzero, double precision is used.

GROUP_COUNTS

Named variable into which an one-dimensional integer array of length n_groups containing the number of observations in each group is stored.

IDX_COLS

One-dimensional array containing the indices of the variables to be used in the analysis.

IDX_VARS

Three element array indicating the column numbers of x in which particular types of data are stored. Columns are numbered 0 ... $N\_ELEMENTS(Idx\_Cols) - 1$.

$Idx\_Vars(0)$ contains the index for the column of x in which the group numbers are stored.

$Idx\_Vars(1)$ and $Idx\_Vars(2)$ contain the column numbers of x in which the frequencies and weights, respectively, are stored. Set $Idx\_Vars(1) = -1$ if there will be no column for frequencies. Set $Idx\_Vars(2) = -1$ if there will be no column for weights. Weights are rounded to the nearest integer. Negative weights are not allowed.

Defaults: $Idx\_Cols = 0, 1, \dots, n\_variables - 1$,

$Idx\_Vars(0) = n\_variables$,

$Idx\_Vars(1) = -1$, and

$Idx\_Vars(2) = -1$

METHOD

Method of discrimination. The method chosen determines whether linear or quadratic discrimination is used, whether the group covariance matrices are computed (the pooled covariance matrix is always computed), and whether the leaving-out-one or the reclassification method is used to classify each observation. The *Method* values are shown in [Table 21-1](#).

Method	discrimination method	covariances computed	classification method
1	linear	pooled, group	reclassification
2	quadratic	pooled, group	reclassification
3	linear	pooled	reclassification
4	linear	pooled, group	<i>leaving-out-one</i>
5	quadratic	pooled, group	<i>leaving-out-one</i>
6	linear	pooled	<i>leaving-out-one</i>

Table 21-1: Method Values

In the leaving-out-one method of classification, the posterior probabilities are adjusted so as to eliminate the effect of the observation from the sample statistics prior to its classification. In the classification method, the effect of the observation is not eliminated from the classification function. Default: *Method* = 1

PRIOR_EQUAL

By default, (or if *Prior_Equal* is used), equal prior probabilities are calculated as $1.0/n\_groups$. Keywords *Prior_Equal*, *Prior_Prop*, and *Prior_Input* must not be used together.

PRIOR_INPUT

If present, an array of length *n_groups* containing the prior probabilities for each group, such that the sum of all prior probabilities is equal to 1.0. Keywords *Prior_Input*, *Prior_Equal*, and *Prior_Prop* must not be used together.

PRIOR_OUTPUT

Named variable into which an one-dimensional array of length *n_groups* containing the most recently calculated or input prior probabilities is stored.

PRIOR_PROP

If present, prior probabilities are calculated to be proportional to the sample size in each group. Keywords *Prior_Prop*, *Prior_Equal*, and *Prior_Input* must not be used together.

MAHALANOBIS

Named variable into which a two-dimensional array of size *n_groups* by *n_groups* containing the Mahalanobis distances:

$$D_{ij}^2$$

between the group means is stored.

For linear discrimination, the Mahalanobis distance is computed using the pooled covariance matrix. Otherwise, the Mahalanobis distance:

$$D_{ij}^2$$

between group means *i* and *j* is computed using the within covariance matrix for group *i* in place of the pooled covariance matrix.

MEANS

Named variable into which a two-dimensional array of size n_groups by $n\_variables$ containing the variable means is stored. The i -th row of means contains the group i variable means.

NMISSING

Named variable into which the number of rows of data encountered containing missing values (NaN) for the classification, group, weight, and/or frequency variables is stored. If a row of data contains a missing value (NaN) for any of these variables, that row is excluded from the computations.

PROB

Named variable into which a two-dimensional array of size n_rows by n_groups containing the posterior probabilities for each observation is stored.

STATS

Named variable into which an one-dimensional array of length $4 + 2 * (n\_groups + 1)$ containing various statistics of interest is stored. The first element of *Stats* is the sum of the degrees of freedom for the within-covariance matrices. The second, third, and fourth elements of *Stats* correspond to the chi-squared statistic, its degrees of freedom, and the probability of a greater chi-squared, respectively, of a test of the homogeneity of the within-covariance matrices (not computed if *Method* is equal to 3 or 6). The fifth through $5 + n\_groups$ elements of *Stats* contain the log of the determinants of each group's covariance matrix (not computed if *Method* is equal to 3 or 6) and of the pooled covariance matrix (element $4 + n\_groups$). Finally, the last $n\_groups + 1$ elements of *Stats* contain the sum of the weights within each group, and in the last position, the sum of the weights in all groups.

Comments

1. Common choices for the Bayesian prior probabilities are given by:
 $Prior\_Input(i) = 1.0/n\_groups$ (equal priors)
 $Prior\_Input(i) = Group\_Count/n\_rows$ (proportional priors)
 $Prior\_Input(i) =$ Past history or subjective judgment.
 In all cases, the priors should sum to 1.0.

Discussion

IMSL_DISCR_ANALYSIS performs discriminant function analysis using either linear or quadratic discrimination. The output includes a measure of distance between

the groups, a table summarizing the classification results, a matrix containing the posterior probabilities of group membership for each observation, and the within-sample means and covariance matrices. Linear discriminant function coefficients are also computed.

Covariance matrices are defined as follows: Let N_i denote the sum of frequencies of observations in group i and M_i denote the number of observations in group i . Then, if S_i denotes the within-group i covariance matrix:

$$S_i = \frac{1}{N_i - 1} \sum_{j=1}^{M_i} w_j f_j (x_j - \bar{x})(x_j - \bar{x})^T$$

Where w_j is the weight of the j -th observation in group i , f_j is the frequency, x_j is the j -th observation column vector (in group i), and:

$$\bar{x}$$

denotes the mean vector of the observations in group i . The mean vectors are computed as:

$$\bar{x} = \left(\frac{1}{W_i} \right) \sum_{j=1}^{M_i} w_j f_j x_j \quad \text{where } W_i = \sum_{j=1}^{M_i} w_j f_j$$

Given the means and the covariance matrices, the linear discriminant function for group i is computed as:

$$z_i = \ln(p_i) - 0.5 \bar{x}_i^T S_p^{-1} \bar{x}_i + x^T S_p^{-1} \bar{x}_i$$

where $\ln(p_i)$ is the natural log of the prior probability for the i -th group, x is the observation to be classified, and S_p denoted the pooled covariance matrix.

Let S denote either the pooled covariance matrix of one of the within-group covariance matrices S_i . (S will be the pooled covariance matrix in linear discrimination, and S_i otherwise.) The Mahalanobis distance between group i and group j is computed as:

$$D_{ij}^2 = (\bar{x}_i - \bar{x}_j)^T S^{-1} (\bar{x}_i - \bar{x}_j)$$

Finally, the asymptotic chi-squared test for the equality of covariance matrices is computed as follows (Morrison 1976, p. 252):

$$\gamma = C^{-1} \sum_{i=1}^k n_i \{ \ln(|S_p|) - \ln(|S_i|) \}$$

where n_i is the number of degrees of freedom in the i -th sample covariance matrix, k is the number of groups, and:

$$C^{-1} = \frac{1 - 2p^2 + 3p - 1}{6(p + 1)(k - 1)} \left(\sum_{i=1}^k \frac{1}{n_i} - \frac{1}{\sum_j n_j} \right)$$

where p is the number of variables.

The estimated posterior probability of each observation x belonging to group is computed using the prior probabilities and the sample mean vectors and estimated covariance matrices under a multivariate normal assumption. Under quadratic discrimination, the within-group covariance matrices are used to compute the estimated posterior probabilities. The estimated posterior probability of an observation x belonging to group i is:

$$\hat{q}_i(x) = \frac{\exp(-0.5 D_i^2(x))}{\sum_{j=1}^k \exp(-0.5 D_j^2(x))}$$

where:

$$D_i^2(x) = \begin{cases} (x - \bar{x}_i)^T S_i^{-1} (x - \bar{x}_i) + \ln(|S_i|) - 2 \ln(p_i) & \text{Method} = 1 \text{ or } 2 \\ (x - \bar{x}_i)^T S_p^{-1} (x - \bar{x}_i) - 2 \ln(p_i) & \text{Method} = 3 \end{cases}$$

For the leaving-out-one method of classification (*Method* equal to 4, 5 or 6), the sample mean vector and sample covariance matrices in the formula for:

$$D_i^2$$

are adjusted so as to remove the observation x from their computation. For linear discrimination (*Method* equal to 1, 2, 4, or 6), the linear discriminant function coefficients are actually used to compute the same posterior probabilities.

Using the posterior probabilities, each observation in x is classified into a group; the result is tabulated in the array *Class_Table* and saved in the array *Class_Member*. Array *Class_Table* is not altered at this stage if $x(i)(Idx\_Vars(0))$ contains a group number that is out of range. If the reclassification method is specified, then all observations with no missing values in the $n\_variables$ classification variables are classified. When the leaving-out-one method is used, observations with invalid group

numbers, weights, frequencies, or classification variables are not classified. Regardless of the frequency, a 1 is added (or subtracted) from *Class_Table* for each row of *x* that is classified and contains a valid group number.

When *Method* > 3, adjustment is made to the posterior probabilities to remove the effect of the observation in the classification rule. In this adjustment, each observation is presumed to have a weight of $x(i)(Idx\_Vars(2))$ if $Idx\_Vars(2) > -1$ (and a weight of 1.0 if $Idx\_Vars(2) = -1$), and a frequency of 1.0. See Lachenbruch (1975, p. 36) for the required adjustment.

The covariance matrices are computed from their *LU* factorizations.

Example

The following example uses liner discrimination with equal prior probabilities on Fisher's (1936) iris data.

```
.RUN
PRO print_results, counts, table, d2, prior_out, coef, means, $
  cov, stats, nrmiss
  num = INDGEN(3)
  PRINT, '          Counts'
  PRINT, num + 1, FORMAT = '(3I5)'
  PRINT, counts, FORMAT = '(3I5)'
  PRINT
  PRINT, '          Table'
  PRINT, num + 1, FORMAT = '(2X, 3I5)'
  FOR i = 0, 2 DO $
    PRINT, num(i) + 1, table(i, *), FORMAT = '(I2, 3I5)'
  PRINT
  PRINT, '          D2'
  PRINT, num + 1, FORMAT = '(3I7)'
  FOR i = 0, 2 DO $
    PRINT, num(i) + 1, d2(i, *), FORMAT = '(I2, 3F7.1)'
  PRINT
  PRINT, '          Prior OUT'
  PRINT, num + 1, FORMAT = '(3I10)'
  PRINT, prior_out, FORMAT = '(3F10.4)'
  PRINT
  num = INDGEN(5)
  PRINT, '          Coef'
  PRINT, num + 1, FORMAT = '(1X, 5I10)'
  FOR i = 0, 2 DO $
    PRINT, num(i) + 1, coef(i, *), FORMAT = '(I2, 5F10.1)'
  PRINT
  num = INDGEN(4)
  PRINT, '          Means'
  PRINT, num + 1, FORMAT = '(4I10)'
```

```

FOR i = 0, 2 DO $
    PRINT, num(i) + 1, means(i, *), FORMAT = '(I2, 4F10.3)'
PRINT
PRINT, '          Covariance'
PRINT, num + 1, FORMAT = '(4I10)'
FOR i = 0, 3 DO $
    PRINT, num(i) + 1, cov(0, *, i), FORMAT = '(I2, 4F10.4)'
PRINT
num = INDGEN(12)
PRINT, '          Stats'
FOR i = 0, 11 DO $
    PRINT, num(i) + 1, stats(i)
PRINT
PRINT, 'nrmiss = ', nrmiss
END

```

```

idxv = [1, 2, 3, 4]
idxc = [0, -1, -1]
n_groups = 3
method = 3
; Retrieve the Fisher Iris Data Set
x = IMSL_STATDATA(3)
IMSL_DISCR_ANALYSIS, x, n_groups, Idx_Vars = idxv, $
    Idx_cols = idxc, Method = method, /Prior_Equal, $
    Prior_Output = prior_out, Group_Counts = counts, $
    Means = means, Covariances = cov, $
    Coefficients = coef, Class_Member = cm, $
    Class_Table = table, Prob = prob, $
    Mahalanobis = d2, Stats = stats, Nmissing = nrmiss
print_results, counts, table, d2, prior_out, coef, means, $
    cov, stats, nrmiss

```

```

Counts
1      2      3
50    50    50

```

```

Table
      1      2      3
1    50      0      0
2      0    48      2
3      0      1    49

```

```

D2
      1      2      3
1 0.0    89.9  179.4
2 89.9      0.0   17.2
3 179.4   17.2      0.0

```

```

      Prior OUT
      1         2         3
0.3333    0.3333    0.3333
      Coef
      1         2         3         4         5
1      -86.3     23.5     23.6     -16.4     -17.4
2      -72.9     15.7      7.1       5.2       6.4
3     -104.4     12.4      3.7      12.8      21.1
      Means
      1         2         3         4
1      5.006     3.428     1.462     0.246
2      5.936     2.770     4.260     1.326
3      6.588     2.974     5.552     2.026
      Covariance
      1         2         3         4
1      0.2650     0.0927     0.1675     0.0384
2      0.0927     0.1154     0.0552     0.0327
3      0.1675     0.0552     0.1852     0.0427
4      0.0384     0.0327     0.0427     0.0419
      Stats
1          147.000
2           NaN
3           NaN
4           NaN
5           NaN
6           NaN
7           NaN
8      -9.95854
9          50.0000
10         50.0000
11         50.0000
12        150.000

nrmiss =          0

```

Errors

Warning Errors

STAT_BAD_OBS_1—In call #, row # of the data matrix, “x”, has group number = #. The group number must be an integer between 1.0 and “n_groups” = #, inclusively. This observation will be ignored.

STAT_BAD_OBS_2—The leaving-out-one method is specified but this observation does not have a valid group number (Its group number is #.). This observation (row #) is ignored.

STAT_BAD_OBS_3—The leaving-out-one method is specified but this observation does not have a valid weight or it does not have a valid frequency. This observation (row #) is ignored.

STAT_COV_SINGULAR_3—The group # covariance matrix is singular. “Stats(1)” cannot be computed. “Stats(1)” and “Stats(3)” are set to the missing value code (NaN).

Fatal Errors

STAT_COV_SINGULAR_1—The variance-covariance matrix for population number # is singular. The computations cannot continue.

STAT_COV_SINGULAR_2—The pooled variance-covariance matrix is singular. The computations cannot continue.

STAT_COV_SINGULAR_4—A variance-covariance matrix is singular. The index of the first zero element is equal to #.

Version History

6.4	Introduced



Chapter 22

Survival Analysis

This section contains the following topics:

Overview: Survival Analysis	1004	Survival Analysis Routines	1005

Overview: Survival Analysis

The routine described in this chapter has primary application in the areas of reliability and life testing, but they may find application in any situation in which time is a variable of interest. Kalbfleisch and Prentice (1980), Elandt-Johnson and Johnson (1980), Lee (1980), Gross and Clark (1975), Lawless (1982), and Chiang (1968) are references for discussing the models and methods used here.

[IMSL_SURVIVAL_GLM](#) fits any of several generalized linear models, and computes estimates of survival probabilities based on the same models.

Survival Analysis Routines

- [IMSL_SURVIVAL_GLM](#)—Analyzes survival data using a generalized linear model and estimates using various parametric modes.

IMSL_SURVIVAL_GLM

The IMSL_SURVIVAL_GLM function analyzes censored survival data using a generalized linear model and estimates survival probabilities and hazard rates for the various parametric models.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_SURVIVAL_GLM(n_class, n_continuous, model, x
    [, /CASE_ANALYSIS=variable] [, CLASS_VALS=variable]
    [, COEF_STAT=variable] [, COVARIANCES=variable]
    [, CRITERION=variable] [, /DOUBLE] [, EPS=value] [, EST_DELTA=value]
    [, EST_NOBS=value] [, EST_NPT=value] [, EST_PROB=variable]
    [, EST_TIME=value] [, EST_XBETA=variable] [, ICEN=value] [, IFIX=value]
    [, IFREQ=value] [, ILT=value] [, INDICES_EFFECTS=array]
    [, INIT_EST=array] [, IRT=value] [, ITERATIONS=variable] [, ITMAX=value]
    [, LAST_STEP=variable] [, LP_MAX=value] [, MAX_CLASS=value]
    [, MEANS=variable] [, N_CLASS_VALS=variable] [, NMISsing=variable]
    [, /NO_INTERCEPT] [, OBS_STATUS=variable] [, VAR_EFFECTS=array])
```

Return Value

An integer value indicating *n_coefficients*, where *n_coefficients* is the number of estimated coefficients in the model.

Arguments

model

Specifies the model used to analyze the data.

- 0—Exponential
- 1—Linear hazard
- 2—Log-normal

- 3—Normal
- 4—Log-logistic
- 5—Logistic
- 6—Log least extreme value
- 7—Least extreme value
- 8—Log extreme value
- 9—Extreme value
- 10—Weibull

See the *Discussion* section for more information about these models.

n_class

Number of classification variables.

n_continuous

Number of continuous variables.

x

Two-dimensional array of size $n\_observations$ by $((n\_class + n\_continuous) + m)$ containing data for the independent variables, dependent variable, and optional parameters where $n\_observations$ is the number of observations and the optional parameters correspond to keywords *Icen*, *Ilt*, *Irt*, *Ifreq*, and *Ifix*.

The columns must be ordered such that the first n_class columns contain data for the class variables, the next $n\_continuous$ columns contain data for the continuous variables, and the next column contains the response variable. The final (and optional) $m - 1$ columns contain optional parameters.

Keywords

CASE_ANALYSIS

Named variable into which a two-dimensional array of size $n\_observations$ by 5 containing the case analysis below is stored:

- 0—Estimated predicted value.
- 1—Estimated influence or leverage.

- 2—Estimated residual.
- 3—Estimated cumulative hazard.
- 4—Non-censored observation: Estimated density at the observation failure time and covariate values. Censored observations: The corresponding estimated probability.

CLASS_VALS

Named variable into which one-dimensional array of length:

$$\sum_{i=0}^{n_{class}-1} N\_Class\_Vals(i)$$

containing the distinct values of the classification variables in ascending order is stored. The first $N\_Class\_Vals(0)$ elements of $Class\_Vals$ contain the values for the first classification variables, the next $N\_Class\_Vals(1)$ elements contain the values for the second classification variable, etc.

COEF_STAT

Named variable into which a two-dimensional array of size $n_{coefficients}$ by 4 containing the parameter estimates and associated statistics is stored:

- 0—Coefficient estimate.
- 1—Estimated standard deviation of the estimated coefficient.
- 2—Asymptotic normal score for testing that the coefficient is zero.
- 3—The p -value associated with the normal score in Column 2.

When present in the model, the first coefficient in $Coef\_Stat$ is the estimate of the “nuisance” parameter, and the remaining coefficients are estimates of the parameters associated with the “linear” model, beginning with the intercept, if present. Nuisance parameters are as follows:

- 0—No nuisance parameter
- 1—Coefficient of the quadratic term, term in time, θ
- 2–9—Scale parameter, σ
- 10—Scale parameter, θ

COVARIANCES

Named variable into which a two-dimensional array of size $n\_coefficients$ by $n\_coefficients$ containing the estimated asymptotic covariance matrix of the coefficients is stored. For $Itmax = 0$, this is the Hessian computed at the initial parameter estimates.

CRITERION

Named variable into which the optimized criterion is stored. The criterion to be maximized is a constant plus the log-likelihood.

DOUBLE

If present and nonzero, double precision is used.

EPS

Convergence criterion. Convergence is assumed when maximum relative change in any coefficient estimate is less than Eps from one iteration to the next or when the relative change in the log-likelihood, criterion, from one iteration to the next is less than $Eps/100.0$. Default: $Eps = 0.001$

EST_DELTA

Increment between time points on the time grid. Keywords *Est_Delta*, *Est_Nobs*, *Est_Time*, *Est_Npt*, and *Est_Prob* must be used together.

EST_NOBS

Number of observations for which estimates are to be calculated. *Est_Nobs* must be positive. Keywords *Est_Nobs*, *Est_Time*, *Est_Npt*, *Est_Delta*, and *Est_Prob* must be used together.

EST_NPT

Number of points on the time grid for which survival probabilities are desired. *Est_Npt* must be positive. Keywords *Est_Npt*, *Est_Nobs*, *Est_Time*, *Est_Delta*, and *Est_Prob* must be used together.

EST_PROB

Named variable into which a two-dimensional array of size Est_Npt by $(2 * n\_observations + 1)$ containing the estimated survival probabilities for the covariate groups specified in x is stored. Column 0 contains the survival time.

Columns 1 and 2 contain the estimated survival probabilities and hazard rates, respectively, for the covariates in the first row of x . In general, the survival and hazard row i of x is contained in columns $2i - 1$ and $2i$, respectively, for $i = 1, 2, \dots, Est\_Npt$. Keywords *Est_Prob*, *Est_Nobs*, *Est_Time*, *Est_Npt*, and *Est_Delta* must be used together.

EST_TIME

Beginning of the time grid for which estimates are desired. Survival probabilities and hazard rates are computed for each covariate vector over the grid of time points $Est\_Time + i * Est\_Delta$ for $i = 0, 1, \dots, Est\_Npt - 1$. Keywords *Est_Time*, *Est_Nobs*, *Est_Npt*, *Est_Delta*, and *Est_Prob* must be used together.

EST_XBETA

Named variable into which an one-dimensional array of length $n\_observations$ containing the estimated linear response:

$$w + x\hat{\beta}$$

for each row of x is stored. To use keyword *Est_Xbeta*, you must also use keywords *Est_Nobs*, *Est_Time*, *Est_Npt*, *Est_Delta*, and *Est_Prob*.

ICEN

The column in x containing the censoring code for each observation. Possible values are shown in [Table 22-1](#).

$x(i, Icen)$	Censoring type
0	Exact failure at $x(i, Irt)$
1	Right Censored. The response is greater than $x(i, Irt)$
2	Left Censored. The response is less than or equal to $x(i, Irt)$
3	Interval Censored. The response is greater than $x(i, Irt)$, but less than or equal to $x(i, Irt)$.

Table 22-1: *Icen* Values

IFIX

Column number in x containing a fixed parameter for each observation that is added to the linear response prior to computing the model parameter. The “fixed” parameter allows one to test hypothesis about the parameters via the log-likelihoods.

IFREQ

The column number of x containing the frequency of response for each observation.

ILT

The column number of x containing the upper endpoint of the failure interval for interval- and left-censored observations.

INDICES_EFFECTS

One-dimensional index array of length $Var\_Effects(0) + Var\_Effects(1) + \dots + Var\_Effects(n\_effects - 1)$. The first $Var\_Effects(0)$ elements give the column numbers of x for each variable in the first effect. The next $Var\_Effects(1)$ elements give the column numbers for each variable in the second effect. The last $Var\_Effects(n\_effects - 1)$ elements give the column numbers for each variable in the last effect. Keywords *Indices_Effects* and *Var_Effects* must be used together.

INIT_EST

One-dimensional array containing the initial estimates of the parameters (which requires that you know the number of coefficients in the model prior to the use of *IMSL_SURVIVAL_GLM*). See output keyword *Coef_Stat* for a description of the “nuisance” parameter, which is the first element of array *Init_Est*. By default, un-weighted linear regression is used to obtain initial estimates.

IRT

The column number of x containing the lower endpoint of the failure interval for interval- and right-censored observations.

ITERATIONS

Named variable into which a two-dimensional array of size, n by 5 containing information about each iteration of the analysis is stored, where n is equal to the number of iterations. This is shown in [Table 22-2](#).

Column	Statistic
0	Method of iteration Q-N Step = 0 N-R Step = 1
1	Iteration number
2	Step size
3	Maximum scaled coefficient update
4	Log-likelihood

Table 22-2: Column Information

ITMAX

Maximum number of iterations. Use *Itmax* = 0 to compute the Hessian, stored in *Covariances*, and the Newton step, stored in *Last_Step*, at the initial estimates (The initial estimates must be input. Use keyword *Init_Est*). See Example 3. Default: *Itmax* = 30

LAST_STEP

Named variable into which an one-dimensional array of length $n\_coefficients$ containing the last parameter updates (excluding step halvings) is stored. Keyword *Last_Step* is computed as the inverse of the matrix of second partial derivatives times the vector of first partial derivatives of the log-likelihood. When *Itmax* = 0, the derivatives are computed at the initial estimates.

LP_MAX

Remove a right- or left-censored observation from the log-likelihood whenever the probability of the observation exceeds 0.995. At convergence, use linear programming to check that all removed observations actually have infinite linear response:

$$z_i \hat{\beta}$$

Obs_Status(i) is set to 2 if the linear response is infinite (See keyword *Obs_Status*). If not all removed observations have infinite linear response, recompute the estimates based upon the observations with finite:

$$z_i \hat{\beta}$$

Keyword *Lp_Max* is the maximum number of observations that can be handled in the linear programming. Setting *Lp_Max* = *n_observations* is always sufficient. By default, the function iterates without checking for infinite estimates. Default: No infinity checking; *Lp_Max* = 0

MAX_CLASS

An upper bound on the sum of the number of distinct values taken on by each classification variable. Internal workspace usage can be significantly reduced with an appropriate choice of *Max_Class*. Default: *Max_Class* = *n_observations* * *n_class*

MEANS

Named variable into which an one-dimensional array containing the means of the design variables is stored. The array is of length *n_coefficients* – *k* if keyword *No_Intercept* is used, and of length *n_coefficients* – *k* – 1 otherwise. Here, *k* is equal to 0 if *model* = 0, and equal to 1 otherwise.

N_CLASS_VALS

Named variable into which an one-dimensional array of length *n_class* containing the number of values taken by each classification variable is stored; the *i*-th classification variable has *N_Class_Vals(i)*.

NMISSING

Named variable into which the number of rows of data that contain missing values in one or more of the following vectors or columns of *x* is stored: *Icen*, *Ilt*, *Irt*, *Ifreq*, *Ifix*, or *Indicies_Effects*.

NO_INTERCEPT

If present and nonzero, there is no intercept in the model. By default, the intercept is automatically included in the model.

OBS_STATUS

Named variable into which an one-dimensional array of length *n_observations* indicating which observations are included in the extended likelihood is stored.

- 0—Observation i is in the likelihood
- 1—Observation i cannot be in the likelihood because it contains at least one missing value in x .
- 2—Observation i is not in the likelihood. Its estimated parameter is infinite.

VAR_EFFECTS

One-dimensional array of length `n_effects` containing the number of variables associated with each effect in the model, where `n_effects` is the number of effects (sources of variation) in the model. Keywords *Var_Effects* and *Indicies_Effects* must be used together.

Comments

1. Dummy variables are generated for the classification variables as follows: An ascending list of all distinct values of each classification variable is obtained and stored in *Class_Vals*. Dummy variables are then generated for each but the last of these distinct values. Each dummy variable is zero unless the classification variable equals the list value corresponding to the dummy variable, in which case the dummy variable is one. See keyword *Dummy_Method* in the “[IMSL_REGRESSORS](#)” on page 600.
2. The “product” of a classification variable with a covariate yields dummy variables equal to the product of the covariate with each of the dummy variables associated with the classification variable.
3. The “product” of two classification variables yields dummy variables in the usual manner. Each dummy variable associated with the first classification variable multiplies each dummy variable associated with the second classification variable. The resulting dummy variables are such that the index of the second classification variable varies fastest.

Discussion

The `IMSL_SURVIVAL_GLM` function computes the maximum likelihood estimates of parameters and associated statistics in generalized linear models commonly found in survival (reliability) analysis. Although the terminology used will be from the survival area, the methods discussed have applications in many areas of data analysis, including reliability analysis and event history analysis. These methods can be used anywhere a random variable from one of the discussed distributions is parameterized via one of the models available in `IMSL_SURVIVAL_GLM`. Thus, while it is not advisable to do so, standard multiple linear regression can be performed by routine `IMSL_SURVIVAL_GLM`. Estimates for any of 10 standard models can be computed.

Exact, left-censored, right-censored, or interval-censored observations are allowed (note that left censoring is the same as interval censoring with the left endpoint equal to the left endpoint of the support of the distribution).

Let $\eta = x^T \beta$ be the linear parameterization, where x is a design vector obtained by `IMSL_SURVIVAL_GLM` via `IMSL_REGRESSORS` from a row of x , and β is a vector of parameters associated with the linear model. Let T denote the random response variable and $S(t)$ denote the probability that $T > t$. All models considered also allow a fixed parameter w_i for observation i (input in column *Ifix* of x). Use of this parameter is discussed below. There also may be nuisance parameters $\theta > 0$, or $\sigma > 0$ to be estimated (along with β) in the various models. Let Φ denote the cumulative normal distribution. The survival models available in `IMSL_SURVIVAL_GLM` are listed in [Table 22-3](#).

model	Name	$S(t)$
0	Exponential	$\exp[-t \exp(w_i + \eta)]$
1	Linear hazard	$\exp\left[-\left(t + \frac{\theta t^2}{2}\right) \exp(w_i + \eta)\right]$
2	Log-normal	$1 - \Phi\left(\frac{\ln(t) - \eta - w_i}{\sigma}\right)$
3	Normal	$1 - \Phi\left(\frac{t - \eta - w_i}{\sigma}\right)$
4	Log-logistic	$\left\{1 + \exp\left(\frac{\ln(t) - \eta - w_i}{\sigma}\right)\right\}^{-1}$

Table 22-3: Available Survival Models

model	Name	$S(t)$
5	Logistic	$\left\{ 1 + \exp\left(\frac{t - \eta - w_i}{\sigma}\right) \right\}^{-1}$
6	Log least extreme value	$\exp\left\{-\exp\left(\frac{\ln(t) - \eta - w_i}{\sigma}\right)\right\}$
7	Least extreme value	$\exp\left\{-\exp\left(\frac{t - \eta - w_i}{\sigma}\right)\right\}$
8	Log extreme value	$1 - \exp\left\{-\exp\left(\frac{\ln(t) - \eta - w_i}{\sigma}\right)\right\}$
9	Extreme value	$1 - \exp\left\{-\exp\left(\frac{t - \eta - w_i}{\sigma}\right)\right\}$
10	Weibull	$\exp\left\{-\left[\frac{t}{\exp(w_i + \eta)}\right]^\theta\right\}$

Table 22-3: Available Survival Models (Continued)

Note that the log-least-extreme-value model is a re-parameterization of the Weibull model. Moreover, models 0, 1, 2, 4, 6, 8, and 10 require that $T > 0$, while all of the remaining models allow any value for T , $-\infty < T < \infty$.

Each row vector in the data matrix can represent a single observation; or, through the use of vector frequencies, each row can represent several observations. Also note that

classification variables and their products are easily incorporated into the models via the usual regression-type specifications.

The constant parameter W_i is input in x and may be used for a number of purposes. For example, if the parameter in an exponential model is known to depend upon the size of the area tested, volume of a radioactive mass, or population density, and so on, then a multiplicative factor of the exponential parameter $\lambda = \exp(x\beta)$ may be known beforehand. This factor can be input in W_i (W_i is the log of the factor).

An alternate use of W_i is as follows: It may be that $\lambda = \exp(x_1\beta_1 + x_2\beta_2)$, where β_2 is known. Letting $W_i = x_2\beta_2$, estimates for β_1 can be obtained via `IMSL_SURVIVAL_GLM` with the known fixed values for β_2 . Standard methods can then be used to test hypothesis about β_1 via computed log-likelihoods.

Computational Details

The computations proceed as follows:

1. The input parameters are checked for consistency and validity.
 - Estimates of the means of the “independent” or design variables are computed. Means are computed as:

$$\bar{x} = \frac{\sum f_i x_i}{\sum f_i}$$

2. If initial estimates are not provided (see keyword *Init_Est*), they are calculated as follows:

Models 2-10:

- A. Kaplan-Meier estimates of the survival probability:

$$\hat{S}(t)$$

at the upper limit of each failure interval are obtained. (Because upper limits are used, interval- and left-censored data are assumed to be exact failures at the upper endpoint of the failure interval.) The Kaplan-Meier estimate is computed under the assumption that all failure distributions are identical (i.e., all β 's but the intercept, if present, are assumed to be zero).

- B. If there is an intercept in the model, a simple linear regression is performed predicting:

$$S^{-1}(\hat{S}(t)) - w_i = \alpha + \phi t'$$

where t' is computed at the upper endpoint of each failure interval, $t' = t$ in models 3, 5, 7, and 9, and $t' = \ln(t)$ in models 2, 4, 6, 8, and 10, and w_i is the fixed constant, if present.

If there is no intercept in the model, then α is fixed at zero, and the model:

$$S^{-1}(\hat{S}(t)) - \hat{\phi} t' - w_i = x^T \beta$$

is fit instead. In this model, the coefficients β are used in place of the location estimate α above. Here:

$$\hat{\phi}$$

is estimated from the simple linear regression with $\alpha = 0$.

- C. If the intercept is in the model, then in log-location-scale models (models 1-8):

$$\hat{\sigma} = \hat{\phi}$$

and the initial estimate of the intercept is assumed to be:

$$\hat{\alpha}$$

In the Weibull model:

$$\hat{\theta} = 1/\hat{\phi}$$

and the intercept is assumed to be:

$$\hat{\alpha}$$

Initial estimates of all parameters β , other than the intercept, are assumed to be zero.

If there is no intercept in the model, the scale parameter is estimated as above, and the estimates:

$$\hat{\beta}$$

from Step 2 are used as initial estimates for the β 's.

Models 0 and 1:

For the exponential models (*model* = 0 or 1), the “average total time on” test statistic is used to obtain an estimate for the intercept. Specifically, let T_t denote the total number of failures divided by the total time on test. The initial estimates for the intercept is then $\ln(T_t)$. Initial estimates for the remaining parameters β are assumed to be zero, and if *model* = 1, the

initial estimate for the linear hazard parameter θ is assumed to be a small positive number. When the intercept is not in the model, the initial estimate for the parameter θ is assumed to be a small positive number, and initial estimates of the parameters β are computed via multiple linear regression as in Part A.

3. A quasi-Newton algorithm is used in the initial iterations based on a Hessian estimate:

$$\hat{H}_{\kappa j \kappa l} = \sum_i l'_{i\alpha_j i \alpha_l}$$

where $l'_{i\alpha_j}$ is the partial derivative of the i -th term in the log-likelihood with respect to the parameter α_j , and α_j denotes one of the parameters to be estimated.

When the relative change in the log-likelihood from one iteration to the next is 0.1 or less, exact second partial derivatives are used for the Hessian so the Newton-Rapheson iteration is used.

If the initial step size results in an increase in the log-likelihood, the full step is used. If the log-likelihood decreases for the initial step size, the step size is halved, and a check for an increase in the log-likelihood performed. Step-halving is performed (as a simple line search) until an increase in the log-likelihood is detected, or until the step size becomes very small (the initial step size is 1.0).

4. Convergence is assumed when the maximum relative change in any coefficient update from one iteration to the next is less than *Eps* or when the relative change in the log-likelihood from one iteration to the next is less than *Eps*/100. Convergence is also assumed after *Itmax* iterations or when step halving leads to a very small step size with no increase in the log-likelihood.
5. If requested (see keyword *Lp_Max*), the methods of Clarkson and Jennrich (1988) are used to check for the existence of infinite estimates in:

$$\eta_i = x_i^T \beta$$

As an example of a situation in which infinite estimates can occur, suppose that observation j is right-censored with $t_j > 15$ in a normal distribution model in which the mean is:

$$\mu_j = x_j^T \beta = \eta_j$$

where x_j is the observation design vector. If the design vector x_j for parameter β_m is such that $x_{jm} = 1$ and $x_{im} = 0$ for all $i \neq j$, then the optimal estimate of β_m occurs at:

$$\hat{\beta}_m = \infty$$

leading to an infinite estimate of both β_m and η_j . In `IMSL_SURVIVAL_GLM`, such estimates can be “computed.”

In all models fit by `IMSL_SURVIVAL_GLM`, infinite estimates can only occur when the optimal estimated probability associated with the left- or right-censored observation is 1. If infinity checking is on, left- or right-censored observations that have estimated probability greater than 0.995 at some point during the iterations are excluded from the log-likelihood, and the iterations proceed with a log-likelihood based on the remaining observations. This allows convergence of the algorithm when the maximum relative change in the estimated coefficients is small and also allows for a more precise determination of observations with infinite:

$$\eta_i = x_i^T \beta$$

At convergence, linear programming is used to ensure that the eliminated observations have infinite η_i . If some (or all) of the removed observations should not have been removed (because their estimated η_i 's must be finite), then the iterations are restarted with a log-likelihood based upon the finite η_i observations. See Clarkson and Jennrich (1988) for more details.

By default, or when not using keyword `Lp_Max` (see keyword `Lp_Max`), no observations are eliminated during the iterations. In this case, the infinite estimates occur, some (or all) of the coefficient estimates:

$$\hat{\beta}$$

will become large, and it is likely that the Hessian will become (numerically) singular prior to convergence.

6. The case statistics are computed as follows: Let $I_i(\theta_i)$ denote the log-likelihood of the i -th observation evaluated at θ_i , let I'_i denote the vector of derivatives of I_i with respect to all parameters, I'_{η_i} denote the derivative of I_i with respect to $\eta = x^T \beta$, H denote the Hessian, and E denote expectation. Then the columns of *Case_Analysis* are:
 - A. Predicted values are computed as $E(T/x)$ according to standard formulas. If model is 4 or 8, and if $s \geq 1$, then the expected values cannot be computed because they are infinite.

- B. Following Cook and Weisberg (1982), the influence (or leverage) of the i -th observation is assumed to be:

$$(\mathbf{I}'_i)^T \mathbf{H}^{-1} \mathbf{I}'_i$$

This quantity is a one-step approximation of the change in the estimates when the i -th observation is deleted (ignoring the nuisance parameters).

- C. The “residual” is computed as $\mathbf{I}'_{\eta,i}$.
- D. The cumulative hazard is computed at the observation covariate values and, for interval observations, the upper endpoint of the failure interval. The cumulative hazard also can be used as a “residual” estimate. If the model is correct, the cumulative hazards should follow a standard exponential distribution. See Cox and Oakes (1984).

The IMSL_SURVIVAL_GLM function computes estimates of survival probabilities and hazard rates for the parametric survival/reliability models when using the *Est_** keywords.

Let $\eta = \mathbf{x}^T \boldsymbol{\beta}$ be the linear parameterization, where $\mathbf{x}$ is the design vector corresponding to a row of $\mathbf{x}$ (IMSL_SURVIVAL_GLM generates the design vector using IMSL_REGRESSORS), and $\boldsymbol{\beta}$ is a vector of parameters associated with the linear model. Let T denote the random response variable and $S(t)$ denote the probability that $T > t$. All models considered also allow a fixed parameter w (input in column *Ifix* of $\mathbf{x}$). Use of the keyword is discussed in above. There also may be nuisance parameters $\theta > 0$ or $\sigma > 0$. Let $\lambda(t)$ denote the hazard rate at time t . Then $\lambda(t)$ and $S(t)$ are related at:

$$S(t) = \exp\left(-\int_{-\infty}^t \lambda(s) ds\right)$$

Models 0, 1, 2, 4, 6, 8, and 10 require that $T > 0$ (in which case assume $\lambda(s) = 0$ for $s < 0$), while the remaining models allow arbitrary values for T , $-\infty < T < \infty$. The computations proceed in IMSL_SURVIVAL_GLM when using the keywords *Est_** as follows:

1. The input arguments are checked for consistency and validity.
2. For each row of $\mathbf{x}$, the explanatory variables are generated from the classification and variables and the covariates using IMSL_REGRESSORS with keyword *Dummy_Method*.
3. For each point requested in the time grid, the survival probabilities and hazard rates are computed.

Programming Notes

Indicator (dummy) variables are created for the classification variables using `IMSL_REGRESSORS` (Chapter 3, *Regression*) using keyword *Dummy_Method*.

Examples

The following four examples all use the array `x`, defined as follows:

```
x1 = [[1.0, 0.0, 7.0, 64.0, 5.0, 411.0, 0.0] , $
      [1.0, 0.0, 6.0, 63.0, 9.0, 126.0, 0.0] , $
      [1.0, 0.0, 7.0, 65.0, 11.0, 118.0, 0.0] , $
      [1.0, 0.0, 4.0, 69.0, 10.0, 92.0, 0.0] , $
      [1.0, 0.0, 4.0, 63.0, 58.0, 8.0, 0.0] , $
      [1.0, 0.0, 7.0, 48.0, 9.0, 25.0, 1.0] , $
      [1.0, 0.0, 7.0, 48.0, 11.0, 11.0, 0.0] , $
      [2.0, 0.0, 8.0, 63.0, 4.0, 54.0, 0.0] , $
      [2.0, 0.0, 6.0, 63.0, 14.0, 153.0, 0.0] , $
      [2.0, 0.0, 3.0, 53.0, 4.0, 16.0, 0.0]] ; , $
x2 = [[2.0, 0.0, 8.0, 43.0, 12.0, 56.0, 0.0] , $
      [2.0, 0.0, 4.0, 55.0, 2.0, 21.0, 0.0] , $
      [2.0, 0.0, 6.0, 66.0, 25.0, 287.0, 0.0] , $
      [2.0, 0.0, 4.0, 67.0, 23.0, 10.0, 0.0] , $
      [3.0, 0.0, 2.0, 61.0, 19.0, 8.0, 0.0] , $
      [3.0, 0.0, 5.0, 63.0, 4.0, 12.0, 0.0] , $
      [4.0, 0.0, 5.0, 66.0, 16.0, 177.0, 0.0] , $
      [4.0, 0.0, 4.0, 68.0, 12.0, 12.0, 0.0] , $
      [4.0, 0.0, 8.0, 41.0, 12.0, 200.0, 0.0] , $
      [4.0, 0.0, 7.0, 53.0, 8.0, 250.0, 0.0]] ; , $
x3 = [[4.0, 0.0, 6.0, 37.0, 13.0, 100.0, 0.0] , $
      [1.0, 1.0, 9.0, 54.0, 12.0, 999.0, 0.0] , $
      [1.0, 1.0, 5.0, 52.0, 8.0, 231.0, 1.0] , $
      [1.0, 1.0, 7.0, 50.0, 7.0, 991.0, 0.0] , $
      [1.0, 1.0, 2.0, 65.0, 21.0, 1.0, 0.0] , $
      [1.0, 1.0, 8.0, 52.0, 28.0, 201.0, 0.0] , $
      [1.0, 1.0, 6.0, 70.0, 13.0, 44.0, 0.0] , $
      [1.0, 1.0, 5.0, 40.0, 13.0, 15.0, 0.0] , $
      [2.0, 1.0, 7.0, 36.0, 22.0, 103.0, 1.0] , $
      [2.0, 1.0, 4.0, 44.0, 36.0, 2.0, 0.0]] ; , $
x4 = [[2.0, 1.0, 3.0, 54.0, 9.0, 20.0, 0.0] , $
      [2.0, 1.0, 3.0, 59.0, 87.0, 51.0, 0.0] , $
      [3.0, 1.0, 4.0, 69.0, 5.0, 18.0, 0.0] , $
      [3.0, 1.0, 6.0, 50.0, 22.0, 90.0, 0.0] , $
      [3.0, 1.0, 8.0, 62.0, 4.0, 84.0, 0.0] , $
      [4.0, 1.0, 7.0, 68.0, 15.0, 164.0, 0.0] , $
      [4.0, 1.0, 3.0, 39.0, 4.0, 19.0, 0.0] , $
      [4.0, 1.0, 6.0, 49.0, 11.0, 43.0, 0.0] , $
      [4.0, 1.0, 8.0, 64.0, 10.0, 340.0, 0.0] , $
```

```

[4.0, 1.0, 7.0, 67.0, 18.0, 231.0, 0.0]]
x = [[x1], [x2], [x3], [x4]]
x = TRANSPOSE(x)

```

Example 1

This example is taken from Lawless (1982, p. 287) and involves the mortality of patients suffering from lung cancer. An exponential distribution is fit for the model:

$$\eta = \mu + \alpha_i + \gamma_k + \beta_6 x_3 + \beta_7 x_4 + \beta_8 x_5$$

where α_i is associated with a classification variable with four levels, and γ_k is associated with a classification variable with two levels. Note that because the computations are performed in single precision, there will be some small variation in the estimated coefficients across different machine environments.

```

.RUN
PRO print_results, cs
  PRINT, '          Coefficient Statistics'
  PRINT, '          Coefficient      s.e          z          p'
  PM, cs, FORMAT = '(4F14.4)'
END

n_class = 2
n_continuous = 3
model = 0
icen = 6
irt = 5
lp_max = 40
n_coef = IMSL_SURVIVAL_GLM(n_class, n_continuous, model, x, $
  ICEN = icen, IRT = irt, LP_MAX = lp_max, COEF_STAT = cs)
print_results, cs

Coefficient Statistics
Coefficient      s.e          z          p
-1.1027         1.3091       -0.8423       0.3998
-0.3626         0.4446       -0.8156       0.4149
0.1271          0.4863        0.2613       0.7939
0.8690          0.5861        1.4825       0.1385
0.2697          0.3882        0.6948       0.4873
-0.5400         0.1081       -4.9946       0.0000
-0.0090         0.0197       -0.4594       0.6460
-0.0034         0.0117       -0.2912       0.7710

```

Example 2

This example uses the same array x defined in Example 1, but more optional arguments are demonstrated.

```
.RUN
PRO print_results, cs, iter, crit, nmiss
  PRINT, '          Coefficient Statistics'
  PRINT, '      Coefficient      s.e          z          p'
  PM, cs, FORMAT = '(4F14.4)'
  PRINT
  PRINT, '          Iteration Information'
  PRINT, 'Method  Iteration  Step Size    Coef Update  ', $
    'Log-Likelihood'
  PM, iter, FORMAT = '(I3, I10, 2F14.4, F14.1)'
  PRINT
  PRINT, 'Log-Likelihood =', crit
  PRINT
  PRINT, 'Number of Missing Value = ', nmiss, $
    FORMAT = '(A26, I3)'
END

n_class = 2
n_continuous = 3
model = 0
icen = 6
irt = 5
lp_max = 40
n_coef = IMSL_SURVIVAL_GLM(n_class, n_continuous, model, x, $
  ICEN = icen, IRT = irt, LP_MAX = lp_max, $
  N_CLASS_VALS = ncv, CLASS_VALS = cv, $
  COEF_STAT = cs, CRITERION = crit, $
  MEANS = means, CASE_ANALYSIS = ca, $
  ITERATIONS = iter, OBS_STATUS = os, NMISSING = nmiss)
print_results, cs, iter, crit, nmiss
```

```
Coefficient Statistics
Coefficient      s.e          z          p
-1.1027         1.3091        -0.8423        0.3998
-0.3626         0.4446        -0.8156        0.4149
 0.1271         0.4863         0.2613        0.7939
 0.8690         0.5861         1.4825        0.1385
 0.2697         0.3882         0.6948        0.4873
-0.5400         0.1081        -4.9946        0.0000
-0.0090         0.0197        -0.4594        0.6460
-0.0034         0.0117        -0.2912        0.7710
```

```
Iteration Information
Method  Iteration  Step Size    Coef Update  Log-Likelihood
  0         0         NaN          NaN         -224.0
  0         1         1.0000         0.9839        -213.4
  1         2         1.0000         3.6034        -207.3
  1         3         1.0000        10.1238        -204.3
  1         4         1.0000         0.1430        -204.1
  1         5         1.0000         0.0117        -204.1
```

```
Log-Likelihood =      -204.139
Number of Missing Value =    0
```

Example 3

In this example, the same data and model as example 1 are used, but *Itmax* is set to zero iterations with model coefficients restricted such that $\mu = -1.25$, $\beta_6 = -0.6$, and the remaining six coefficients are equal to zero. A chi-squared statistic, with 8 degrees of freedom for testing the coefficients is specified as above (versus the alternative that it is not as specified), can be computed, based on the output, as:

$$\chi^2 = \mathbf{g}^T \hat{\Sigma}^{-1} \mathbf{g}$$

where:

$$\hat{\Sigma}$$

is output in *Covariances*. The resulting test statistic, $\chi^2 = 6.107$, based upon no iterations is comparable to likelihood ratio test that can be computed from the log-likelihood output in this example (-206.683) and the log-likelihood output in Example 2 (-204.139).

```
.RUN
PRO print_results, cs, means, cov, crit, ls
  PRINT, '          Coefficient Statistics'
  PRINT, '      Coefficient      s.e          z          p'
  PM, cs, FORMAT = '(4F14.4)'
  PRINT
  PRINT, '          Covariate Means'
  PRINT, means, FORMAT = '(7F8.2)'
  PRINT
  PRINT, '          Hessian'
  PM, cov, FORMAT = '(8F8.4)'
  PRINT
  PRINT, 'Log-Likelihood =', crit
  PRINT
  PRINT, '          Newton_Raphson Step'
  PRINT, ls, FORMAT = '(8F8.4)'
END

n_class = 2
n_continuous = 3
model = 0
icen = 6
irt = 5
lp_max = 40
itmax = 0
init_est = [-1.25, 0.0, 0.0, 0.0, 0.0, -0.6, 0.0, 0.0]
n_coef = IMSL_SURVIVAL_GLM(n_class, n_continuous, model, x, $
```

```

ICEN = icen, IRT = irt, ITMAX = itmax, $
LP_MAX = lp_max, INIT_EST = init_est, $
COEF_STAT = cs, CRITERION = crit, $
COVARIANCES = cov, MEANS = means, LAST_STEP = ls)
print_results, cs, means, cov, crit, ls

```

Coefficient Statistics			
Coefficient	s.e	z	p
-1.2500	1.3773	-0.9076	0.3643
0.0000	0.4288	0.0000	1.0000
0.0000	0.5299	0.0000	1.0000
0.0000	0.7748	0.0000	1.0000
0.0000	0.4051	0.0000	1.0000
-0.6000	0.1118	-5.3652	0.0000
0.0000	0.0215	0.0000	1.0000
0.0000	0.0109	0.0000	1.0000

Covariate Means						
0.35	0.28	0.12	0.53	5.65	56.58	15.65

Hessian							
1.8969	-0.0906	-0.1641	-0.1681	0.0778	-0.0818	-0.0235	-0.0012
-0.0906	0.1839	0.0996	0.1191	0.0358	-0.0005	-0.0008	0.0006
-0.1641	0.0996	0.2808	0.1264	-0.0226	0.0104	0.0005	-0.0021
-0.1681	0.1191	0.1264	0.6003	0.0460	0.0193	-0.0016	0.0007
0.0778	0.0358	-0.0226	0.0460	0.1641	0.0060	-0.0040	0.0017
-0.0818	-0.0005	0.0104	0.0193	0.0060	0.0125	0.0000	0.0003
-0.0235	-0.0008	0.0005	-0.0016	-0.0040	0.0000	0.0005	-0.0001
-0.0012	0.0006	-0.0021	0.0007	0.0017	0.0003	-0.0001	0.0001

Log-Likelihood = -206.683

Newton_Raphson Step							
0.1706	-0.3365	0.1333	1.2967	0.2985	0.0625	-0.0112	-0.0026

Example 4

This example is a continuation of the first example above. Keywords *Est_** are used in `IMSL_SURVIVAL_GLM` to compute the parameter estimates. The example is taken from Lawless (1982, p. 287) and involves the mortality of patients suffering from lung cancer.

```

.RUN
PRO print_results, ep
  PRINT, '          Survival and Hazard Estimates'
  PRINT, '   Time          S1          H1          S2          H2'
  PM, ep, FORMAT = '(F7.2, F10.4, F13.6, F10.4, F13.6)'
END

```

```

n_class = 2
n_continuous = 3
model = 0
icen = 6
irt = 5
lp_max = 40
time = 10.0
npt = 10
delta = 20.0
n_coef = IMSL_SURVIVAL_GLM(n_class, n_continuous, model, x, $
    ICEN=icen, IRT=irt, LP_MAX=lp_max, N_CLASS_VALS=nvc, $
    CLASS_VALS=cv, COEF_STAT=cs, CRITERION=crit, MEANS=means, $
    CASE_ANALYSIS=ca, OBS_STATUS=os, ITERATIONS=iter, $
    EST_NOBS=2, EST_TIME=time, EST_NPT=npt, $
    EST_DELTA=delta, EST_PROB=ep, EST_XBETA=xb)
print_results, ep

```

Survival and Hazard Estimates				
Time	S1	H1	S2	H2
10.00	0.9626	0.003807	0.9370	0.006503
30.00	0.8921	0.003807	0.8228	0.006503
50.00	0.8267	0.003807	0.7224	0.006503
70.00	0.7661	0.003807	0.6343	0.006503
90.00	0.7099	0.003807	0.5570	0.006503
110.00	0.6579	0.003807	0.4890	0.006503
130.00	0.6096	0.003807	0.4294	0.006503
150.00	0.5649	0.003807	0.3770	0.006503
170.00	0.5235	0.003807	0.3310	0.006503
190.00	0.4852	0.003807	0.2907	0.006503

Errors

Warning Errors

STAT_CONVERGENCE_ASSUMED_1—Too many step halvings. Convergence is assumed.

STAT_CONVERGENCE_ASSUMED_2—Too many step iterations. Convergence is assumed.

STAT_NO_PREDICTED_1—“estimates(0)” > 1.0. The expected value for the log logistic distribution (“model” = 4) does not exist. Predicted values will not be calculated.

STAT_NO_PREDICTED_2—“estimates(0)” > 1.0. The expected value for the log extreme value distribution (“model” = 8) does not exist. Predicted values will not be calculated.

STAT_NEG_EIGENVALUE—The Hessian has at least one negative eigenvalue. An upper bound on the absolute value of the minimum eigenvalue is # corresponding to variable index #.

STAT_INVALID_FAILURE_TIME_4—“x(#)(“Ilt”= #)” = # and “x(#) (“Irt”= #)” = #. The censoring interval has length 0.0. The censoring code for this observation is being set to 0.0.

Fatal Error

STAT_MAX_CLASS_TOO_SMALL—The number of distinct values of the classification variables exceeds “Max_Class” = #.

STAT_TOO_FEW_COEF—Init_Est is specified, and “Init_Est” = #. The model specified requires # coefficients.

STAT_TOO_FEW_VALID_OBS—“n_observations” = # and “Nmissing” = #. “n_observations”(“Nmissing” must be greater than or equal to 2 in order to estimate the coefficients.

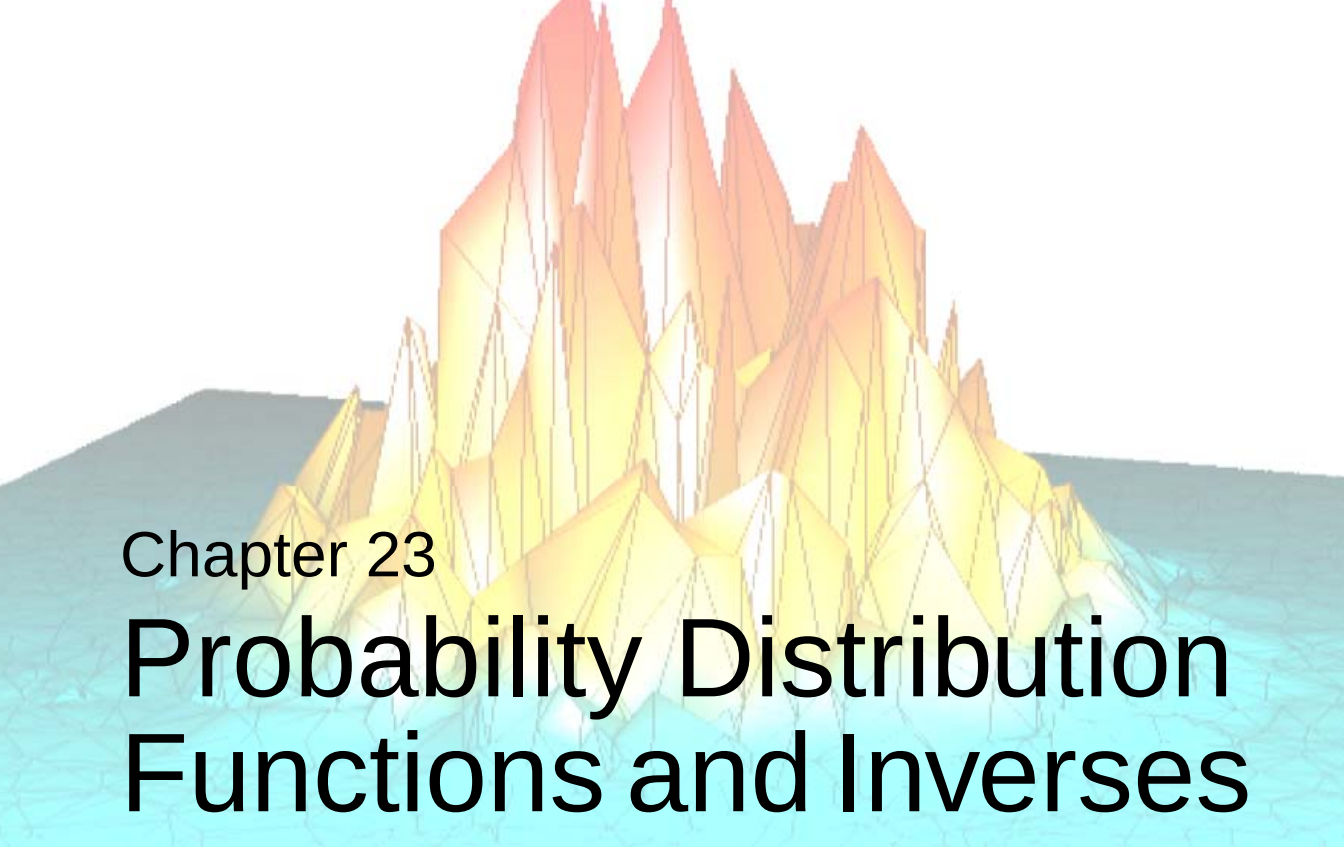
STAT_SVGLM_1—For the exponential model (“model” = 0) with “n_effects” = # and no intercept, “n_coef” has been determined to equal 0. With no coefficients in the model, processing cannot continue.

STAT_INCREASE_LP_MAX—Too many observations are to be deleted from the model. Either use a different model or increase the workspace.

STAT_INVALID_DATA_8—“Class_Vals(#)” = #. The number of distinct values for each classification variable must be greater than one.

Version History

6.4	Introduced



Chapter 23

Probability Distribution Functions and Inverses

This section contains the following topics:

Overview: Probability Distribution Functions
and Inverses 1030

Probability Distribution Functions and Inverses
Routines 1031

Overview: Probability Distribution Functions and Inverses

This chapter describes probability distribution functions and inverses included in IDL Advanced Math and Stats. See [“Probability Distribution Functions and Inverses Routines”](#) on page 1031 for a list of the included routines.

Probability Distribution Functions and Inverses Routines

[IMSL_NORMALCDF](#)—Normal (Gaussian) distribution function.

[IMSL_BINORMALCDF](#)—Bivariate normal distribution.

[IMSL_CHISQCDF](#)—Chi-squared distribution function.

[IMSL_FCDF](#)— F distribution function.

[IMSL_TCDF](#)—Student's t distribution function.

[IMSL_GAMMACDF](#)—Gamma distribution function.

[IMSL_BETACDF](#)—Beta distribution function.

[IMSL_BINOMIALCDF](#)—Binomial distribution function.

[IMSL_BINOMIALPDF](#)—Binomial probability function.

[IMSL_HYPERGEOCDF](#)—Hypergeometric distribution function.

[IMSL_POISSONCDF](#)—Poisson distribution function.

IMSL_NORMALCDF

The IMSL_NORMALCDF function evaluates the standard normal (Gaussian) distribution function. Using a keyword, the inverse of the standard normal (Gaussian) distribution can be evaluated.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_NORMALCDF(*x* [, /DOUBLE] [, /INVERSE])

Return Value

The probability that a normal random variable takes a value less than or equal to *x*.

Arguments

x

Expression for which the normal distribution function is to be evaluated.

Keywords

DOUBLE

If present and nonzero, double precision is used.

INVERSE

If present and nonzero, evaluates the inverse of the standard normal (Gaussian) distribution function. If *Inverse* is specified, then argument *x* represents the probability for which the inverse of the normal distribution function is to be evaluated. In this case, *x* must be in the open interval (0.0, 1.0).

Discussion

The IMSL_NORMALCDF function evaluates the distribution function, Φ , of a standard normal (Gaussian) random variable; that is:

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-t^2/2} dt$$

The value of the distribution function at the point x is the probability that the random variable takes a value less than or equal to x .

The standard normal distribution (for which IMSL_NORMALCDF is the distribution function) has mean of zero and variance of 1. The probability that a normal random variable with mean μ and variance σ^2 is less than y is given by IMSL_NORMALCDF evaluated at $(y - \mu)/\sigma$.

The function $\Phi(x)$ is evaluated by use of the complementary error function, IMSL_ERFC. The relationship follows below:

$$\Phi(x) = \text{ERFC}((-x/\sqrt{2.0})/2)$$

If the keyword *Inverse* is specified, the IMSL_NORMALCDF function evaluates the inverse of the distribution function, Φ , of a standard normal (Gaussian) random variable; that is:

$$\text{IMSL\_NORMALCDF}(x, /Inverse) = \Phi^{-1}(x)$$

where:

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-t^2/2} dt$$

The value of the distribution function at the point x is the probability that the random variable takes a value less than or equal to x . The standard normal distribution has a mean of zero and a variance of 1.

The IMSL_NORMALCDF function is evaluated by use of minimax rational-function approximations for the inverse of the error function. General descriptions of these approximations are given in Hart et al. (1968) and Strecok (1968). The rational functions used in IMSL_NORMALCDF are described by Kinnucan and Kuki (1968).

Example

Suppose X is a normal random variable with mean 100 and variance 225. This example finds the probability that X is less than 90 and the probability that X is between 105 and 110.

```
x1 = (90-100)/15.  
p = IMSL_NORMALCDF(x1)  
PM, p, Title = 'The probability that X is less than 90 is:'  
The probability that X is less than 90 is: 0.252493  
x1 = (105 - 100)/15.  
x2 = (110 - 100)/15.  
p = IMSL_NORMALCDF(x2) - IMSL_NORMALCDF(x1)  
PM, p, Title = 'The probability that X is between 105 and ', $  
    '110 is:'
```

The probability that X is between 105 and 110 is: 0.116949

Version History

6.4	Introduced

IMSL_BINORMALCDF

The IMSL_BINORMALCDF function evaluates the bivariate normal distribution function.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_BINORMALCDF(*x*, *y*, *rho* [, /DOUBLE])

Return Value

The probability that a bivariate normal random variable with correlation *rho* takes a value less than or equal to *x* and less than or equal to *y*.

Arguments

rho

Correlation coefficient.

x

The x-coordinate of the point for which the bivariate normal distribution function is to be evaluated.

y

The y-coordinate of the point for which the bivariate normal distribution function is to be evaluated.

Keywords

DOUBLE

If present and nonzero, double precision is used.

Discussion

The IMSL_BINORMALCDF function evaluates the distribution function F of a bivariate normal distribution with means of zero, variances of 1, and correlation of ρ ; that is, $\rho = \text{rho}$ and $|\rho| < 1$.

$$F(x, y) = \frac{1}{2\pi\sqrt{1-\rho^2}} \int_{-\infty}^x \int_{-\infty}^y \exp\left(-\frac{u^2 - 2\rho uv + v^2}{2(1-\rho^2)}\right) dudv$$

To determine the probability that $U \leq u_0$ and $V \leq v_0$, where (U, V) is a bivariate normal random variable with mean $\mu = (\mu_U, \mu_V)$ and the following variance-covariance matrix:

$$\Sigma = \begin{bmatrix} \sigma_U^2 & \sigma_{UV} \\ \sigma_{UV} & \sigma_V^2 \end{bmatrix}$$

transform $(U, V)^T$ to a vector with zero means and unit variances. The input to IMSL_BINORMALCDF would be as follows: , ,

$$X = \frac{(u_0 - \mu_U)}{\sigma_U}$$

$$Y = \frac{(v_0 - \mu_V)}{\sigma_V}$$

and

$$\rho = \frac{\sigma_{UV}}{(\sigma_U \sigma_V)}$$

The IMSL_BINORMALCDF function uses the method of Owen (1962, 1965). For $|\rho| = 1$, the distribution function is computed based on the univariate statistic $Z = \min(x, y)$ and on the normal distribution IMSL_NORMALCDF.

Example

Suppose (x, y) is a bivariate normal random variable with mean $(0, 0)$ and the following variance-covariance matrix:

$$\begin{bmatrix} 1.0 & 0.9 \\ 0.9 & 1.0 \end{bmatrix}$$

This example finds the probability that x is less than -2.0 and y is less than 0.0 .

```
x = -2
y = 0
rho = .9
; Define x, y, and rho.
p = IMSL_BINORMALCDF(x, y, rho)
; Call IMSL_BINORMALCDF and output the results.
PM, 'P((x < -2.0) and (y < 0.0)) = ', p, FORMAT = '(a29, f8.4)'

P((x < -2.0) and (y < 0.0)) = 0.0228
```

Version History

6.4	Introduced

IMSL_CHISQCDF

The IMSL_CHISQCDF function evaluates the chi-squared distribution or non-central chi-squared distribution. Using a keyword the inverse of these distributions can be computed.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_CHISQCDF(*chisq*, *df* [, *delta*] [, /DOUBLE] [, /INVERSE])

Return Value

The probability that a chi-squared random variable takes a value less than or equal to *chisq*.

Arguments

chisq

Expression for which the chi-squared distribution function is to be evaluated. If the keyword INVERSE is specified, the probability for which the inverse of the non-central, chi-squared distribution function is to be evaluated, the parameter *chisq* must be in the open interval (0.0, 1.0).

delta

(Optional) The non-centrality parameter. *delta* must be nonnegative, and *delta* + *df* must be less than or equal to 200,000.

df

Number of degrees of freedom of the chi-squared distribution. Argument *df* must be greater than or equal to 0.5.

Keywords

DOUBLE

If present and nonzero, double precision is used.

INVERSE

If present and nonzero, evaluates the inverse of the chi-squared distribution function. If inverse is specified, then argument *chisq* represents the probability for which the inverse of the chi-squared distribution function is to be evaluated. Parameter *chisq* must be in the open interval (0.0, 1.0).

Discussion

If Two Input Arguments Are Used

The IMSL_CHISQCDF function evaluates the distribution function, F , of a chi-squared random variable with $v = df$. Then:

$$F(x) = \frac{1}{2^{v/2} \Gamma(v/2)} \int_0^x e^{-t/2} t^{v/2-1} dt$$

where $\Gamma(\cdot)$ is the gamma function. The value of the distribution function at the point x is the probability that the random variable takes a value less than or equal to x .

For $v > 65$, IMSL_CHISQCDF uses the Wilson-Hilferty approximation (Abramowitz and Stegun 1964, Equation 26.4.17) to the normal distribution, and IMSL_NORMALCDF function is used to evaluate the normal distribution function.

For $v \leq 65$, IMSL_CHISQCDF uses series expansions to evaluate the distribution function. If $x < \max(v/2, 26)$, IMSL_CHISQCDF uses the series 6.5.29 in Abramowitz and Stegun (1964); otherwise, it uses the asymptotic expansion 6.5.32 in Abramowitz and Stegun.

If *Inverse* is specified, the IMSL_CHISQCDF function evaluates the inverse distribution function of a chi-squared random variable with $v = df$ and with probability p . That is, it determines x , such that:

$$p = \frac{1}{2^{v/2} \Gamma(v/2)} \int_0^x e^{-t/2} t^{v/2-1} dt$$

where $\Gamma(\cdot)$ is the gamma function. The probability that the random variable takes a value less than or equal to x is p .

For $v < 40$, IMSL_CHISQCDF uses bisection (if $v \leq 2$ or $p > 0.98$) or regula falsi to find the expression for which the chi-squared distribution function is equal to p .

For $40 \leq v < 100$, a modified Wilson-Hilferty approximation (Abramowitz and Stegun 1964, Equation 26.4.18) to the normal distribution is used. The IMSL_NORMALCDF function is used to evaluate the inverse of the normal distribution function. For $v \geq 100$, the ordinary Wilson-Hilferty approximation (Abramowitz and Stegun 1964, Equation 26.4.17) is used.

If Three Input Arguments Are Used

The IMSL_CHISQCDF function evaluates the distribution function of a non-central chi-squared random variable with df degrees of freedom and non-centrality parameter $delta$, that is, with $v = df$, $\lambda = delta$, and $x = chisq$:

$$non\_central\_chi\_sq(x) = \sum_{i=0}^{\infty} \frac{e^{-\lambda/2} (\lambda/2)^i}{i!} \int_0^x \frac{t^{(v+2i)/2-1} e^{-t/2}}{2^{(v+2i)/2} \Gamma\left(\frac{v+2i}{2}\right)} dt$$

where $\Gamma(\cdot)$ is the gamma function. This is a series of central chi-squared distribution functions with Poisson weights. The value of the distribution function at the point x is the probability that the random variable takes a value less than or equal to x .

The non-central chi-squared random variable can be defined by the distribution function above, or alternatively and equivalently, as the sum of squares of independent normal random variables. If Y_i have independent normal distributions with means μ_i and variances equal to one and:

$$X = \sum_{i=1}^n Y_i^2$$

then X has a non-central chi-squared distribution with n degrees of freedom and non-centrality parameter equal to:

$$\sum_{i=1}^n \mu_i^2$$

With a non-centrality parameter of zero, the non-central chi-squared distribution is the same as the chi-squared distribution.

The `IMSL_CHISQCDF` function determines the point at which the Poisson weight is greatest, and then sums forward and backward from that point, terminating when the additional terms are sufficiently small or when a maximum of 1000 terms have been accumulated. The recurrence relation 26.4.8 of Abramowitz and Stegun (1964) is used to speed the evaluation of the central chi-squared distribution functions.

If *Inverse* is specified, `IMSL_CHISQCDF` evaluates the inverse distribution function of a non-central chi-squared random variable with *df* degrees of freedom and non-centrality parameter *delta*; that is, with $P = \text{chisq}$, $v = df$, and $\lambda = \text{delta}$, it determines $c_0 (= \text{IMSL\_CHISQCDF}(\text{chisq}, df, \text{delta}))$, such that:

$$P = \sum_{i=0}^{\infty} \frac{e^{-\lambda/2} (\lambda/2)^i}{i!} \int_0^{c_0} \frac{x^{(v+2i)/2-1} e^{-x/2}}{2^{(v+2i)/2} \Gamma\left(\frac{v+2i}{2}\right)} dx$$

where $\Gamma(\cdot)$ is the gamma function. The probability that the random variable takes a value less than or equal to c_0 is P .

Example

Suppose X is a chi-squared random variable with two degrees of freedom. This example finds the probability that X is less than 0.15 and the probability that X is greater than 3.0.

```
df = 2
chisq = .15
p = IMSL_CHISQCDF(chisq, df)
PM, p, Title = 'The probability that chi-squared with 2 df ' + $
               'is less than .15 is:'
```

```
The probability that chi-squared with 2 df is less than .15 is:
0.0722565
```

```
df = 2
chisq = 3
p = 1 - IMSL_CHISQCDF(chisq, df)
PM, p, Title = 'The probability that chi-squared ' + $
               'with 2 df is greater than 3 is:'
```

```
The probability that chi-squared with 2 df is greater than 3 is:
0.223130
```

Errors

Informational Errors

- STAT_ARG_LESS_THAN_ZERO—Input parameter, *chisq*, is less than zero.
- STAT_UNABLE_TO_BRACKET_VALUE—Bounds that enclose *p* could not be found. An approximation for IMSL_CHISQCDF is returned.
- STAT_CHI_2_INV_CDF_CONVERGENCE—Value of the inverse chi-squared could not be found within a specified number of iterations. An approximation for IMSL_CHISQCDF is returned.

Alert Errors

- STAT_NORMAL_UNDERFLOW—Using the normal distribution for large degrees of freedom, underflow would have occurred.

Version History

6.4	Introduced

IMSL_FCDF

The IMSL_FCDF function evaluates the F distribution function. Using a keyword, the inverse of the F distribution function can be evaluated.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_FCDF(*f*, *dfnum*, *dfden* [, /DOUBLE] [, /INVERSE])

Return Value

The probability that an F random variable takes a value less than or equal to the input point f .

Arguments

dfden

Denominator degrees of freedom. Parameter *dfden* must be positive.

dfnum

Numerator degrees of freedom. Parameter *dfnum* must be positive.

f

Expression for which the F distribution function is to be evaluated.

Keywords

DOUBLE

If present and nonzero, double precision is used.

INVERSE

If present and nonzero, evaluates the inverse of the F distribution function. If inverse is specified, argument f represents the probability for which the inverse of the F distribution function is to be evaluated. In this case, f must be in the open interval $(0.0, 1.0)$.

Discussion

The `IMSL_FCDF` function evaluates the distribution function of a Snedecor's F random variable with $dfnum$ and $dfden$. The function is evaluated by making a transformation to a beta random variable and then evaluating the incomplete beta function. If X is an F variate with v_1 and v_2 degrees of freedom and $Y = (v_1 X) / (v_2 + v_1 X)$, then Y is a beta variate with parameters $p = v_1/2$ and $q = v_2/2$. The `IMSL_FCDF` function also uses a relationship between F random variables that is expressed as follows: $F_F(f, v_1, v_2) = 1 - F_F(1/f, v_2, v_1)$, where F_F is the distribution function for an F random variable.

If *Inverse* is specified, the `IMSL_FCDF` function evaluates the inverse distribution function of a Snedecor's F random variable with $v_1 = dfnum$ numerator degrees of freedom and $v_2 = dfden$ denominator degrees of freedom. The function is evaluated by making a transformation to a beta random variable and then evaluating the inverse of an incomplete beta function.

Example

This example finds the probability that an F random variable with one numerator and one denominator degree of freedom is greater than 648.

```
f = 648
p = 1 - IMSL_FCDF(f, 1, 1)
PM, p, Title = 'The probability that an F(1,1) ' + $
               'variate is greater than 648 is:'
```

```
The probability that an F(1,1) variate is greater than 648 is:
0.0249959
```

Errors

Fatal Errors

`STAT_F_INVERSE_OVERFLOW`— `IMSL_FCDF` is set to machine infinity since overflow would occur upon modifying the inverse value for the F distribution with the result obtained from the inverse beta distribution.

Version History

6.4	Introduced

IMSL_TCDF

The IMSL_TCDF function evaluates the Student's t distribution or non-central Student's t distribution. Using a keyword the inverse of these distributions can be computed.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

$Result = \text{IMSL\_TCDF}(chisq, df[, delta] [, /DOUBLE] [, /INVERSE])$

Return Value

The probability that a Student's t random variable takes a value less than or equal to the input t .

Arguments

delta

(Optional) The non-centrality parameter.

df

Degrees of freedom. Argument df must be greater than or equal to 1.0.

t

Argument for which the Student's t distribution function is to be evaluated. If *Inverse* is specified, argument t represents the probability for which the inverse of the Student's t distribution function is to be evaluated. In this case, t must be in the open interval (0.0, 1.0).

Keywords

DOUBLE

If present and nonzero, double precision is used.

INVERSE

If present and nonzero, evaluates the inverse of the Student's t distribution function. If *Inverse* is specified, argument t represents the probability for which the inverse of the Student's t distribution function is to be evaluated. In this case, t must be in the open interval (0.0, 1.0).

Discussion

If Two Input Arguments Are Used

The IMSL_TCDF function evaluates the distribution function of a Student's t random variable with $v = df$ degrees of freedom. If $t^2 \geq v$, the relationship of a t to an F random variable (and subsequently, to a beta random variable) is exploited, and percentage points from a beta distribution are used. Otherwise, the method described by Hill (1970) is used. If v is not an integer or if v is greater than 19, a Cornish-Fisher expansion is used to evaluate the distribution function. If v is less than 20 and $|t|$ is less than 2.0, a trigonometric series (see Abramowitz and Stegun 1964, Equations 26.7.3 and 26.7.4, with some rearrangement) is used. For the remaining cases, a series given by Hill (1970) that converges well for large values of t is used.

If keyword *Inverse* is specified, the IMSL_TCDF function evaluates the inverse distribution function of a Student's t random variable with $v = df$ degrees of freedom. If v equals 1 or 2, the inverse can be obtained in closed form. If v is between 1 and 2, the relationship of a t to a beta random variable is exploited, and the inverse of the beta distribution is used to evaluate the inverse. Otherwise, the algorithm of Hill (1970) is used. For small values of v greater than 2, Hill's algorithm inverts an integrated expansion in $1 / (1 + t^2 / v)$ of the t density. For larger values, an asymptotic inverse Cornish-Fisher type expansion about normal deviates is used.

If Three Input Arguments Are Used

The IMSL_TCDF function evaluates the distribution function F of a non-central t random variable with df degrees of freedom and non-centrality parameter δ ; that is, with $v = df$, $\delta = \delta$, and $t_0 = t$:

$$F(t_0) = \int_{-\infty}^{t_0} \frac{(v^{v/2} e^{-\delta^2/2})}{\sqrt{\pi} \Gamma(v/2) (v + x^2)^{(v+1)/2}} \sum_{i=0}^{\infty} \Gamma((v+i+1)/2) \left(\frac{\delta}{i!}\right) \left(\frac{2x^2}{v+x^2}\right)^{i/2} dx$$

where $\Gamma(\cdot)$ is the gamma function. The value of the distribution function at the point t_0 is the probability that the random variable takes a value less than or equal to t_0 .

The non-central t random variable can be defined by the distribution function above, or alternatively and equivalently, as the ratio of a normal random variable and an independent chi-squared random variable. If w has a normal distribution with mean δ and variance equal to one, u has an independent chi-squared distribution with v degrees of freedom, and:

$$x = w / (\sqrt{u/v})$$

then x has a non-central t distribution with degrees of freedom and non-centrality parameter δ .

The distribution function of the non-central t can also be expressed as a double integral involving a normal density function (see, for example, Owen 1962, page 108). The function `TNDF` uses the method of Owen (1962, 1965), which uses repeated integration by parts on that alternate expression for the distribution function.

If *Inverse* is specified `IMSL_TCDF` evaluates the inverse distribution function of a non-central t random variable with df degrees of freedom and non-centrality parameter *delta*; that is, with $P = t$, $v = df$, and $\delta = \text{delta}$, it determines t_0 (`= IMSL_TCDF (t, df, delta)`), such that:

$$P = \int_{-\infty}^{t_0} \frac{v^{v/2} e^{-\delta^2/2}}{\sqrt{\pi} \Gamma(v/2) (v + x^2)^{(v+1)/2}} \sum_{i=0}^{\infty} \Gamma((v+i+1)/2) \left(\frac{\delta}{i!}\right) \left(\frac{2x^2}{v+x^2}\right)^{i/2} dx$$

where $\Gamma(\cdot)$ is the gamma function. The probability that the random variable takes a value less than or equal to t_0 is P .

Example

This example finds the probability that a t random variable with six degrees of freedom is greater in absolute value than 2.447. Argument t is symmetric about zero.

```
p = 2 * IMSL_TCDF(-2.447, 6)
PM, 'Pr(|t(6)| > 2.447) = ', p, FORMAT = '(a21, f7.4)'

Pr(|t(6)| > 2.447) = 0.0500
```

Errors

Informational Errors

STAT_OVERFLOW—IMSL_TCDF is set to machine infinity since overflow would occur upon modifying the inverse value for the F distribution with the result obtained from the inverse beta distribution.

Version History

6.4	Introduced

IMSL_GAMMACDF

The IMSL_GAMMACDF function evaluates the gamma distribution function.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_GAMMACDF(*x*, *a* [, /DOUBLE])

Return Value

The probability that a gamma random variable takes a value less than or equal to *x*.

Arguments

a

Shape parameter of the gamma distribution. This parameter must be positive.

x

Argument for which the gamma distribution function is to be evaluated.

Keywords

DOUBLE

If present and nonzero, double precision is used.

Discussion

The IMSL_GAMMACDF function evaluates the distribution function, F , of a gamma random variable with shape parameter a ; that is:

$$F(x) = \frac{1}{\Gamma(a)} \int_0^x e^{-t} t^{a-1} dt$$

where $\Gamma(\cdot)$ is the gamma function. (The gamma function is the integral from 0 to *infinity* of the same integrand as above.) The value of the distribution function at the point x is the probability that the random variable takes a value less than or equal to x .

The gamma distribution is often defined as a two-parameter distribution with a scale parameter b (which must be positive) or even as a three-parameter distribution in which the third parameter c is a location parameter. In the most general case, the probability density function over $(c, \text{infinity})$ is as follows:

$$f(t) = \frac{1}{b^a \Gamma(a)} e^{-(t-c)/b} (x-c)^{a-1}$$

If T is such a random variable with parameters a , b , and c , the probability that $T \leq t_0$ can be obtained from IMSL_GAMMACDF by setting $x = (t_0 - c) / b$.

If x is less than a or if x is less than or equal to 1.0, IMSL_GAMMACDF uses a series expansion; otherwise, a continued fraction expansion is used. (See Abramowitz and Stegun, 1964.)

Example

Let X be a gamma random variable with a shape parameter of 4. (In this case, it has an Erlang distribution, since the shape parameter is an integer.) This example finds the probability that X is less than 0.5 and the probability that X is between 0.5 and 1.0.

```
a = 4
x = .5
p = IMSL_GAMMACDF(x, a)
PM, p, Title = 'The probability that X is less ' + $
               'than .5 is:'

The probability that X is less than .5 is: 0.00175162

x = 1
p = IMSL_GAMMACDF(x, a) - p
PM, p, Title = 'The probability that X is between .5 and 1 is:'

The probability that X is between .5 and 1 is: 0.0172365
```

Errors

Informational Errors

STAT_LESS_THAN_ZERO—Input argument, x , is less than zero.

Fatal Errors

STAT_X_AND_A_TOO_LARGE—Function overflows because x and a are too large.

Version History

6.4	Introduced

IMSL_BETACDF

The IMSL_BETACDF function evaluates the beta probability distribution function.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_BETACDF(*x*, *pin*, *qin* [, /DOUBLE] [, /INVERSE])

Return Value

The probability that a beta random variable takes on a value less than or equal to *x*.

Arguments

pin

First beta distribution parameter. Parameter *pin* must be positive.

qin

Second beta distribution parameter. Parameter *qin* must be positive.

x

Argument for which the beta probability distribution function is to be evaluated. If *Inverse* is specified, argument *x* represents the probability for which the inverse of the Beta distribution function is to be evaluated. In this case, *x* must be in the open interval (0.0, 1.0).

Keywords

DOUBLE

If present and nonzero, double precision is used.

INVERSE

If `present` and `nonzero`, evaluates the inverse of the Beta distribution function. If *Inverse* is specified, argument *x* represents the probability for which the inverse of the Beta distribution function is to be evaluated. In this case, *x* must be in the open interval (0.0, 1.0).

Discussion

The `IMSL_BETACDF` function evaluates the distribution function of a beta random variable with parameters *pin* and *qin*. This function is sometimes called the incomplete beta ratio and is denoted by $I_x(p, q)$, where $p = pin$ and $q = qin$. It is given by:

$$I_x(p, q) = \frac{\Gamma(p)\Gamma(q)}{\Gamma(p+q)} \int_0^x t^{p-1} (1-t)^{q-1} dt$$

where $\Gamma(\cdot)$ is the gamma function. The value of the distribution function by $I_x(p, q)$ is the probability that the random variable takes a value less than or equal to *x*.

The integral in the expression above is called the incomplete beta function and is denoted by $\beta_x(p, q)$. The constant in the expression is the reciprocal of the beta function (the incomplete function evaluated at 1) and is denoted by $\beta_x(p, q)$.

If the keyword *Inverse* is specified, the `IMSL_BETACDF` function evaluates the inverse distribution function of a beta random variable with parameters *pin* and *qin*. With $P = x$, $p = pin$ and $q = qin$, it returns *x* such that:

$$P = \frac{\Gamma(p)\Gamma(q)}{\Gamma(p+q)} \int_0^x t^{p-1} (1-t)^{q-1} dt$$

where $\Gamma(\cdot)$ is the gamma function. The probability that the random variable takes a value less than or equal to *x* is *P*.

The `BETCDF` function uses the method of Bosten and Battiste (1974).

Example

Suppose *X* is a beta random variable with parameters 12 and 12 (*X* has a symmetric distribution). This example finds the probability that *X* is less than 0.6 and the probability that *X* is between 0.5 and 0.6. (Since *X* is a symmetric beta random variable, the probability that it is less than 0.5 is 0.5.)

```
p = IMSL_BETACDF(.6, 12, 12)
; Call IMSL_BETACDF to compute first probability and output
results.
```

```
PM, p, Title = 'The probability that X is less than ' + $  
              '0.6 is:', FORMAT= '(f8.4)'
```

The probability that X is less than 0.6 is: 0.8364

```
p = p - IMSL_BETACDF(.5, 12, 12)  
; Call IMSL_BETACDF and use the previously computed  
; probability to determine the next probability.  
PM, p, FORMAT = '(f8.4)', title = 'The probability that X ' + $  
              'is between 0.5 and 0.6 is:'
```

The probability that X is between 0.5 and 0.6 is: 0.3364

Version History

6.4	Introduced

IMSL_BINOMIALCDF

The IMSL_BINOMIALCDF function evaluates the binomial distribution function.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

$Result = \text{IMSL\_BINOMIALCDF}(k, n, p [, /DOUBLE])$

Return Value

The probability that k or fewer successes occur in n independent Bernoulli trials, each of which has a probability p of success.

Arguments

k

Argument for which the binomial distribution function is to be evaluated.

n

Number of Bernoulli trials.

p

Probability of success on each trial.

Keywords

DOUBLE

If present and nonzero, double precision is used.

Discussion

The `IMSL_BINOMIALCDF` function evaluates the distribution function of a binomial random variable with parameters n and p by summing probabilities of the random variable taking on the specific values in its range. These probabilities are computed by the following recursive relationship:

$$\Pr(X = j) = \frac{(n+1-j)p}{j(1-p)} \Pr(X = j-1)$$

To avoid the possibility of underflow, the probabilities are computed forward from 0 if k is not greater than n times p ; otherwise, they are computed backward from n . The smallest positive machine number, ϵ , is used as the starting value for summing the probabilities, which are rescaled by $(1-p)^n \epsilon$ if forward computation is performed and by $p^n \epsilon$ if backward computation is done.

For the special case of $p = 0$, `IMSL_BINOMIALCDF` is set to 1; for the case $p = 1$, `IMSL_BINOMIALCDF` is set to 1 if $k = n$ and is set to zero otherwise.

Example

Suppose X is a binomial random variable with $n = 5$ and $p = 0.95$. This example finds the probability that X is less than or equal to 3.

```
p = IMSL_BINOMIALCDF(3, 5, .95)
PM, 'Pr(x < 3) = ', p, FORMAT = '(a12, f7.4) '

Pr(x < 3) = 0.0226
```

Errors

Informational Errors

`STAT_LESS_THAN_ZERO`—Input parameter, k , is less than zero.

`STAT_GREATER_THAN_N`—Input parameter, k , is greater than the number of Bernoulli trials, n .

Version History

6.4	Introduced

IMSL_BINOMIALPDF

The IMSL_BINOMIALPDF function evaluates the binomial probability function.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_BINOMIALPDF (*k*, *n*, *p*)

Return Value

The probability that a binomial random variable takes a value equal to *k*.

Arguments

k

Argument for which the binomial probability function is to be evaluated.

n

Number of Bernoulli trials.

p

Probability of success on each trial.

Discussion

The IMSL_BINOMIALPDF function evaluates the probability that a binomial random variable with parameters *n* and *p* takes on the value *k*. It does this by computing probabilities of the random variable taking on the values in its range less than (or the values greater than) *k*. These probabilities are computed by the recursive relationship:

$$\Pr(X = j) = \frac{(n + 1 - j)}{j(1 - p)} \Pr(X = j - 1)$$

To avoid the possibility of underflow, the probabilities are computed forward from 0, if k is not greater than n times p , and are computed backward from n , otherwise. The smallest positive machine number, ϵ , is used as the starting value for computing the probabilities, which are rescaled by $(1 - p)^n \epsilon$ if forward computation is performed and by $p^n \epsilon$ if backward computation is done.

For the special case of $p = 0$, `IMSL_BINOMIALPDF` returns 0 if k is greater than 0 and to 1 otherwise; and for the case $p = 1$, `IMSL_BINOMIALPDF` returns 0 if k is less than n and to 1 otherwise.

Example

Suppose X is a binomial random variable with $n = 5$ and $p = 0.95$. In this example, we find the probability that X is equal to 3.

```
PRINT, IMSL_BINOMIALPDF(3, 5, .95)
0.0214344
```

Version History

6.4	Introduced

IMSL_HYPERGEOCDF

The IMSL_HYPERGEOCDF function evaluates the hypergeometric distribution function.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

$Result = \text{IMSL\_HYPERGEOCDF}(k, n, m, l [, \text{/DOUBLE}])$

Return Value

The probability that k or fewer defectives occur in a sample of size n drawn from a lot of size l that contains m defectives.

Arguments

k

Parameter for which the hypergeometric distribution function is to be evaluated.

l

Lot size. Parameter l must be greater than or equal to n and m .

m

Number of defectives in the lot.

n

Sample size. Argument n must be greater than or equal to k .

Keywords

DOUBLE

If present and nonzero, double precision is used.

Discussion

The IMSL_HYPERGEOCDF function evaluates the distribution function of a hypergeometric random variable with parameters n , l , and m . The hypergeometric random variable X can be thought of as the number of items of a given type in a random sample of size n that is drawn without replacement from a population of size l containing m items of this type.

The probability function is:

$$\Pr(x = j) = \frac{\binom{m}{j} \binom{l-m}{n-j}}{\binom{l}{n}} \quad \text{for } j = i, i+1, \dots, \min(n, m)$$

where $i = \max(0, n - l + m)$.

If k is greater than or equal to i and less than or equal to $\min(n, m)$, IMSL_BINOMIALCDF sums the terms in this expression for j going from i up to k ; otherwise, 0 or 1 is returned, as appropriate. To avoid rounding in the accumulation, IMSL_BINOMIALCDF performs the summation differently, depending on whether or not k is greater than the mode of the distribution, which is the greatest integer in $(m + 1)(n + 1)/(l + 2)$.

Example

Suppose X is a hypergeometric random variable with $n = 100$, $l = 1000$, and $m = 70$. In this example, the distribution function is evaluated at 7.

```
p = IMSL_HYPERGEOCDF(7, 100, 70, 1000)
PM, 'Pr(x <= 7) = ', p, FORMAT = '(a13,f7.4) '

Pr(x <= 7) = 0.5995
```

Errors

Informational Errors

STAT_LESS_THAN_ZERO—Input parameter, k , is less than zero.

STAT_K_GREATER_THAN_N—Input parameter, k , is greater than the sample size.

Fatal Errors

STAT_LOT_SIZE_TOO_SMALL—Lot size must be greater than or equal to n and m .

Version History

6.4	Introduced

IMSL_POISSONCDF

The IMSL_POISSONCDF function evaluates the Poisson distribution function.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_POISSONCDF(*k*, *theta* [, /DOUBLE])

Return Value

The probability that a Poisson random variable takes a value less than or equal to *k*.

Arguments

k

Parameter for which the Poisson distribution function is to be evaluated.

theta

Mean of the Poisson distribution. Parameter *theta* must be positive.

Keywords

DOUBLE

If present and nonzero, double precision is used.

Discussion

The IMSL_POISSONCDF function evaluates the distribution function of a Poisson random variable with parameter *theta*. The mean of the Poisson random variable, *theta*, must be positive.

The probability function (with $\theta = \textit{theta}$) is as follows:

$$f(x) = (e^{-\theta} \theta^x) / x! \quad \text{for } x = 0, 1, 2, \dots$$

The individual terms are calculated from the tails of the distribution to the mode of the distribution and summed. The IMSL_POISSONCDF function uses the recursive relationship:

$$f(x + 1) = f(x)(\theta / (x + 1)) \text{ , } \text{ for } x = 0, 1, 2, \dots, k - 1$$

with .

$$f(0) = e^{-\theta}$$

Example

Suppose X is a Poisson random variable with $\theta = 10$. This example evaluates the probability that $X \leq 7$.

```
p = IMSL_POISSONCDF(7, 10)
PM, 'Pr(x <= 7) = ', p, FORMAT = '(a13,f7.4) '

Pr(x <= 7) = 0.2202
```

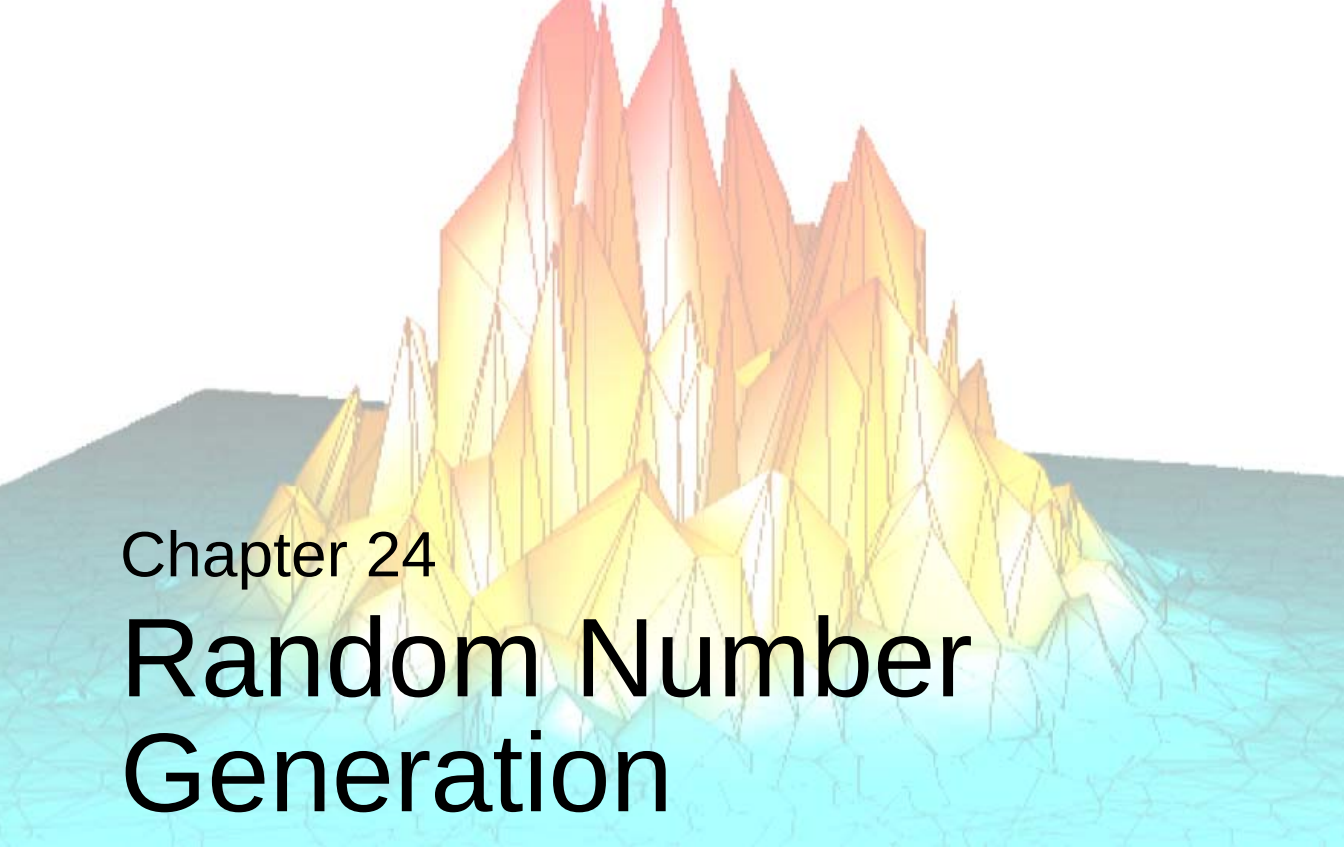
Errors

Informational Errors

STAT_LESS_THAN_ZERO— Input parameter, k , is less than zero.

Version History

6.4	Introduced



Chapter 24

Random Number Generation

This section contains the following topics:

[Overview: Random Number Generation](#) . 1066 [Random Number Generation Routines](#) . 1069

Overview: Random Number Generation

This chapter describes random number generation functions used for applications in Monte Carlo or simulation studies. Before using random number generators, the generator must be initialized by selecting a seed or starting value. You can do this by using `IMSL_RANDOMOPT`. If you do not select a seed, one is generated using the system clock. A seed needs to be selected only once in a program, unless two or more separate streams of random numbers are maintained. Utility functions in this chapter can be used to select the form of the basic generator to restart simulations and to maintain separate simulation streams.

In the following sections, the terms *random numbers*, *random deviates*, *deviates*, and *variates* are used interchangeably. The phrase *pseudorandom* is sometimes used to emphasize that the numbers generated are really not random since they result from a deterministic process. The usefulness of pseudorandom numbers is derived from the similarity, in a statistical sense, of samples of the pseudorandom numbers to samples of observations from the specified distributions. In short, while the pseudorandom numbers are deterministic and repeatable, they simulate the realizations of independent and identically distributed random variables.

Basic Uniform Generator

The default action of the `IMSL_RANDOM` function is the generation of uniform (0,1) numbers. This function is portable in that, given the same seed, it produces the same sequence in all computer/compiler environments.

The random number generators in this chapter use either a multiplicative congruential method or a generalized feedback shift register (GFSR) method. The selection of the type of generator is made by calling the “[IMSL_RANDOMOPT](#)” on page 1071. If no selection is made explicitly, a multiplicative generator (with multiplier 16807) is used. Whatever distribution is being simulated, uniform (0, 1) numbers are first generated and then transformed if necessary. These routines are portable in the sense that, given the same seed and for a given type of generator, they produce the same sequence in all computer/compiler environments. There are many other issues that must be considered in developing programs for the methods described below (see Gentle 1981 and 1990).

Multiplicative Congruential Generators

The form of the multiplicative congruential generators is:

$$x_i \equiv cx_{i-1} \bmod (2^{31} - 1)$$

Each x_i is then scaled into the unit interval (0,1). If the multiplier, c , is a primitive root modulo $2^{31} - 1$ (which is a prime), then the generator will have a maximal period of $2^{31} - 2$. There are several other considerations, however. See Knuth (1981) for a good general discussion. The possible values for c in the generators are 16807, 397204094, and 950706376. The selection is made by `IMSL_RANDOMOPT`. The choice of 16807 will result in the fastest execution time, but other evidence suggests that the performance of 950706376 is best among these three choices (Fishman and Moore 1982). If no selection is made explicitly, the functions use the multiplier 16807, which has been in use for some time (Lewis et al. 1969).

Shuffled Generators

You also can select a shuffled version of these generators using `IMSL_RANDOMOPT`. The shuffled generators use a scheme due to Learmonth and Lewis (1973). In this scheme, a table is filled with the first 128 uniform (0,1) numbers resulting from the simple multiplicative congruential generator. Then, for each x_i from the simple generator, the low-order bits of x_i are used to select a random integer, j , from 1 to 128. The j -th entry in the table is then delivered as the random number; and x_i , after being scaled into the unit interval, is inserted into the j -th position in the table. This scheme is similar to that of Bays and Durham (1976), and their analysis is applicable to this scheme as well.

Generalized Feedback Shift Register Generator

The GFSR generator uses the recursion $X_t = X_{t-1563} \oplus X_{t-96}$. This generator, which is different from earlier GFSR generators, was proposed by Fushimi (1990), who discusses the theory behind the generator and reports on several empirical tests of it. Background discussions on this type of generator can be found in Kennedy and Gentle (1980), pages 150-162.

Setting Seed

The seed of the generator can be set and retrieved using `IMSL_RANDOMOPT`. Prior to invoking any generator in this section, you can call `IMSL_RANDOMOPT` to initialize the seed, which is an integer variable with a value between 1 and 2147483647. If it is not initialized by `IMSL_RANDOMOPT`, a random seed is obtained from the system clock. Once it is initialized, the seed need not be set again.

If you want to restart a simulation, `IMSL_RANDOMOPT` can be used to obtain the final seed value of one run to be used as the starting value in a subsequent run. Also, if two simultaneous random number streams are desired in one run,

IMSL_RANDOMOPT can be used before and after the invocations of the generators in each stream.

If a shuffled generator or the GFSR generator is used, in addition to resetting the seed, you must also reset some values in a table. For the shuffled generators, this is done using the routine IMSL_RANDOM_TABLE. The tables for the shuffled generators are separate for single and double precision; so, if precisions are mixed in a program, it is necessary to manage each precision separately for the shuffled generators.

Distributions Other than Uniform

The nonuniform generators use a variety of transformation procedures. All of the transformations used are exact (mathematically). The most straightforward transformation is the inverse CDF technique, but it is often less efficient than others involving acceptance/rejection and mixtures. See Kennedy and Gentle (1980) for discussion of these and other techniques.

Many of the nonuniform generators in this chapter use different algorithms depending on the values of the parameters of the distributions. This is particularly true of the generators for discrete distributions. Schmeiser (1983) gives an overview of techniques for generating deviates from discrete distributions.

Although, as noted above, the uniform generators yield the same sequences on different computers, because of rounding, the nonuniform generators that use acceptance/rejection may occasionally produce different sequences on different computer/compiler environments.

Although the generators for nonuniform distributions use fast algorithms, if a very large number of deviates from a fixed distribution are to be generated, it might be worthwhile to consider a table sampling method, as implemented in the routines IMSL_RAND_GEN_CONT and IMSL_RAND_GEN_DISCR.

Additional Notes on Syntax

The generators for continuous distributions are available in both single and double precision versions. This is merely for your convenience; the double precision versions should not be considered more “accurate,” except possibly for the multivariate distributions.

Random Number Generation Routines

Random Numbers

IMSL_RANDOMOPT—Retrieves uniform (0, 1) multiplicative, congruential pseudorandom-number generator.

IMSL_RANDOM_TABLE—Sets or retrieves the current table used in either the shuffled or GFSR random number generator.

IMSL_RANDOM—Generates pseudorandom numbers.

IMSL_RANDOM_NPP—Generates pseudorandom numbers from a nonhomogeneous Poisson process.

IMSL_RANDOM_ORDER—Generates pseudorandom order statistics from a uniform (0, 1) distribution, or optionally from a standard normal distribution.

IMSL_RAND_TABLE_2WAY—Generates a pseudorandom two-way table.

IMSL_RAND_ORTH_MAT—Generates a pseudorandom orthogonal matrix or a correlation matrix.

IMSL_RANDOM_SAMPLE—Generates a simple pseudorandom sample from a finite population.

IMSL_RAND_FROM_DATA—Generates pseudorandom numbers from a multivariate distribution determined from a given sample.

IMSL_CONT_TABLE—Sets up table to generate pseudorandom numbers from a general continuous distribution.

IMSL_RAND_GEN_CONT—Generates pseudorandom numbers from a general continuous distribution.

IMSL_DISCR_TABLE—Sets up table to generate pseudorandom numbers from a general discrete distribution.

IMSL_RAND_GEN_DISCR—Generates pseudorandom numbers from a general discrete distribution using an alias method or optionally a table lookup method.

Stochastic Processes

IMSL_RANDOM_ARMA—Generate pseudorandom IMSL_ARMA process numbers.

Low-discrepancy Sequences

`IMSL_FAURE_INIT`—Initializes the structure used for computing a shuffled Faure sequence.

`IMSL_FAURE_NEXT_PT`—Generates a shuffled Faure sequence.

IMSL_RANDOMOPT

The IMSL_RANDOMOPT procedure uses keywords to set or retrieve the random number seed or to select the form of the IMSL random number generator.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
IMSL_RANDOMOPT ([, CURRENT_OPTION=variable] [, GEN_OPTION=value]  
                [, GET_SEED=variable] [, SET_SEED=value] [, SUBSTREAM_SEED=value])
```

Arguments

The IMSL_RANDOMOPT procedure does not have any positional Input Parameters. Keywords are required for specific actions to be taken.

Keywords

CURRENT_OPTION

Named variable into which the value of the current random-number generator option is stored.

GEN_OPTION

Indicator of the generator. The random-number generator is a multiplicative, congruential generator with modulus $2^{31} - 1$. Keyword *Gen_Option* is used to choose the multiplier and to determine whether or not shuffling is done.

- 1—multiplier 16807 used (default)
- 2—multiplier 16807 used with shuffling
- 3—multiplier 397204094 used
- 4—multiplier 397204094 used with shuffling
- 5—multiplier 950706376 used
- 6—multiplier 950706376 used with shuffling

- 7—GFSR, with the recursion $X_t = X_{t-1563} \oplus X_{t-96}$ is used

GET_SEED

Named variable into which the value of the current random-number seed is stored.

SET_SEED

Seed of the random-number generator. The seed must be in the range (0, 2147483646). If the seed is zero, a value is computed using the system clock; hence, the results of programs using the IDL Advanced Math and Stats random-number generators are different at various times.

SUBSTREAM_SEED

If present and nonzero, then a seed for the congruential generators that do not do shuffling that will generate random numbers beginning 100,000 numbers farther along will be returned in keyword *Get*. If keyword *Substream_seed* is set, then keyword *Get* is required.

Discussion

The IMSL_RANDOMOPT procedure is designed to allow a user to set certain key elements of the random-number generator functions.

The uniform pseudorandom-number generators use a multiplicative congruential method, or a generalized feedback shift register. The choice of generator is determined by keyword *Gen_Option*. The chapter introduction and the description of IMSL_RANDOM may provide some guidance in the choice of the form of the generator. If no selection is made explicitly, the generators use the multiplier 16807 without shuffling. This form of the generator has been in use for some time (Lewis et al. 1969).

Keyword *Set* is used to initialize the seed used in the IDL Advanced Math and Stats random-number generators. See the chapter introduction for details of the various generator options. The seed can be reinitialized to a clock-dependent value by calling IMSL_RANDOMOPT with *Set* set to zero.

A common use of keyword *Set* is in conjunction with the keyword *Get* to restart a simulation. Keyword *Get* retrieves the current value of the “seed” used in the random-number generators.

If keyword *Substream_seed* is set, IMSL_RANDOMOP determines another seed, such that if one of the IMSL multiplicative congruential generators, using no shuffling, went through 100,000 generations starting with *Substream_seed*, the next

number in that sequence would be the first number in the sequence that begins with the returned seed.

Note that *Substream_seed* works only when a multiplicative congruential generator without shuffling is used. This means that either the routine `IMSL_RANDOMOPT` has not been called at all or that it has been last called with *Gen_Option* having a value of 1, 3, or 5.

For many IMSL generators for nonuniform distributions that do not use the inverse CDF method, the distance between sequences generated starting with *Substream_seed* and starting with returned seed may be less than 100,000. This is because nonuniform generators that use other techniques may require more than one uniform deviate for each output deviate.

The reason that one may want two seeds that generate sequences a known distance apart is for blocking Monte Carlo experiments or for running parallel streams.

Examples

Example 1

This example illustrates the statements required to restart a simulation using the keywords *Get* and *Set*. The example shows that restarting the sequence of random numbers at the value of the last seed generated is the same as generating the random numbers all at once.

```
seed = 123457
nrandom = 5
IMSL_RANDOMOPT, Set = seed
; Set the seed using the keyword Set.
r1 = IMSL_RANDOM(nrandom)
PM, r1, Title = 'First Group of Random Numbers'
```

```
First Group of Random Numbers
0.966220
0.260711
0.766262
0.569337
0.844829
```

```
IMSL_RANDOMOPT, Get = seed
; Get the current value of the seed using the keyword Get.
IMSL_RANDOMOPT, Set = seed
; Set the seed.
r2 = IMSL_RANDOM(nrandom)
PM, r2, Title = 'Second Group of Random Numbers'
```

Second Group of Random Numbers

```
0.0442665
0.987184
0.601350
0.896375
0.380854
```

```
IMSL_RANDOMOPT, Set = 123457
```

```
; Reset the seed to the original seed.
```

```
r3 = IMSL_RANDOM(2 * nrandom)
```

```
PM, r3, Title = 'Both Groups of Random Numbers'
```

Both Groups of Random Numbers

```
0.966220
0.260711
0.766262
0.569337
0.844829
0.0442665
0.987184
0.601350
0.896375
0.380854
```

Example 2

In this example, `IMSL_RANDOMOPT` is used to determine seeds for 4 separate streams, each 200,000 numbers apart, for a multiplicative congruential generator without shuffling. (Since `IMSL_RANDOMOPT` is not invoked to select a generator, the multiplier is 16807.) Since the streams are 200,000 numbers apart, each seed requires two invocations of `IMSL_RANDOMOPT` with keyword *Substream_seed*. All of the streams are non-overlapping, since the period of the underlying generator is 2,147,483,646.

```
IMSL_RANDOMOPT, GEN_OPTION = 1
```

```
is1 = 123457;
```

```
IMSL_RANDOMOPT, GET = itmp, SUBSTREAM_SEED = is1
```

```
IMSL_RANDOMOPT, GET = is2, SUBSTREAM_SEED = itmp
```

```
IMSL_RANDOMOPT, GET = itmp, SUBSTREAM_SEED = is2
```

```
IMSL_RANDOMOPT, GET = is3, SUBSTREAM_SEED = itmp
```

```
IMSL_RANDOMOPT, GET = itmp, SUBSTREAM_SEED = is3
```

```
IMSL_RANDOMOPT, GET = is4, SUBSTREAM_SEED = itmp
```

```
PRINT, is1, is2, is3, is4
```

```
123457 2016130173 85016329 979156171
```

Version History

6.4	Introduced

IMSL_RANDOM_TABLE

The IMSL_RANDOM_TABLE procedure sets or retrieves the current table used in either the shuffled or GFSR random number generator.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
IMSL_RANDOM_TABLE, table [, /DOUBLE] [, /GET | /SET] [, /GFSR]
```

Arguments

table

One dimensional array used in the generators. For the shuffled generators table is length 128. For the GFSR generator table is length 1565. The argument table is input if the keyword *Set* is used, and output if the keyword *Get* is used.

Keywords

DOUBLE

If present and nonzero, double precision is used. This keyword is active only when the shuffled table is being set or retrieved.

GET

If present and nonzero, then the specified table is being retrieved.

GFSR

If present and nonzero, then the specified GFSR table is being set or retrieved.

SET

If present and nonzero, then the specified table is being set.

Discussion

The values in table are initialized by the IMSL random number generators. The values are all positive except if you wish to reinitialize the array, in which case the first element of the array is input as a nonpositive value. (Usually, one should avoid reinitializing these arrays, but it might be necessary sometimes in restarting a simulation.) If the first element of table is set to a nonpositive value on the call to `IMSL_RANDOM_TABLE` with the keyword *Set*, on the next invocation of a routine to generate random numbers, the appropriate table will be reinitialized.

For more details on the shuffled and GFSR generators see the [“Overview: Random Number Generation”](#) on page 1066.

Example

In this example, three separate simulation streams are used, each with a different form of the generator. Each stream is stopped and restarted. (Although this example is obviously an artificial one, there may be reasons for maintaining separate streams and stopping and restarting them because of the nature of the usage of the random numbers coming from the separate streams.)

```
nr = 5
iseed1 = 123457
iseed2 = 123457
iseed7 = 123457

; Begin first stream, iopt = 1 (by default)
IMSL_RANDOMOPT, Set = iseed1
r = IMSL_RANDOM(nr)
IMSL_RANDOMOPT, Get = iseed1
PM, r, TITLE = 'First stream output'

First stream output
    0.966220
    0.260711
    0.766262
    0.569337
    0.844829

PRINT, 'output seed ', iseed1

output seed    1814256879

; Begin second stream, iopt = 2
IMSL_RANDOMOPT, gen_opt = 2
IMSL_RANDOMOPT, Set = iseed2
```

```

r = IMSL_RANDOM(nr)
IMSL_RANDOMOPT, Get = iseed2
IMSL_RANDOM_TABLE, table, /Get
PM, r, TITLE = 'Second stream output'

Second stream output
    0.709518
    0.186145
    0.479442
    0.603839
    0.379015

PRINT, 'output seed ', iseed2

output seed    1965912801

; Begin third stream, iopt = 7
IMSL_RANDOMOPT, gen_opt = 7
IMSL_RANDOMOPT, Set = iseed7
r = IMSL_RANDOM(nr)
IMSL_RANDOMOPT, Get = iseed7
IMSL_RANDOM_TABLE, itable, /Get, /GFSR
PM, r, TITLE = 'Third stream output'

Third stream output
    0.391352
    0.0262676
    0.762180
    0.0280987
    0.899731

PRINT, 'output seed ', iseed7

output seed    1932158269

; Reinitialize seed and resume first stream
IMSL_RANDOMOPT, gen_opt = 1
IMSL_RANDOMOPT, Set = iseed1
r = IMSL_RANDOM(nr)
IMSL_RANDOMOPT, Get = iseed1
PM, r, TITLE = 'First stream output'

First stream output
    0.0442665
    0.987184
    0.601350
    0.896375
    0.380854

```

```

PRINT, 'output seed ', iseed1

output seed      817878095

; Reinitialize seed & table for shuffling & resume second stream
IMSL_RANDOMOPT,  gen_opt = 2
IMSL_RANDOMOPT,  Set = iseed2
IMSL_RANDOM_TABLE, table, /Set
r = IMSL_RANDOM(nr)
IMSL_RANDOMOPT,  Get = iseed2
PM, r, TITLE = 'Second stream output'

Second stream output
    0.255690
    0.478770
    0.225802
    0.345467
    0.581051

PRINT, 'output seed ', iseed2

output seed      2108806573

; Reinitialize seed and table for GFSR and resume third stream.
IMSL_RANDOMOPT,  GEN_OPT = 7
IMSL_RANDOMOPT,  SET = iseed7
IMSL_RANDOM_TABLE, itable, /SET, /GFSR
r = IMSL_RANDOM(nr)
IMSL_RANDOMOPT,  GET = iseed7
PM, r, TITLE = 'Third stream output'

Third stream output
    0.751854
    0.508370
    0.906986
    0.0910035
    0.691663

PRINT, 'output seed ', iseed7

output seed      1485334679

```

Version History

6.4	Introduced

IMSL_RANDOM

The IMSL_RANDOM function generates pseudorandom numbers. The default distribution is a uniform (0, 1) distribution, but many different distributions can be specified through the use of keywords.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_RANDOM(n [, /BETA] [, /BINOMIAL] [, /CAUCHY]
    [, /COVARIANCES=value] [, /CHI_SQUARED] [, /DISCRETE_UNIF]
    [, /DOUBLE] [, /EXPONENTIAL] [, /GAMMA] [, /GEOMETRIC]
    [, /HYPERGEOMETRIC] [, /LOGARITHMIC] [, /LOGNORMAL]
    [, /MIX_EXPONENTIAL] [, /MULTINOMIAL] [, /MVAR_NORMAL]
    [, /NEG_BINOMIAL] [, /NORMAL] [, /PARAMETERS=value]
    [, /PERMUTATION] [, /POISSON] [, /PROBABILITIES=array]
    [, /SAMPLE_INDICES] [, /SPHERE] [, /STABLE] [, /STUDENT_T]
    [, /TRIANGULAR] [, /UNIFORM] [, /VON_MISES] [, /WEIBULL])
```

Generally, it is best to first identify the desired distribution from the “*Discussion*” section, then refer to the “*Input Keywords*” section for specific usage instructions.

Return Value

A one-dimensional array of length n containing the random numbers. If one of the keywords *Sphere*, *Multinomial*, or *Mvar_Normal* are used, then a two-dimensional array is returned.

Arguments

n

Number of random numbers to generate.

Keywords

BETA

If present and nonzero, the random numbers are generated from a beta distribution. Requires the *Parameters* keyword to specify the parameters (p , q) for the distribution. The parameters p and q must be positive.

BINOMIAL

If present and nonzero, random numbers are generated from a binomial distribution. Requires *Parameters* keyword to specify the parameters (p , n) for the distribution. The parameter n is the number of Bernoulli trials, and it must be greater than zero. The parameter p represents the probability of success on each trial, and it must be between 0.0 and 1.0.

CAUCHY

If present and nonzero, the random numbers are generated from a Cauchy distribution.

COVARIANCES

Two-dimensional, square matrix containing the variance-covariance matrix. The two-dimensional array returned by `IMSL_RANDOM` is of the following size:

n by `N_ELEMENTS(Covariances(*, 0))`

Keywords *Mvar_Normal* and *Covariances* must be specified to return numbers from a multivariate normal distribution.

CHI_SQUARED

If present and nonzero, the random numbers are generated from a chi-squared distribution. Requires the *Parameters* keyword to specify the parameter Df for the distribution. The parameter Df is the number of degrees of freedom for the distribution, and it must be positive.

DISCRETE_UNIF

If present and nonzero, the random numbers are generated from a discrete uniform distribution. Requires the *Parameters* keyword to specify the parameter k for the distribution. This generates integers in the range from 1 to k (inclusive) with equal probability. The parameter k must be positive.

DOUBLE

If present and nonzero, double precision is used.

EXPONENTIAL

If present and nonzero, the random numbers are generated from a standard exponential distribution.

GAMMA

If present and nonzero, the random numbers are generated from a standard Gamma distribution. Requires the *Parameters* keyword to specify the parameter a for the distribution. The parameter a is the shape parameter of the distribution, and it must be positive n .

GEOMETRIC

If present and nonzero, the random numbers are generated from a geometric distribution. Requires the *Parameters* keyword to specify the parameter P for the distribution. The parameter P must be positive and less than 1.0.

HYPERGEOMETRIC

If present and nonzero, the random numbers are generated from a hypergeometric distribution. Requires the *Parameters* keyword to specify the parameters (M, N, L) for the distribution. The parameter N represents the number of items in the sample, M is the number of special items in the population, and L is the total number of items in the population. The parameters N and M must be greater than zero, and L must be greater than both N and M .

LOGARITHMIC

If present and nonzero, the random numbers are generated from a logarithmic distribution. Requires the *Parameters* keyword to specify the parameter a for the distribution. The parameter a must be greater than zero.

LOGNORMAL

If present and nonzero, the random numbers are generated from a lognormal distribution. Requires the *Parameters* keyword to specify the parameters (μ, σ) for the distribution. The parameter μ is the mean of the distribution, while σ represents the standard deviation.

MIX_EXPONENTIAL

If present and nonzero, the random numbers are generated from a mixture of two exponential distributions. Requires the *Parameters* keyword to specify the parameters (θ_1, θ_2, p) for the distribution. The parameters θ_1 and θ_2 are the means for the two distributions; both must be positive, and θ_1 must be greater than θ_2 . The parameter p is the relative probability of the θ_1 distribution, and it must be non-negative and less than or equal to $\theta_1/(\theta_1 - \theta_2)$.

NEG_BINOMIAL

If present and nonzero, the random numbers are generated from a negative binomial distribution. Requires the *Parameters* keyword to specify the parameters (r, p) for the distribution. The parameter r must be greater than zero. If r is an integer, the generated deviates can be thought of as the number of failures in a sequence of Bernoulli trials before r successes occur. The parameter p is the probability of success on each trial. It must be greater than the machine epsilon, and less than 1.0.

MULTINOMIAL

If present and nonzero, the random numbers are generated from a multinomial distribution. Requires the *Parameters* keyword to specify the parameter (*ntrials*) for the distribution, and the keyword *Probabilities* to specify the array containing the probabilities of the possible outcomes. The value if *ntrials* is the multinomial parameter indicating the number of independent trials.

MVAR_NORMAL

If present and nonzero, the random numbers are generated from a multivariate normal distribution. Keywords *Mvar_Normal* and *Covariances* must be specified to return numbers from a multivariate normal distribution.

NORMAL

If present and nonzero, the random numbers are generated from a standard normal distribution using an inverse CDF method.

PARAMETERS

Specifies parameters for the distribution used by IMSL_RANDOM to generate numbers. Some distributions require this keyword to execute successfully. The type and range of these parameters depends upon which distribution is specified. See the keyword for the desired distribution or the *Discussion* section for more details.

Note

The keywords *A*, *Pin*, *Qin*, and *Theta* are still supported, but are now deprecated. Please use the *Parameters* keyword instead.

PERMUTATION

If present and nonzero, then generate a pseudorandom permutation.

POISSON

If present and nonzero, the random numbers are generated from a Poisson distribution. Requires the *Parameters* keyword to specify the parameter θ for the distribution. The parameter θ represents the mean of the distribution, and it must be positive.

PROBABILITIES

Specifies the array containing the probabilities of the possible outcomes. The elements of *P* must be positive and must sum to 1.0.

Keywords *Multinomial* and *Probabilities* must be specified to return numbers from a Multinomial distribution.

SAMPLE_INDICES

If present and nonzero, generate a simple pseudorandom sample of indices. Requires the *Parameters* keyword to specify the parameter *npop*, the number of items in the population.

SPHERE

If present and nonzero, the random numbers are generated on a unit circle or K-dimensional sphere. Requires the *Parameters* keyword to specify the parameter *k*, the dimension of the circle ($k = 2$) or of the sphere.

STABLE

If present and nonzero, the random numbers are generated from a stable distribution. Requires the *Parameters* keyword to specify the parameters *A* and *bprime* for the stable distribution. *A* is the characteristic exponent of the stable distribution. *A* must be positive and less than or equal to 2. *bprime* is related to the usual skewness parameter *b* of the stable distribution.

STUDENT_T

If present and nonzero, the random numbers are generated from a Student's t distribution. Requires the *Parameters* keyword to specify the parameter Df for the distribution. The Df parameter is the number of degrees of freedom for the distribution, and it must be positive.

TRIANGULAR

If present and nonzero, the random numbers are generated from a triangular distribution.

UNIFORM

If present and nonzero, the random numbers are generated from a uniform (0, 1) distribution. The default action of this returns random numbers from a uniform (0, 1) distribution.

VON_MISES

If present and nonzero, the random numbers are generated from a von Mises distribution. Requires the *Parameters* keyword to specify the parameter c for the function. The parameter c must be greater than one-half the machine epsilon.

WEIBULL

If present and nonzero, the random numbers are generated from a Weibull distribution. Requires the *Parameters* keyword to specify the parameters (a , b) for the distribution. The parameter a is the shape parameter, and it is required. The parameter b is the scale parameter, and is optional (Default: $b = 1.0$).

Discussion

The `IMSL_RANDOM` function is designed to return random numbers from any of a number of different distributions. The determination of which distribution to generate the random numbers from is based on the presence of a keyword or groups of keywords. If `IMSL_RANDOM` is called without any keywords, then random numbers from a uniform (0, 1) distribution are returned.

Uniform (0,1) Distribution

The default action of `IMSL_RANDOM` generates pseudorandom numbers from a uniform (0, 1) distribution using a multiplicative, congruential method. The form of the generator follows:

$$x_i \equiv cx_{i-1} \bmod (2^{31} - 1)$$

Each x_i is then scaled into the unit interval (0, 1). The possible values for c in the generators are 16807, 397204094, and 950706376. The selection is made by using the `IMSL_RANDOMOPT` procedure with the *Gen_Option* keyword. The choice of 16807 results in the fastest execution time. If no selection is made explicitly, the functions use the multiplier 16807. See the “[IMSL_RANDOMOPT](#)” on page 1071 for further discussion of generator options.

The `IMSL_RANDOMOPT` procedure called with the *Set* keyword is used to initialize the seed of the random-number generator.

You can select a shuffled version of these generators. In this scheme, a table is filled with the first 128 uniform (0, 1) numbers resulting from the simple multiplicative congruential generator. Then, for each x_i from the simple generator, the low-order bits of x_i are used to select a random integer, j , from 1 to 128. The j -th entry in the table is then delivered as the random number, and x_i , after being scaled into the unit interval, is inserted into the j -th position in the table.

The values returned are positive and less than 1.0. Some values returned may be smaller than the smallest relative spacing; however, it may be the case that some value, for example $r(i)$, is such that $1.0 - r(i) = 1.0$.

Deviates from the distribution with uniform density over the interval (a, b) can be obtained by scaling the output. See “[Example 3: Beta Distribution](#)” on page 1097 for more details.

Normal Distribution

Calling `IMSL_RANDOM` with keyword *Normal* generates pseudorandom numbers from a standard normal (Gaussian) distribution using an inverse CDF technique. In this method, a uniform (0,1) random deviate is generated. Then, the inverse of the normal distribution function is evaluated at that point using the `IMSL_NORMALCDF` function with keyword *Inverse*.

If the *Parameters* keyword is specified in addition to *Normal*, `IMSL_RANDOM` generates pseudorandom numbers using an acceptance/rejection technique due to Kinderman and Ramage (1976). In this method, the normal density is represented as a mixture of densities over which a variety of acceptance/rejection methods due to Marsaglia (1964), Marsaglia and Bray (1964), and Marsaglia et al. (1964) are applied. This method is faster than the inverse CDF technique.

Deviates from the normal distribution with mean specific mean and standard deviation can be obtained by scaling the output from `IMSL_RANDOM`. See “[Example 3: Beta Distribution](#)” on page 1097 for more details.

Exponential Distribution

Calling `IMSL_RANDOM` with keyword *Exponential* generates pseudorandom numbers from a standard exponential distribution. The probability density function is $f(x) = e^{-x}$, for $x > 0$. The `IMSL_RANDOM` function uses an antithetic inverse CDF technique. In other words, a uniform random deviate U is generated, and the inverse of the exponential cumulative distribution function is evaluated at $1.0 - U$ to yield the exponential deviate.

Poisson Distribution

Calling `IMSL_RANDOM` with keywords *Poisson* and *Parameters = θ* generates pseudorandom numbers from a Poisson distribution with positive mean θ . The probability function follows:

$$f(x) = (e^{-\theta} \theta^x) / x! , \quad \text{for } x = 0, 1, 2, \dots$$

If θ is less than 15, `IMSL_RANDOM` uses an inverse CDF method; otherwise, the PTPE method of Schmeiser and Kachitvichyanukul (1981) is used. (See also Schmeiser 1983.) The PTPE method uses a composition of four regions, a triangle, a parallelogram, and two negative exponentials. In each region except the triangle, acceptance/rejection is used. The execution time of the method is essentially insensitive to the mean of the Poisson.

Gamma Distribution

Calling `IMSL_RANDOM` with keywords *Gamma* and *Parameters = a* generates pseudorandom numbers from a Gamma distribution with shape parameter a and unit scale parameter. The probability density function follows:

$$f(x) = \frac{1}{\Gamma(a)} x^{a-1} e^{-x} \quad \text{for } x \geq 0$$

Various computational algorithms are used depending on the value of the shape parameter a . For the special case of $a = 0.5$, squared and halved normal deviates are used; for the special case of $a = 1.0$, exponential deviates are generated. Otherwise, if a is less than 1.0, an acceptance-rejection method due to Ahrens, described in Ahrens and Dieter (1974), is used. If a is greater than 1.0, a 10-region rejection procedure developed by Schmeiser and Lal (1980) is used.

The Erlang distribution is a standard Gamma distribution with the shape parameter having a value equal to a positive integer; hence, `IMSL_RANDOM` generates pseudorandom deviates from an Erlang distribution with no modifications required.

Beta Distribution

Calling `IMSL_RANDOM` with keywords *Beta*, and *Parameters*=[*p,q*] generates pseudorandom numbers from a beta distribution. With *p* and *q* both positive, the probability density function is:

$$f(x) = \frac{\Gamma(p+q)}{\Gamma(p)\Gamma(q)} x^{p-1} (1-x)^{q-1}$$

where $\Gamma(\cdot)$ is the Gamma function.

The algorithm used depends on the values of *p* and *q*. Except for the trivial cases of *p* = 1 or *q* = 1, in which the inverse CDF method is used, all the methods use acceptance/rejection. If *p* and *q* are both less than 1, the method of Jöhnk (1964) is used. If either *p* or *q* is less than 1 and the other is greater than 1, the method of Atkinson (1979) is used. If both *p* and *q* are greater than 1, algorithm BB of Cheng (1978), which requires very little setup time, is used if *x* is less than 4, and algorithm B4PE of Schmeiser and Babu (1980) is used if *x* is 4 or greater. Note that for *p* and *q* both greater than 1, calling `IMSL_RANDOM` to generate random numbers from a beta distribution a loop getting less than four variates on each call yields the same set of deviates as executing one call and getting all deviates at once.

The values returned are less than 1.0 and greater than ϵ , where ϵ is the smallest positive number such that $1.0 - \epsilon$ is less than 1.0.

Multivariate Normal Distribution

Calling `IMSL_RANDOM` with keywords *Mvar_Normal* and *Covariances* generates pseudorandom numbers from a multivariate normal distribution with mean vector consisting of all zeros and variance-covariance matrix defined using keyword *Covariances*. First, the Cholesky factor of the variance-covariance matrix is computed. Then, independent random normal deviates with mean zero and variance 1 are generated, and the matrix containing these deviates is postmultiplied by the Cholesky factor. Because the Cholesky factorization is performed in each invocation, it is best to generate as many random vectors as needed at once.

Deviates from a multivariate normal distribution with means other than zero can be generated by using `IMSL_RANDOM` with keywords *Mvar_Normal* and *Covariances*, then adding the vectors of means to each row of the result.

Binomial Distribution

Calling `IMSL_RANDOM` with keywords *Binomial*, *Parameters*= [*p, n*] generates pseudorandom numbers from a binomial distribution with parameters *n* and *p*. Parameters *n* and *p* must be positive, and *p* must be less than 1. The probability function (where *n* = *Binom_n* and *p* = *Binom_p*) is:

$$f(x) = \binom{n}{x} p^x (1-p)^{n-x}$$

for $x = 0, 1, 2, \dots, n$.

The algorithm used depends on the values of n and p . If $n * p < 10$ or p is less than machine epsilon, the inverse CDF technique is used; otherwise, the BTPE algorithm of Kachitvichyanukul and Schmeiser (see Kachitvichyanukul 1982) is used. This is an acceptance /rejection method using a composition of four regions. (TPE=Triangle, Parallelogram, Exponential, left and right.)

Cauchy Distribution

Calling IMSL_RANDOM with the keyword *Cauchy* generates pseudorandom numbers from a Cauchy distribution. The probability density function is:

$$f(x) = \frac{S}{\pi[S^2 + (x - T)^2]}$$

where T is the median and $T - S$ is the first quartile. This function first generates standard Cauchy random numbers ($T = 0$ and $S = 1$) using the technique described below, and then scales the values using T and S .

Use of the inverse CDF technique would yield a Cauchy deviate from a uniform (0, 1) deviate, u , as $\tan[\pi(u - 0.5)]$. Rather than evaluating a tangent directly, however, IMSL_RANDOM generates two uniform $(-1, 1)$ deviates, x_1 and x_2 . These values can be thought of as sine and cosine values. If:

$$\frac{x_1^2}{1} + \frac{x_2^2}{2}$$

is less than or equal to 1, then x_1/x_2 is delivered as the unscaled Cauchy deviate; otherwise, x_1 and x_2 are rejected and two new uniform $(-1, 1)$ deviates are generated. This method is also equivalent to taking the ration of two independent normal deviates.

Chi-squared Distribution

Calling IMSL_RANDOM with keywords *Chi_squared* and *Parameters=Df* generates pseudorandom numbers from a chi-squared distribution with Df degrees of freedom. If Df is an even integer less than 17, the chi-squared deviate r is generated as:

$$r = -2 \ln \left(\prod_{i=1}^n u_i \right)$$

where $n = Df/2$ and the u_i are independent random deviates from a uniform (0, 1) distribution. If Df is an odd integer less than 17, the chi-squared deviate is generated

in the same way, except the square of a normal deviate is added to the expression above. If Df is greater than 16 or is not an integer, and if it is not too large to cause overflow in the gamma random number generator, the chi-squared deviate is generated as a special case of a gamma deviate.

Mixed Exponential Distribution

Calling `IMSL_RANDOM` with keywords *Mix_Exponential*, and *Parameters* = $[\theta_1, \theta_2]$ generates pseudorandom numbers from a mixture of two exponential distributions. The probability density function is:

$$f(x) = \frac{p}{\theta_1} e^{-x/\theta_1} + \frac{1-p}{\theta_2} e^{-x/\theta_2}$$

for $x > 0$.

In the case of a convex mixture, that is, the case $0 < p < 1$, the mixing parameter p is interpretable as a probability; and `IMSL_RANDOM` with probability p generates an exponential deviate with mean θ_1 , and with probability $1 - p$ generates an exponential with mean θ_2 . When p is greater than 1, but less than $\theta_1/(\theta_1 - \theta_2)$, then either an exponential deviate with mean θ_1 or the sum of two exponentials with means θ_1 and θ_2 is generated. The probabilities are $q = p - (p - 1)(\theta_1/\theta_2)$ and $1 - q$, respectively, for the single exponential and the sum of the two exponentials.

Geometric Distribution

Calling `IMSL_RANDOM` with keywords *Geometric* and *Parameters* = P generates pseudorandom numbers from a geometric distribution. The parameter P is the probability of getting a success on any trial. A geometric deviate can be interpreted as the number of trials until the first success (including the trial in which the first success is obtained). The probability function is:

$$f(x) = P(1 - P)^{x-1}$$

for $x = 1, 2, \dots$ and $0 < P < 1$.

The geometric distribution as defined above has mean $1/P$.

The i -th geometric deviate is generated as the smallest integer not less than $(\log(U_i))/(\log(1 - P))$, where the U_i are independent uniform(0, 1) random numbers (see Knuth 1981).

The geometric distribution is often defined on 0, 1, 2, ..., with mean $(1 - P)/P$. Such deviates can be obtained by subtracting 1 from each element of the returned vector of random deviates.

Hypergeometric Distribution

Calling `IMSL_RANDOM` with keywords *Hypergeometric*, and *Parameter*=[*M*, *N*, *L*,] generates pseudorandom numbers from a hypergeometric distribution with parameters *N*, *M*, and *L*. The hypergeometric random variable *X* can be thought of as the number of items of a given type in a random sample of size *N* that is drawn without replacement from a population of size *L* containing *M* items of this type. The probability function is:

$$f(x) = \frac{\binom{M}{x} \binom{L-M}{N-x}}{\binom{L}{N}}$$

for $x = \max(0, N - L + M), 1, 2, \dots, \min(N, M)$

If the hypergeometric probability function with parameters *N*, *M*, and *L* evaluated at $N - L + M$ (or at 0 if this is negative) is greater than the machine, and less than 1.0 minus the machine epsilon, then `IMSL_RANDOM` uses the inverse CDF technique. The routine recursively computes the hypergeometric probabilities, starting at $x = \max(0, N - L + M)$ and using the ratio:

$$\frac{f(X = x + 1)}{f(X = x)}$$

(see Fishman 1978, p. 475).

If the hypergeometric probability function is too small or too close to 1.0, then `IMSL_RANDOM` generates integer deviates uniformly in the interval $[1, L - i]$ for $i = 0, 1, \dots$, and at the i -th step, if the generated deviate is less than or equal to the number of special items remaining in the lot, the occurrence of one special item is tallied and the number of remaining special items is decreased by one. This process continues until the sample size of the number of special items in the lot is reached, whichever comes first. This method can be much slower than the inverse CDF technique. The timing depends on *N*. If *N* is more than half of *L* (which in practical examples is rarely the case), You may wish to modify the problem, replacing *N* by $L - N$, and to consider the generated deviates to be the number of special items not included in the sample.

Logarithmic Distribution

Calling `IMSL_RANDOM` with keywords *Logarithmic* and *Parameter*=*a* generates pseudorandom numbers from a logarithmic distribution. The probability function is:

$$f(x) = \frac{a^x}{x \ln(1 - a)}$$

for $x = 1, 2, 3, \dots$, and $0 < a < 1$

The methods used are described by Kemp (1981) and depend on the value of a . If a is less than 0.95, Kemp's algorithm LS, which is a "chop-down" variant of an inverse CDF technique, is used. Otherwise, Kemp's algorithm LK, which gives special treatment to the highly probable values of 1 and 2 is used.

Lognormal Distribution

Calling `IMSL_RANDOM` with keywords *Lognormal*, and *Parameter* = $[\mu, \sigma]$ generates pseudorandom numbers from a lognormal distribution. The scale parameter σ in the underlying normal distribution must be positive. The method is to generate normal deviates with mean μ and standard deviation Σ and then to exponentiate the normal deviates.

The probability density function for the lognormal distribution is:

$$f(x) = \frac{1}{\sigma x \sqrt{2\pi}} \exp\left[-\frac{1}{2\sigma^2}(\ln x - \mu)^2\right]$$

for $x > 0$. The mean and variance of the lognormal distribution are $\exp(\mu + \sigma^2/2)$ and $\exp(2\mu + 2\sigma^2) - \exp(2\mu + \sigma^2)$, respectively.

Negative Binomial

Calling `IMSL_RANDOM` with keywords *Neg_binomial* and *Parameters* = $[r, p]$ generates pseudorandom numbers from a negative binomial distribution. The parameters r and p must be positive and p must be less than 1. The probability function is:

$$f(x) = \binom{r+x-1}{x} (1-p)^r p^x$$

for $x = 0, 1, 2, \dots$

If r is an integer, the distribution is often called the Pascal distribution and can be thought of as modeling the length of a sequence of Bernoulli trials until r successes are obtained, where p is the probability of getting a success on any trial. In this form, the random variable takes values $r, r+1, r+2, \dots$ and can be obtained from the negative binomial random variable defined above by adding r to the negative binomial variable defined by adding r to the negative binomial variable. This latter

form is also equivalent to the sum of r geometric random variables defined as taking values 1, 2, 3, ...

If $rp/(1-p)$ is less than 100 and $(1-p)^r$ is greater than the machine epsilon, *IMSL_RANDOM* uses the inverse CDF technique; otherwise, for each negative binomial deviate, *IMSL_RANDOM* generates a *gamma* ($r, p/(1-p)$) deviate Y and then generates a Poisson deviate with parameter Y .

Discrete Uniform Distribution

Calling *IMSL_RANDOM* with keywords *Discrete_unif* and *Parameters = k* generates pseudorandom numbers from a uniform discrete distribution over the integers 1, 2, ..., k . A random integer is generated by multiplying k by a uniform (0, 1) random number, adding 1.0, and truncating the result to an integer. This, of course, is equivalent to sampling with replacement from a finite population of size k .

Student's t Distribution

Calling *IMSL_RANDOM* with keywords *Students_t* and *Parameters=Df* generates pseudorandom numbers from a Student's t distribution with Df degrees of freedom, using a method suggested by Kinderman et al. (1977). The method ("TMX" in the reference) involves a representation of the t density as the sum of a triangular density over $(-2, 2)$ and the difference of this and the t density. The mixing probabilities depend on the degrees of freedom of the t distribution. If the triangular density is chosen, the variate is generated as the sum of two uniforms; otherwise, an acceptance/rejection method is used to generate the difference density.

Triangular Distribution

Calling *IMSL_RANDOM* with the keyword *Triangular* generates pseudorandom numbers from a triangular distribution over the unit interval. The probability density function is $f(x) = 4x$, for $0 \leq x \leq 0.5$, and $f(x) = 4(1-x)$, for $0.5 < x \leq 1$. An inverse CDF technique is used.

von Mises Distribution

Calling *IMSL_RANDOM* with keywords *Von_mises* and *Parameters = c* generates pseudorandom numbers from a von Mises distribution where c must be positive. The probability density function is:

$$f(x) = \frac{1}{2\pi I_0(c)} \exp[c \cos(x)]$$

for $-\pi < x < \pi$, where $I_0(c)$ is the modified Bessel function of the first kind of order 0. The probability density is equal to 0 outside the interval $(-\pi, \pi)$.

The algorithm is an acceptance/rejection method using a wrapped Cauchy distribution as the majorizing distribution. It is due to Nest and Fisher (1979).

Weibull Distribution

Calling `IMSL_RANDOM` with keywords *Weibull* and *Parameters*=[*a*,*b*] generates pseudorandom numbers from a Weibull distribution with shape parameter *a* and scale parameter *b*. The probability density function is:

$$f(x) = abx^{a-1}\exp(-bx^a)$$

for $x \geq 0$, $a > 0$, and $b > 0$. The value of *b* is optional; if it is not specified, it is set to 1.0.

The `IMSL_RANDOM` function uses an antithetic inverse CDF technique to generate a Weibull variate; that is, a uniform random deviate *U* is generated and the inverse of the Weibull cumulative distribution function is evaluated at $1.0 - U$ to yield the Weibull deviate.

Note that the Rayleigh distribution with probability density function:

$$r(x) = \frac{1}{\alpha^2} x e^{-(x^2/(2\alpha^2))}$$

for $x \geq 0$ is the same as a Weibull distribution with shape parameter *a* equal to 2 and scale parameter *b* equal to:

$$\sqrt{2}\alpha$$

Stable Distribution

Calling `IMSL_RANDOM` with keywords *Stable* and *Parameters* = [*α*, *β*'] generates pseudorandom numbers from a stable distribution with parameters *α*' and *β*'. *α* is the usual characteristic exponent parameter *α*, and *β*' is related to the usual skewness parameter *β* of the stable distribution. With restrictions $0 < \alpha \leq 2$ and $-1 \leq \beta \leq 1$, the characteristic function of the distribution is:

$$\phi(t) = \exp[-|t|^\alpha \exp(-i\beta (1 - |1 - \alpha|)\text{sign}(t)/2)] \text{ for } \alpha \neq 1$$

and

$$\phi(t) = \exp[-|t|(1 + 2i\beta \ln|t|)\text{sign}(t)/\pi)] \text{ for } \alpha = 1$$

When $\beta = 0$, the distribution is symmetric. In this case, if $\alpha = 2$, the distribution is normal with mean 0 and variance 2; and if $\alpha = 1$, the distribution is Cauchy.

The parameterization using *β*' and the algorithm used here are due to Chambers, Mallows, and Stuck (1976). The relationship between *β*' and the standard *β* is:

$$\beta' = -\tan(\pi (1 - \alpha)/2) \tan(-\pi \beta (1 - |1 - \alpha|)/2) \text{ for } \alpha \neq 1$$

and:

$$\beta' = \beta \text{ for } \alpha = 1$$

The algorithm involves formation of the ratio of a uniform and an exponential random variate.

Multinomial Distribution

Calling `IMSL_RANDOM` with keywords *Multinomial*, *Probabilities*, and *Parameters = ntrials* generates pseudorandom numbers from a K-variate multinomial distribution with parameters n and p . $k = \text{N\_ELEMENTS}(\text{Probabilities})$ and $ntrials$ must be positive. Each element of *Probabilities* must be positive and the elements must sum to 1. The probability function (with $n = n$, $k = k$, and $p_i = \text{Probabilities}(i)$) is:

$$f(x_1, x_2, \dots, x_k) = \frac{n!}{x_1! x_2! \dots x_k!} p_1^{x_1} p_2^{x_2} \dots p_k^{x_k}$$

$$\text{for } x_i: \sum_{i=1}^{k-1} x_i = n$$

The deviate in each row of r is produced by generation of the binomial deviate x_0 with parameters n and p_1 and then by successive generations of the conditional binomial deviates x_j given $x_0, x_1, \dots, x_{j-2}$ with parameters $n - x_0 - x_1 - \dots - x_{j-2}$ and $p_j / (1 - p_0 - p_1 - \dots - p_{j-2})$.

Random Points on a K-dimensional Sphere

Calling `IMSL_RANDOM` with the keywords *Sphere* and *Parameters = k* generates pseudorandom coordinates of points that lie on a unit circle or a unit sphere in K-dimensional space. For points on a circle ($k = 2$), pairs of uniform $(-1, 1)$ points are generated and accepted only if they fall within the unit circle (the sum of their squares is less than 1), in which case they are scaled so as to lie on the circle.

For spheres in three or four dimensions, the algorithms of Marsaglia (1972) are used. For three dimensions, two independent uniform $(-1, 1)$ deviates U_1 and U_2 are generated and accepted only if the sum of their squares S_1 is less than 1. Then, the coordinates:

$$Z_1 = 2U_1\sqrt{1-S_1}, Z_2 = 2U_2\sqrt{1-S_1}, \text{ and } Z_3 = 1-2S_1$$

are formed. For four dimensions, U_1, U_2 , and S_1 are produced as described above. Similarly, U_3, U_4 , and S_2 are formed. The coordinates are then:

$$Z_1 = U_1, Z_2 = U_2, Z_3 = U_3 \sqrt{(1 - S_1)/S_2}$$

and:

$$Z_4 = U_4 \sqrt{(1 - S_1)/S_2}$$

For spheres in higher dimensions, K independent normal deviates are generated and scaled so as to lie on the unit sphere in the manner suggested by Muller (1959).

Random Permutation

Calling `IMSL_RANDOM` with the keyword *Permutation* generates a pseudorandom permutation of the integers from 1 to n . It begins by filling a vector of length n with the consecutive integers 1 to n . Then, with M initially equal to n , a random index J between 1 and M (inclusive) is generated. The element of the vector with the index M and the element with index J swap places in the vector. M is then decremented by 1 and the process repeated until $M = 1$.

Sample Indices

Calling `IMSL_RANDOM` with the keywords *Sample_indices* and *Parameters = npop* generates the indices of a pseudorandom sample, without replacement, of size n numbers from a population of size $npop$. If n is greater than $npop/2$, the integers from 1 to $npop$ are selected sequentially with a probability conditional on the number selected and the number remaining to be considered. If, when the i -th population index is considered, j items have been included in the sample, then the index i is included with probability $(n - j)/(npop + 1 - i)$.

If n is not greater than $npop/2$, a $O(n)$ algorithm due to Ahrens and Dieter (1985) is used. Of the methods discussed by Ahrens and Dieter, the one called SG^* is used. It involves a preliminary selection of q indices using a geometric distribution for the distances between each index and the next one. If the preliminary sample size q is less than n , a new preliminary sample is chosen, and this is continued until a preliminary sample greater in size than n is chosen. This preliminary sample is then thinned using the same kind of sampling as described above for the case in which the sample size is greater than half of the population size. This routine does not store the preliminary sample indices, but rather restores the state of the generator used in selecting the sample initially, and then passes through once again, making the final selection as the preliminary sample indices are being generated.

Examples

Example 1

In this example, `IMSL_RANDOM` is used to generate five pseudorandom, uniform numbers. Since `RANDOMOPT` is not called, the generator used is a simple multiplicative congruential one with a multiplier of 16807.

```
IMSL_RANDOMOPT, Set = 123457
; Set the random seed.
r = IMSL_RANDOM(5)
; Call IMSL_RANDOM to compute the random numbers.
PM, r
```

The results are something like:

```
0.966220
0.260711
0.766262
0.569337
0.844829
```

Example 2: Poisson Distribution

In this example, random numbers from a Poisson distribution are computed.

```
IMSL_RANDOMOPT, Set = 123457
r = IMSL_RANDOM(5, /POISSON, PARAMETERS = 0.5)
; Call IMSL_RANDOM with keywords Poisson and Parameters.
PM, r
```

Example 3: Beta Distribution

In this example, random numbers are computed from a Beta distribution.

```
IMSL_RANDOMOPT, set = 123457
r = IMSL_RANDOM(5, /Beta, Parameter = [3,2])
; Call IMSL_RANDOM with keywords Beta, Pin, and Qin.
PM, r
```

Example 4: Scaling the Results of `IMSL_RANDOM`

This example computes deviates with uniform density over the interval (10, 20) and deviates from the normal distribution with a mean of 10 and a standard deviation of 2.

```
IMSL_RANDOMOPT, Set = 123457
; Set the random number seed.
a = 10
; Define the lowerbound.
b = 20
```

```

; Define the upperbound.
r = a + (b - a) * IMSL_RANDOM(5)
; Call IMSL_RANDOM to compute the deviates on (0,1) and scale the
; results to (a,b).
PM, r

```

The results are something like:

```

19.6622
12.6071
17.6626
15.6934
18.4483

```

```

; Define a standard deviation.
stdev = 2
; Define a mean.
mean = 10
; Call IMSL_RANDOM to compute the deviates normal deviates
; and scale the results using the specified mean and standard
; deviation.
r = IMSL_RANDOM(6, /Normal) * stdev + mean
PM, r

```

The results are something like:

```

6.59363
14.4635
10.5137
12.5223
9.39352
5.71021

```

Example 5: Multivariate Normal Distribution

In this example, IMSL_RANDOM generates five pseudorandom normal vectors of length 2 with variance covariance matrix equal to the following:

$$\begin{bmatrix} 0.500 & 0.375 \\ 0.375 & 0.500 \end{bmatrix}$$

```

IMSL_RANDOMOPT, SET = 123457
; Set the random number seed.
cov = [[.5, .375], [.375, .5]]
; Define the covariance matrix.
PM, IMSL_RANDOM(5, /MVAR_NORMAL, COVARIANCES = cov)

```

The results are something like:

1.45068	1.24634
0.765975	-0.0429410
0.0583781	-0.669214
0.903489	0.462826
-0.866886	-0.933426

Version History

6.4	Introduced

IMSL_RANDOM_NPP

The IMSL_RANDOM_NPP function generates pseudorandom numbers from a nonhomogeneous Poisson process.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
result = IMSL_RANDOM_NPP(tbegin, tend, ftheta, theta_min, theta_max, neub  
[, /DOUBLE])
```

Return Value

A one dimensional array containing the times to events. If the length of the result is less than *neub*, the time *tend* is reached before *neub* events are realized

Arguments

neub

Upper bound on the number of events to be generated. In order to be reasonably sure that the full process through time *tend* is generated, calculate *neub* as $neub = X + 10.0 * \text{SQRT}(X)$, where $X = theta\_max * (tend - tbegin)$.

ftheta

Scalar string specifying a user-supplied function to provide the value of the rate of the process as a function of time. This function accepts one argument and must be defined over the interval from *tbegin* to *tend* and must be nonnegative in that interval.

tbegin

Lower endpoint of the time interval of the process. *tbegin* must be nonnegative. Usually, *tbegin* = 0.

tend

Upper endpoint of the time interval of the process. *tend* must be greater than *tbegin*.

theta_max

Maximum value of the rate function $ftheta$ in the interval $(tbegin, tend)$. If the actual maximum is unknown, set $theta\_max$ to a known upper bound of the maximum. The efficiency of IMSL_RANDOM_NPP is less the greater $theta\_max$ exceeds the true maximum.

theta_min

Minimum value of the rate function $ftheta()$ in the interval $(tbegin, tend)$. If the actual minimum is unknown, set $theta\_min = 0.0$.

Keywords**DOUBLE**

If present and nonzero, double precision is used.

Discussion

Routine IMSL_RANDOM_NPP simulates a one-dimensional nonhomogeneous Poisson process with rate function $theta$ in a fixed interval $(tend - tbegin)$.

Let $\lambda(t)$ be the rate function and $t_0 = tbegin$ and $t_1 = tend$. Routine IMSL_RANDOM_NPP uses a method of thinning a nonhomogeneous Poisson process $\{N^*(t), t \geq t_0\}$ with rate function $\lambda^*(t) \geq \lambda(t)$ in (t_0, t_1) , where the number of events, N^* , in the interval (t_0, t_1) has a Poisson distribution with parameter:

$$\mu_0 = \int_{t_0}^{t_1} \lambda(t) dt$$

The function:

$$\Lambda(t) = \int_0^{t'} \lambda(t) dt$$

is called the integrated rate function. In IMSL_RANDOM_NPP, $\lambda^*(t)$ is taken to be a constant $\lambda^*(= theta\_max)$ so that at time t_i , the time of the next event t_{i+1} is obtained by generating and cumulating exponential random numbers:

$$E_{1,i}^*, E_{2,i}^*, \dots$$

with parameter λ^* , until for the first time:

$$u_{j,i} \leq (t_i + E_{1,i}^* + \dots + E_{j,i}^*) / \lambda^*$$

where the $u_{j,i}$ are independent uniform random numbers between 0 and 1. This process is continued until the specified number of events, *neub*, is realized or until the time, *tend*, is exceeded. This method is due to Lewis and Shedler (1979), who also review other methods. The most straightforward (and most efficient) method is by inverting the integrated rate function, but often this is not possible.

If *theta_max* is actually greater than the maximum of $\lambda(t)$ in (t_0, t_1) , the routine will work, but less efficiently. Also, if $\lambda(t)$ varies greatly within the interval, the efficiency is reduced. In that case, it may be desirable to divide the time interval into subintervals within which the rate function is less variable. This is possible because the process is without memory.

If no time horizon arises naturally, *tend* must be set large enough to allow for the required number of events to be realized. Care must be taken, however, that *ftheta* is defined over the entire interval.

After simulating a given number of events, the next event can be generated by setting *tbegin* to the time of the last event (the sum of the elements in the result) and calling IMSL_RANDOM_NPP again. Cox and Lewis (1966) discuss modeling applications of nonhomogeneous Poisson processes.

Example

In this example, IMSL_RANDOM_NPP is used to generate the first five events in the time 0 to 20 (if that many events are realized) in a nonhomogeneous process with rate function:

$$\lambda(t) = 0.6342 e^{0.001427t}$$

for $0 < t \leq 20$.

Since this is a monotonically increasing function of *t*, the minimum is at *t* = 0 and is 0.6342, and the maximum is at *t* = 20 and is $0.6342 e^{0.02854} = 0.652561$.

```
.RUN
  FUNCTION ftheta_npp, t
    RETURN, .6342*exp(.001427*t)
  END

randomopt, set=123457
neub = 5
tmax = .652561
tmin = .6342
tbegin=0
tend=20
r = IMSL_RANDOM_NPP(tbegin, tend, 'ftheta_npp', tmin, tmax, neub)
PM, r
```

0.0526598
0.407979
0.258399
0.0197666
0.167641

Version History

6.4	Introduced

IMSL_RANDOM_ORDER

The `IMSL_RANDOM_ORDER` function generates pseudorandom order statistics from a uniform (0, 1) distribution, or optionally from a standard normal distribution.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
result = IMSL_RANDOM_ORDER(ifirst, ilast, n [, /DOUBLE] [, /NORMAL]  
                        [, /UNIFORM])
```

Return Value

An array of length $ilast + 1 - ifirst$ containing the random order statistics in ascending order.

The first element is the *ifirst* order statistic in a random sample of size n from the uniform (0, 1) distribution.

Arguments

ifirst

First order statistic to generate.

ilast

Last order statistic to generate. *ilast* must be greater than or equal to *ifirst*. The full set of order statistics from *ifirst* to *ilast* is generated. If only one order statistic is desired, set *ilast* = *ifirst*.

n

Size of the sample from which the order statistics arise.

Keywords

DOUBLE

If present and nonzero, double precision is used.

NORMAL

If present and nonzero, generate pseudorandom order statistics from a standard normal distribution.

UNIFORM

If present and nonzero, generate pseudorandom order statistics from a uniform (0, 1) distribution. (Default)

Discussion

Routine `IMSL_RANDOM_ORDER` generates the *ifirst* through the *ilast* order statistics from a pseudorandom sample of size *n* from a uniform (0, 1) distribution. Depending on the values of *ifirst* and *ilast*, different methods of generation are used to achieve greater efficiency. If *ifirst* = 1 and *ilast* = *n*, that is, if the full set of order statistics are desired, the spacings between successive order statistics are generated as ratios of exponential variates. If the full set is not desired, a beta variate is generated for one of the order statistics, and the others are generated as extreme order statistics from conditional uniform distributions. Extreme order statistics from a uniform distribution can be obtained by raising a uniform deviate to an appropriate power.

Each call to `IMSL_RANDOM_ORDER` yields an independent event. This means, for example, that if on one call the fourth order statistic is requested and on a second call the third order statistic is requested, the “fourth” may be smaller than the “third”. If both the third and fourth order statistics from a given sample are desired, they should be obtained from a single call to `IMSL_RANDOM_ORDER` (by specifying *ifirst* less than or equal to 3 and *ilast* greater than or equal to 4).

If the keyword *Normal* is present and nonzero, then `IMSL_RANDOM_ORDER` generates the *ifirst* through the *ilast* order statistics from a pseudorandom sample of size *n*, from a normal (0, 1) distribution

Example

In this example, `IMSL_RANDOM_ORDER` is used to generate the fifteenth through the nineteenth order statistics from a sample of size twenty.

```

r  = IMSL_RANDOM_ORDER(15, 19, 20)
pm, r

```

Version History

6.4	Introduced

IMSL_RANDOM_TABLE_2WAY

The IMSL_RANDOM_TABLE_2WAY function generates a pseudorandom two-way table.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
result = IMSL_RANDOM_TABLE_2WAY (row_totals, col_totals)
```

Return Value

A N_ELEMENTS(row_totals) by N_ELEMENTS(col_totals) random matrix with the given row and column totals.

Arguments

col_totals

One dimensional array containing the column totals. (Input) The elements of row_totals and col_totals must be nonnegative and must sum to the same quantity.

row_totals

One dimensional array containing the row totals.

Discussion

Routine IMSL_RANDOM_TABLE_2WAY generates pseudorandom entries for a two-way contingency table with fixed row and column totals. The method depends on the size of the table and the total number of entries in the table. If the total number of entries is less than twice the product of the number of rows and columns, the method described by Boyette (1979) and by Agresti, Wackerly, and Boyette (1979) is used. In this method, a work vector is filled with row indices so that the number of times each index appears equals the given row total. This vector is then randomly permuted and used to increment the entries in each row so that the given row total is attained.

For tables with larger numbers of entries, the method of Patefield (1981) is used. This method can be considerably faster in these cases. The method depends on the conditional probability distribution of individual elements, given the entries in the previous rows. The probabilities for the individual elements are computed starting from their conditional means.

Example

In this example, `IMSL_RANDOM_TABLE_2WAY` is used to generate a two by three table with row totals 3 and 5, and column totals 2, 4, and 2.

```
r      =  IMSL_RANDOM_TABLE_2WAY ([3, 5], [2, 4, 2])
PM, r

      2      1      0
      0      3      2
```

Version History

6.4	Introduced

IMSL_RAND_ORTH_MAT

The IMSL_RAND_ORTH_MAT function generates a pseudorandom orthogonal matrix or a correlation matrix.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_RAND_ORTH_MAT(n [, A_MATRIX=array] [, /DOUBLE]  
[, EIGENVALUES=array] )
```

Return Value

A two-dimensional array containing the n by n random correlation matrix.

Arguments

n

The order of the matrix to be generated.

Keywords

A_MATRIX

Two-dimensional array containing n by n random orthogonal matrix. A random correlation matrix is generated using orthogonal matrix input in *A_Matrix*. *Eigenvalues* must also be supplied if *A_Matrix* is used.

DOUBLE

If present and nonzero, double precision is used.

EIGENVALUES

One-dimensional array of length n containing the eigenvalues of the correlation matrix to be generated. The elements of *Eigenvalues* must be positive, they must sum to n , and they cannot all be equal.

Discussion

IMSL_RANDOM_ORTH_MAT generates a pseudorandom orthogonal matrix from the invariant Haar measure. For each column, a random vector from a uniform distribution on a hypersphere is selected and then is projected onto the orthogonal complement of the columns already formed. The method is described by Heiberger (1978). (See also Tanner and Thisted 1982.)

If *Eigenvalues* is used, a correlation matrix is formed by applying a sequence of planar rotations to matrix $A^T D A$, where $D = \text{diag}(\text{Eigenvalues}(0), \dots, \text{Eigenvalues}(n-1))$, so as to yield ones along the diagonal. The planar rotations are applied in such an order that in the two by two matrix that determines the rotation, one diagonal element is less than 1.0 and one is greater than 1.0. This method is discussed by Bendel and Mickey (1978) and by Lin and Bendel (1985).

The distribution of the correlation matrices produced by this method is not known. See Bendel and Mickey (1978) and Johnson and Welch (1980).

For larger matrices, rounding can become severe; and the double precision results may differ significantly from single precision results.

Example

In this example, IMSL_RANDOM_ORTH_MAT is used to generate a 4 by 4 pseudorandom correlation matrix with eigenvalues in the ratio 1:2:3:4.

```
IMSL_RANDOMOPT, set = 123457
a = IMSL_RANDOM_ORTH_MAT(4)
ev = .4d0*[1.0d0, 2.0d0, 3.0d0, 4.0d0]
cor = IMSL_RANDOM_ORTH_MAT(4, EIGENVALUES = ev, A_MATRIX= a)
PM, cor
```

```
1.00000    -0.235786    -0.325795    -0.110139
-0.235786     1.00000     0.190564    -0.0172391
-0.325795     0.190564     1.00000    -0.435339
-0.110139    -0.0172391    -0.435339     1.00000
```

Version History

6.4	Introduced

IMSL_RANDOM_SAMPLE

The IMSL_RANDOM_SAMPLE function generates a simple pseudorandom sample from a finite population.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_RANDOM_SAMPLE(nsamp, population [, /ADDITIONAL_CALL]  
[, /DOUBLE] [, /FIRST_CALL] [, INDEX=array] [, NPOP=value]  
[, SAMPLE=array] )
```

Return Value

nsamp by *nvar* array containing the sample, where *nvar* is the number of columns in the argument population.

Arguments

nsamp

The sample size desired.

population

A one or two dimensional array containing the population to be sampled. If either of the keywords *First_Call* or *Additional_Call* are specified, then population contains a different part of the population on each invocation, otherwise population contains the entire population.

Keywords

ADDITIONAL_CALL

If present and nonzero, then this is an additional invocation of IMSL_RANDOM_SAMPLE, and updating for the subpopulation in population is performed. Keywords *Index*, *Npop*, and *Sample* are required if *Additional_Call* is set.

It is not necessary to know the number of items in the population in advance. *Npop* is used to cumulate the population size and should not be changed between calls to `IMSL_RANDOM_SAMPLE`. See Example 2.

DOUBLE

If present and nonzero, double precision is used.

FIRST_CALL

If present and nonzero, then this is the first invocation with this data; additional calls to `IMSL_RANDOM_SAMPLE` may be made to add to the population. Additional calls should be made using the keyword *Additional_Call*. Keywords *Index* and *Npop* are required if *First_Call* is set. See Example 2.

INDEX

A one-dimensional array of length *nsamp* containing the indices of the sample in the population. Output if keyword *First_Call* is used. Input/Output if keyword *Additional_Call* is used.

NPOP

The number of items in the population. Output if keyword *First_Call* is used. Input/Output if keyword *Additional_Call* is used.

SAMPLE

An array of size *nsamp* by *nvar* containing the sample. Initially, the result of calling `IMSL_RANDOM_SAMPLE` with keyword *First_Call* is used for *Sample*.

Discussion

Routine `IMSL_RANDOM_SAMPLE` generates a pseudorandom sample from a given population, without replacement, using an algorithm due to McLeod and Bellhouse (1983).

The first *nsamp* items in the population are included in the sample. Then, for each successive item from the population, a random item in the sample is replaced by that item from the population with probability equal to the sample size divided by the number of population items that have been encountered at that time.

Examples

Example 1

In this example, IMSL_RANDOM_SAMPLE is used to generate a sample of size 5 from a population stored in the matrix population.

```
IMSL_RANDOMOPT, Set = 123457
pop = IMSL_STATDATA(2)
samp = IMSL_RANDOM_SAMPLE(5, pop)
PM, samp

1764.00      36.4000
1828.00      62.5000
1923.00       5.8000
1773.00      34.8000
1769.00     106.100
```

Example 2

Routine IMSL_RANDOM_SAMPLE is now used to generate a sample of size 5 from the same population as in the example above except the data are input to IMSL_RANDOM_SAMPLE one observation at a time. This is the way IMSL_RANDOM_SAMPLE may be used to sample from a file on disk or tape. Notice that the number of records need not be known in advance.

```
IMSL_RANDOMOPT, SET = 123457
pop = IMSL_STATDATA(2)
samp = IMSL_RANDOM_SAMPLE(5, pop(0, *), /FIRST_CALL, INDEX = ii, $
    NPOP=np)
FOR i=1,175 DO samp = IMSL_RANDOM_SAMPLE(5, pop(i, *), $
    /ADDITIONAL_CALL, INDEX = ii, NPOP = np, SAMPLE = samp)
PM, samp

1764.00      36.4000
1828.00      62.5000
1923.00       5.8000
1773.00      34.8000
1769.00     106.100
```

Version History

6.4	Introduced

IMSL_RAND_FROM_DATA

The IMSL_RAND_FROM_DATA function generates pseudorandom numbers from a multivariate distribution determined from a given sample.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_RAND_FROM_DATA(*n_random*, *x*, *nn* [, /DOUBLE])

Return Value

n x *ndim* matrix containing the random multivariate vectors in its rows.

Arguments

n_random

Number of random multivariate vectors to generate.

nn

Number of nearest neighbors of the randomly selected point in *x* that are used to form the output point in the result.

x

Two dimensional array of size *nsamp* by *ndim* containing the given sample.

Keywords

DOUBLE

If present and nonzero, double precision is used.

Discussion

Given a sample of size *nsamp* of observations of a *k*-variate random variable, IMSL_RANDOM_FROM_DATA generates a pseudorandom sample with approximately the same moments as the given sample. The sample obtained is the same as if sampling from a Gaussian kernel estimate of the sample density. (See Thompson 1989.) Routine IMSL_RANDOM_FROM_DATA uses methods described by Taylor and Thompson (1986).

Assume that the (vector-valued) observations x_i are in the rows of *x*. An observation, x_j , is chosen randomly; its nearest $m (= nn)$ neighbors:

$$x_{j_1}, x_{j_2}, \dots, x_{j_m}$$

are determined; and the mean:

$$\bar{x}_j$$

of those nearest neighbors is calculated. Next, a random sample $u_1, u_2, \dots, u_m$ is generated from a uniform distribution with lower bound:

$$\frac{1}{m} - \sqrt{\frac{3(m-1)}{m^2}}$$

and upper bound:

$$\frac{1}{m} + \sqrt{\frac{3(m-1)}{m^2}}$$

The random variate delivered is:

$$\sum_{l=1}^m u_l(x_{j_l} - \bar{x}_j) + \bar{x}_j$$

The process is then repeated until *n* such simulated variates are generated and stored in the rows of the result.

Example

In this example, IMSL_RANDOM_FROM_DATA is used to generate 5 pseudorandom vectors of length 4 using the initial and final systolic pressure and the initial and final diastolic pressure from Data Set A in Afifi and Azen (1979) as the fixed sample from the population to be modeled. (Values of these four variables are in the seventh, tenth, twenty-first, and twenty-fourth columns of data set number nine in routine IMSL_STATDATA, see [Chapter 25, “Math and Statistics Utilities”](#) of this manual).


```
IMSL_RANDOMOPT, Set = 123457
r = IMSL_STATDATA(9)
x = FLTARR(113, 4)
x(*, 0) = r(*,6)
x(*, 1) = r(*,9)
x(*, 2) = r(*,20)
x(*, 3) = r(*,23)
r = IMSL_RAND_FROM_DATA(5, x, 5)
PM, r
```

162.767	90.5057	153.717	104.877
153.353	78.3180	176.664	85.2155
93.6958	48.1675	153.549	71.3688
101.751	54.1855	113.121	56.2916
91.7403	58.7684	48.4368	28.0994

Version History

6.4	Introduced

IMSL_CONT_TABLE

The IMSL_CONT_TABLE procedure sets up table to generate pseudorandom numbers from a general continuous distribution.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

IMSL_CONT_TABLE, (*f*, *iopt*, *ndata*, *table* [, /DOUBLE])

Arguments

f

A scalar string specifying a user-supplied function to compute the cumulative distribution function. The argument to the function is the point at which the distribution function is to be evaluated.

iopt

Indicator of the extent to which table is initialized prior to calling IMSL_CONT_TABLE.

- 0—IMSL_CONT_TABLE fills the last four columns of table. Input the points at which the CDF is to be evaluated in the first column of table. These must be in ascending order.
- 1—IMSL_CONT_TABLE fills the last three columns of table. The supplied function *f* is not used and may be a dummy function; instead, the cumulative distribution function is specified in the first two columns of table. The abscissas (in the first column) must be in ascending order and the function must be strictly monotonically increasing.

ndata

Number of points at which the CDF is evaluated for interpolation. *ndata* must be greater than or equal to 4.

table

ndata by 5 table to be used for interpolation of the cumulative distribution function. The first column of table contains abscissas of the cumulative distribution function in ascending order, the second column contains the values of the CDF (which must be strictly increasing), and the remaining columns contain values used in interpolation. The first row of table corresponds to the left limit of the support of the distribution and the last row corresponds to the right limit of the support; that is, *table* (0, 1) = 0.0 and *table*(*ndata* – 1, 1) = 1.0.

Keywords

DOUBLE

If present and nonzero, double precision is used.

Discussion

IMSL_CONT_TABLE sets up a table that “[IMSL_RANDOM_GEN_CONT](#)” on page 1120 can use to generate pseudorandom deviates from a continuous distribution. The distribution is specified by its cumulative distribution function, which can be supplied either in tabular form in *table* or by a function *f*. See the documentation for the routine RAND_GEN_CONT for a description of the method.

Example

For an example of using IMSL_CONT_TABLE see the example for [IMSL_RANDOM_GEN_CONT](#).

Version History

6.4	Introduced

IMSL_RAND_GEN_CONT

The IMSL_RAND_GEN_CONT function generates pseudorandom numbers from a general continuous distribution.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_RAND_GEN_CONT(*n*, *table* [, /DOUBLE])

Return Value

An array of length *n* containing the random deviates.

Arguments

n

Number of random numbers to generate.

table

A two-dimensional array setup using IMSL_CONT_TABLE to be used for interpolation of the cumulative distribution function. The first column of *table* contains abscissas of the cumulative distribution function in ascending order, the second column contains the values of the CDF (which must be strictly increasing beginning with 0.0 and ending at 1.0) and the remaining columns contain values used in interpolation.

Keywords

DOUBLE

If present and nonzero, double precision is used.

Discussion

Routine `IMSL_RANDOM_GEN_CONT` generates pseudorandom numbers from a continuous distribution using the inverse CDF technique, by interpolation of points of the distribution function given in table, which is set up by “`IMSL_CONT_TABLE`” on page 1118. A strictly monotone increasing distribution function is assumed. The interpolation is by an algorithm attributable to Akima (1970), using piecewise cubics. The use of this technique for generation of random numbers is due to Guerra, Tapia, and Thompson (1976), who give a description of the algorithm and accuracy comparisons between this method and linear interpolation. The relative errors using the Akima interpolation are generally considered very good.

Example

In this example, `IMSL_RANDOM_GEN_CONT` is used to set up a table for generation of beta pseudorandom deviates. The CDF for this distribution is computed by the routine `IMSL_BETACDF`. The table contains 100 points at which the CDF is evaluated and that are used for interpolation. Notice that two warnings are issued during the computations for this example.

```
.RUN
FUNCTION cdf, x
    return, IMSL_BETACDF(x, 3., 2.)
END

iopt = 0
ndata = 100;
table = FLTARR(100, 5)
x = 0.0;
table(*,0) = FINDGEN(100)/100.
IMSL_CONT_TABLE, 'cdf', iopt, ndata, table
IMSL_RANDOMOPT, Set = 123457

r = IMSL_RANDOM_GEN_CONT(5, table)
PM, r

0.92079391
0.46412855
0.76678398
0.65357975
0.81706959
```

Version History

6.4	Introduced

IMSL_DISCR_TABLE

The IMSL_DISCR_TABLE function sets up table to generate pseudorandom numbers from a general discrete distribution.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_DISCR_TABLE(prf, del, nndx, imin, nmass [, CUM_PROBS=array]  
[, /DOUBLE])
```

Return Value

Array, *cumpr*, of length *nmass* + *nndx* containing in the first *nmass* positions, the cumulative probabilities and in some of the remaining positions, indexes to speed access to the probabilities.

Arguments

del

Maximum absolute error allowed in computing the cumulative probability. Probabilities smaller than *del* are ignored; hence, *del* should be a small positive number. If *del* is too small, however, *cumpr* (*nmass* – 1) must be exactly 1.0 since that value is compared to $1.0 - del$.

imin

Scalar containing the smallest value the random deviate can assume. By default, *prf* is evaluated at *imin*. If this value is less than *del*, *imin* is incremented by 1 and again *prf* is evaluated at *imin*. This process is continued until $prf(imin) \geq del$. *imin* is output as this value and *result*(0) is output as *prf(imin)*.

nmass

Scalar containing the number of mass points in the distribution. Input, if keyword *Cum_probs* is used; otherwise, output. By default, *nmass* is the smallest integer such

that $prf(imin + nmass - 1) > 1.0 - del$. $nmass$ does include the points $imin_{in} + j$ for which $prf(imin_{in} + j) < del$, for $j = 0, 1, \dots, imin_{out} - imin_{in}$, where $imin_{in}$ denotes the input value of $imin$ and $imin_{out}$ denotes its output value.

nndx

The number of elements of *cumpr* available to be used as indexes. *nndx* must be greater than or equal to 1. In general, the larger *nndx* is, to within sixty or seventy percent of *nmass*, the more efficient the generation of random numbers using IMSL_RANDOM_GEN_DISCR will be.

prf

A scalar string specifying a user-supplied function to compute the probability associated with each mass point of the distribution. The argument to the function is the point at which the probability function is to be evaluated. The argument to the function can range from *imin* to the value at which the cumulative probability is greater than or equal to $1.0 - del$.

Keywords

CUM_PROBS

One dimensional array of length *nmass* containing the cumulative probabilities to be used in computing the index portion of the result. If the keyword *Cum_Probs* is used, *prf* is not used and may be a dummy function.

DOUBLE

If present and nonzero, double precision is used.

Discussion

IMSL_DISCR_TABLE sets up a table that “IMSL_RANDOM_GEN_CONT” on page 1120 uses to generate pseudorandom deviates from a discrete distribution. The distribution can be specified either by its probability function *prf* or by a vector of values of the cumulative probability function. Note that *prf* is not the cumulative probability distribution function. If the cumulative probabilities are already available in *Cum_Probs*, the only reason to call IMSL_DISCR_TABLE is to form an index vector in the upper portion of the result so as to speed up the generation of random deviates by the routine RAND_GEN_CONT.

Examples

Example 1

In this example, `IMSL_DISCR_TABLE` is used to set up a table to generate pseudorandom variates from the discrete distribution:

$$Pr(X = 1) = 0.05$$

$$Pr(X = 2) = 0.45$$

$$Pr(X = 3) = 0.31$$

$$Pr(X = 4) = 0.04$$

$$Pr(X = 5) = 0.15$$

In this example, we input the cumulative probabilities directly using keyword `Cum_Probs` and request 3 indexes to be computed (`nndx = 4`). Since the number of mass points is so small, the indexes would not have much effect on the speed of the generation of the random variates.

```
.RUN
FUNCTION prf, x
    RETURN, 0
END

cum_probs = [.05, .5, .81, .85, 1]
cumpr = IMSL_DISCR_TABLE('PRF', 0.00001, 4, 1, 5, $
    CUM_PROBS = cum_probs)
PM, cumpr

0.0500000
0.500000
0.810000
0.850000
1.00000
3.00000
1.00000
2.00000
5.00000
```

Example 2

`IMSL_DISCR_TABLE` is sets up a table to generate binomial variates with parameters 20 and 0.5. `IMSL_BINOMIALPDF` is used to compute the probabilities.

```
.RUN
FUNCTION prf, ix
```

```
        RETURN,  IMSL_BINOMIALPDF(ix, 20, 0.5)
    END

    cumpr = IMSL_DISCR_TABLE('PRF', 0.00001, 12, 0, 21)
    PM, cumpr

    1.90735e-05
    0.000200272
    0.00128746
    0.00590802
    0.0206938
    0.0576583
    0.131587
    0.251722
    0.411901
    0.588099
    0.748278
    0.868413
    0.942342
    0.979306
    0.994092
    0.998713
    0.999800
    0.999981
    1.00000
    11.0000
    1.00000
    7.00000
    8.00000
    9.00000
    9.00000
    10.0000
    11.0000
    11.0000
    12.0000
    13.0000
    19.0000
```

Version History

6.4	Introduced

IMSL_RAND_GEN_DISCR

The `IMSL_RAND_GEN_DISCR` function generates pseudorandom numbers from a general discrete distribution using an alias method or optionally a table lookup method.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_RAND_GEN_DISCR(n, imin, nmass, probs [, /DOUBLE]  
[, /TABLE] )
```

Return Value

Integer array of length *n* containing the random discrete deviates.

Arguments

imin

Smallest value the random deviate can assume. This is the value corresponding to the probability in `probs(0)`.

nmass

Number of mass points in the discrete distribution.

n

Number of random numbers to generate.

probs

Array of length *nmass* containing probabilities associated with the individual mass points. The elements of *probs* must be nonnegative and must sum to 1.0.

If the keyword *Table* is used, then *probs* is a vector of length at least *nmass* + 1 containing in the first *nmass* positions the cumulative probabilities and, possibly,

indexes to speed access to the probabilities. “[IMSL_DISCR_TABLE](#)” on page 1123 can be used to initialize *probs* properly. If no elements of *probs* are used as indexes, *probs* (*nmass*) is 0.0 on input. The value in *probs*(0) is the probability of *imin*. The value in *probs* (*nmass* – 1) must be exactly 1.0 (since this is the CDF at the upper range of the distribution.)

Keywords

DOUBLE

If present and nonzero, double precision is used.

TABLE

If present and nonzero, generate pseudorandom numbers from a general discrete distribution using a table lookup method. If this keyword is used, then *probs* is a vector of length at least *nmass* + 1 containing in the first *nmass* positions the cumulative probabilities and, possibly, indexes to speed access to the probabilities. “[IMSL_DISCR_TABLE](#)” on page 1123 can be used to initialize *probs* properly.

Discussion

IMSL_RANDOM_GEN_DISCR generates pseudorandom numbers from a discrete distribution with probability function given in the vector *probs*; that is:

$$\Pr(X = i) = p_j$$

$$\text{for } i = i_{\min}, i_{\min} + 1, \dots, i_{\min} + n_m - 1$$

where:

$$j = i - i_{\min} + 1, p_j = \text{probs}(j), i_{\min} = \text{imin}, \text{ and } n_m = \text{nmass}$$

The algorithm is the alias method, due to Walker (1974), with modifications suggested by Kronmal and Peterson (1979).

If the keyword *Table* is used, IMSL_RANDOM_GEN_DISCR generates pseudorandom deviates from a discrete distribution, using the table *probs*, which contains the cumulative probabilities of the distribution and, possibly, indexes to speed the search of the table. “[IMSL_DISCR_TABLE](#)” on page 1123 can be used to set up the table *probs*. IMSL_RANDOM_GEN_DISCR uses the inverse CDF method to generate the variates.

Examples

Example 1

In this example, `IMSL_RAND_GEN_DISCR` is used to generate five pseudorandom variates from the discrete distribution:

$$Pr(X = 1) = 0.05$$

$$Pr(X = 2) = 0.45$$

$$Pr(X = 3) = 0.31$$

$$Pr(X = 4) = 0.04$$

$$Pr(X = 5) = 0.15$$

```
probs = [0.05, 0.45, 0.31, 0.04, 0.15]
n = 5
imin = 1
nmass = 5
IMSL_RANDOMOPT, Set_seed = 123457
r = IMSL_RAND_GEN_DISCR(n, imin, nmass, probs)
PM, r

3
2
2
3
5
```

Example 2

In this example, the “[IMSL_DISCR_TABLE](#)” on page 1123 is used to set up a table and then `IMSL_RAND_GEN_DISCR` is used to generate five pseudorandom variates from the binomial distribution with parameters 20 and 0.5.

```
.RUN
FUNCTION prf, ix
    RETURN, IMSL_BINOMIALPDF(ix, 20, .5)
END

imin = 0
nmass = 21
IMSL_RANDOMOPT, Set_seed = 123457
cumpr = IMSL_DISCR_TABLE('prf', 0.00001, 12, imin, nmass)
r = IMSL_RAND_GEN_DISCR(n, imin, nmass, cumpr, /TABLE)
PM, r
```

14
9
12
10
12

Version History

6.4	Introduced

IMSL_RANDOM_ARMA

The `IMSL_RANDOM_ARMA` function generates a time series from a specific `IMSL_ARMA` model.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_RANDOM_ARMA(n, nparams [, /ACCEPT_REJECT]
    [, AR_LAGS=array] [, CONST=value] [, /DOUBLE] [, INPUT_NOISE=array]
    [, MA_LAGS=array] [, OUTPUT_NOISE=variable] [, /VAR_NOISE]
    [, W_INIT=array])
```

```
Result = IMSL_RANDOM_ARMA(n, nparams, ar [, /ACCEPT_REJECT]
    [, AR_LAGS=array] [, CONST=value] [, /DOUBLE] [, INPUT_NOISE=array]
    [, MA_LAGS=array] [, OUTPUT_NOISE=variable] [, /VAR_NOISE]
    [, W_INIT=array])
```

```
Result = IMSL_RANDOM_ARMA(n, nparams, ma [, /ACCEPT_REJECT]
    [, AR_LAGS=array] [, CONST=value] [, /DOUBLE] [, INPUT_NOISE=array]
    [, MA_LAGS=array] [, OUTPUT_NOISE=variable] [, /VAR_NOISE]
    [, W_INIT=array])
```

```
Result = IMSL_RANDOM_ARMA(n, nparams, ar, ma [, /ACCEPT_REJECT]
    [, AR_LAGS=array] [, CONST=value] [, /DOUBLE] [, INPUT_NOISE=array]
    [, MA_LAGS=array] [, OUTPUT_NOISE=variable] [, /VAR_NOISE]
    [, W_INIT=array])
```

Return Value

One-dimensional array of length n containing the generated time series.

Arguments

n

Number of observations to be generated. Parameter n must be greater than or equal to one.

nparams

One-dimensional array containing the parameters p and q consecutively. $nparams(0) = p$, where p is the number of autoregressive parameters. Parameter p must be greater than or equal to zero. $nparams(1) = q$, where q is the number of moving average parameters. Parameter q must be greater than or equal to zero.

ar

One-dimensional array of length p containing the autoregressive parameters.

ma

One-dimensional array of length q containing the moving average parameters.

Keywords**ACCEPT_REJECT**

If present and nonzero, the random noises will be generated from a normal distribution using an acceptance/rejection method. If keyword `Accept_Reject` is not used, the random noises will be generated using an inverse normal CDF method. This argument will be ignored if keyword `Input_Noise` is used.

AR_LAGS

One-dimensional array of length p containing the order of the nonzero autoregressive parameters. Default: $Ar\_Lags = [1, 2, \dots, p]$

CONST

Overall constant. See the *Discussion* section. Default: $Const = 0$

DOUBLE

If present and nonzero, double precision is used.

INPUT_NOISE

One-dimensional array of length $n + \max(Ar\_Lags(i))$ containing the random noises. Keywords `Input_Noise` and `Var_Noise` can not be used together. Keywords `Input_Noise` and `Output_Noise` cannot be used together.

MA_LAGS

One-dimensional array of length q containing the order of the nonzero moving average parameters. Default: $Ma\_Lags = [1, 2, \dots, q]$

OUTPUT_NOISE

Named variable into which a one-dimensional array of length $n + \max(Ma\_Lags(i))$ containing the random noises is stored.

VAR_NOISE

If present (and *Input_Noise* is *not* used), noise a_t is generated from a normal distribution with mean 0 and variance *Var_Noise*. *Var_Noise* and *Input_Noise* cannot be used together. Default: $Var\_Noise = 1.0$

W_INIT

One-dimensional array of length $\max(Ar\_Lags(i))$ containing the initial values of the time series. Default: $W\_Init(*) = Const/(1 - ar(0) - ar(1) - \dots - ar(p - 1))$

Discussion

The IMSL_RANDOM_ARMA function simulates an $IMSL\_ARMA(p, q)$ process, $\{W_t\}$, for $t = 1, 2, \dots, n$. The model is:

$$\phi(B)W_t = \theta_0 + \theta(B)A_t \quad t \in Z$$

$$\phi(B) = 1 - \phi_1 B - \phi_2 B^2 - \dots - \phi_p B^p$$

$$\theta(B) = 1 - \theta_1 B - \theta_2 B^2 - \dots - \theta_q B^q$$

Let μ be the mean of the time series $\{W_t\}$. The overall constant θ_0 (*Const*) is:

$$\theta_0 = \begin{cases} \mu & p = 0 \\ \mu \left(1 - \sum_{i=1}^p \phi_i \right) & p > 0 \end{cases}$$

Time series whose innovations have a nonnormal distribution may be simulated by providing the appropriate innovations in *Input_Noise* and start values in *W_Init*.

The time series is generated according to the following model:

$$X(i) = \text{Const} + \text{ar}(0) * X(i - \text{Ar\_Lags}(0)) + \dots + \text{ar}(p - 1) * X(i - \text{Ar\_Lags}(p - 1)) +$$

$$A(I) - \text{ma}(0) * A(i - \text{Ma\_Lags}(0)) - \dots - \text{ma}(q - 1) * A(i - \text{Ma\_Lags}(q - 1))$$

where the constant is related to the mean of the series:

$$\bar{W}$$

as follows:

$$\text{Const} = \bar{W} \cdot (1 - \text{ar}(0) - \dots + (-\text{ar}(q - 1)))$$

and where:

$$X(t) = W(t), t = 0, 1, \dots, n - 1$$

and:

$$W(t) = W\_Init(t + p), t = -p, -p + 1, \dots, -2, -1$$

and A is either *Input_Noise* (if *Input_Noise* is used) or *Output_Noise* (otherwise).

Examples

Example 1

In this example, `IMSL_RANDOM_ARMA` is used to generate a time series of length five, using an `IMSL_ARMA` model with three autoregressive parameters and two moving average parameters. The start values are 0.1000, 0.0500, and 0.0375.

```
IMSL_RANDOMOPT, SET = 123457
n = 5
nparams = [3, 2]
ar = [0.5, 0.25, 0.125]
ma = [-0.5, -0.25]
r = IMSL_RANDOM_ARMA(n, nparams, ar, ma)
PM, r, FORMAT = '(5F10.3)', $
TITLE = '                                IMSL_ARMA random deviates'

                                IMSL_ARMA random deviates
0.637      0.317      -0.366      -2.122      -1.407
```

Example 2

In this example, a time series of length 5 is generated using an `IMSL_ARMA` model with 4 autoregressive parameters and 2 moving average parameters. The start values are 0.1, 0.05 and 0.0375.

```
IMSL_RANDOMOPT, SET = 123457
```

```

n = 5
nparams = [3, 2]
ar = [0.5, 0.25, 0.125]
ma = [-0.5, -0.25]
wi = [0.1, 0.05, 0.0375]
theta0 = 1
avar = 0.1
r = IMSL_RANDOM_ARMA(n, nparams, ar, ma, /ACCEPT_REJECT, $
  W_INIT = wi, CONST = theta0, VAR_NOISE = avar)
PM, r, FORMAT = '(5F10.3)', $
  TITLE = '                      IMSL_ARMA random deviates:'

                      IMSL_ARMA random deviates:
1.467      1.788      2.459      3.330      3.941

```

Errors

Warning Errors

STAT_RNARM_NEG_VAR—VAR(a) = “Var_Noise” = #, VAR(a) must be greater than 0. The absolute value of # is used for VAR(a).

Version History

6.4	Introduced

IMSL_FAURE_INIT

The IMSL_FAURE_INIT function initializes the structure used for computing a shuffled Faure sequence.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_FAURE_INIT(*ndim* [, *BASE*=value] [, *SKIP*=value])

Return Value

A structure that contains information about the sequence.

Arguments

ndim

The dimension of the hyper-rectangle.

Keywords

BASE

The base of the Faure sequence. Default: The smallest prime greater than or equal to *ndim*.

SKIP

The number of points to be skipped at the beginning of the Faure sequence. Default:

$$\lfloor base^{m/(2-1)} \rfloor$$

where:

$$m = \lfloor \log B / \log base \rfloor$$

and B is the largest representable integer.

Discussion

Discrepancy measures the deviation from uniformity of a point set. The discrepancy of the point set:

$$x_1, \dots, x_n \in [0, 1]^d, d \geq 1,$$

is:

$$D_n^{(d)} = \sup_E \left| \frac{A(E; n)}{n} - \lambda E \right|$$

where the supremum is over all subsets of $[0, 1]^d$ of the form:

$$E = [0, t_1) \times \dots \times [0, t_d), 0 \leq t_j \leq 1, 1 \leq j \leq d,$$

λ is the Lebesgue measure, and:

$$A(E; n)$$

is the number of the x_j contained in E .

The sequence $x_1, x_2, \dots$, of points $[0, 1]^d$ is a low-discrepancy sequence if there exists a constant $c(d)$, depending only on d , such that:

$$D_n^{(d)} \leq c(d) \frac{(\log n)^d}{n}$$

for all $n > 1$.

Generalized Faure sequences can be defined for any prime base $b \geq d$. The lowest bound for the discrepancy is obtained for the smallest prime $b \geq d$, so the keyword *Base* defaults to the smallest prime greater than or equal to the dimension. The generalized Faure sequence $x_1, x_2, \dots$, is computed as follows:

Write the positive integer n in its b -ary expansion:

$$n = \sum_{i=0}^{\infty} a_i(n) b^i$$

where $a_i(n)$ are integers:

$$0 \leq a_i(n) < b$$

The j -th coordinate of x_n is:

The generator matrix for the series:

$$x_n^{(j)} = \sum_{k=0}^{\infty} \sum_{d=0}^{\infty} c_{kd}^{(j)} a_d(n) b^{-k-1}, 1 \leq j \leq d$$

$$c_{kd}^{(j)}$$

is defined to be:

$$c_{kd}^{(j)} = j^{d-k} c_{kd}$$

and:

$$c_{kd}$$

is an element of the Pascal matrix:

$$c_{kd} = \begin{cases} \frac{d!}{c!(d-c)!} & k \leq d \\ 0 & k > d \end{cases}$$

It is faster to compute a shuffled Faure sequence than to compute the Faure sequence itself. It can be shown that this shuffling preserves the low-discrepancy property.

The shuffling used is the b -ary Gray code. The function $G(n)$ maps the positive integer n into the integer given by its b -ary expansion.

The sequence computed by this function is $x(G(n))$, where x is the generalized Faure sequence.

Example

In this example, five points in the Faure sequence are computed. The points are in the three-dimensional unit cube.

Note that `IMSL_FAURE_INIT` is used to create a structure that holds the state of the sequence. Each call to `IMSL_FAURE_NEXT_PT` returns the next point in the sequence and updates the state structure.

```
state = IMSL_FAURE_INIT(3)
p = IMSL_FAURE_NEXT_PT(5, state)
PM, p

0.333689    0.492659    0.0640654
0.667022    0.825992    0.397399
0.778133    0.270436    0.175177
0.111467    0.603770    0.508510
0.444800    0.937103    0.841843
```

Version History

6.4	Introduced

IMSL_FAURE_NEXT_PT

The IMSL_FAURE_NEXT_PT function computes a shuffled Faure sequence.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_FAURE_NEXT_PT(*npts*, *state* [, /DOUBLE] [, SKIP=*value*])

Return Value

An array of size *npts* by *state.dim* containing the *npts* next points in the shuffled Faure sequence.

Arguments

npts

The number of points to generate in the hyper-rectangle.

state

State structure created by a call to IMSL_FAURE_INIT.

Keywords

DOUBLE

If present and nonzero, double precision is used.

SKIP

The current point in the sequence. The sequence can be restarted by initializing a new sequence using this value for *Skip*, and using the same dimension for *ndim*.

Discussion

Discrepancy measures the deviation from uniformity of a point set.

The discrepancy of the point set:

$$x_1, \dots, x_n \in [0, 1]^d, d \geq 1,$$

is:

$$D_n^{(d)} = \sup_E \left| \frac{A(E;n)}{n} - \lambda E \right|$$

where the supremum is over all subsets of $[0, 1]^d$ of the form:

$$E = [0, t_1) \times \dots \times [0, t_d), 0 \leq t_j \leq 1, 1 \leq j \leq d,$$

λ is the Lebesgue measure, and:

$$A(E;n)$$

is the number of the x_j contained in E .

The sequence $x_1, x_2, \dots$ of points $[0,1]^d$ is a low-discrepancy sequence if there exists a constant $c(d)$, depending only on d , such that:

$$D_n^{(d)} \leq c(d) \frac{(\log n)^d}{n}$$

for all $n > 1$.

Generalized Faure sequences can be defined for any prime base $b \geq d$. The lowest bound for the discrepancy is obtained for the smallest prime $b \geq d$, so the keyword *Base* defaults to the smallest prime greater than or equal to the dimension. The generalized Faure sequence $x_1, x_2, \dots$, is computed as follows:

Write the positive integer n in its b -ary expansion:

$$n = \sum_{i=0}^{\infty} a_i(n) b^i$$

where $a_i(n)$ are integers:

$$0 \leq a_i(n) < b$$

The j -th coordinate of x_n is:

$$x_n^{(j)} = \sum_{k=0}^{\infty} \sum_{d=0}^{\infty} c_{kd}^{(j)} a_d(n) b^{-k-1}, 1 \leq j \leq d$$

The generator matrix for the series:

$$c_{kd}^{(j)}$$

is defined to be:

$$c_{kd}^{(j)} = j^{d-k} c_{kd}$$

and:

$$c_{kd}$$

is an element of the Pascal matrix:

$$c_{kd} = \begin{cases} \frac{d!}{k!(d-k)!} & k \leq d \\ 0 & k > d \end{cases}$$

It is faster to compute a shuffled Faure sequence than to compute the Faure sequence itself. It can be shown that this shuffling preserves the low-discrepancy property.

The shuffling used is the *b*-ary Gray code. The function $G(n)$ maps the positive integer n into the integer given by its *b*-ary expansion.

The sequence computed by this function is $x(G(n))$, where x is the generalized Faure sequence.

Example

In this example, five points in the Faure sequence are computed. The points are in the three-dimensional unit cube.

Note that `IMSL_FAURE_INIT` is used to create a structure that holds the state of the sequence. Each call to `IMSL_FAURE_NEXT_PT` returns the next point in the sequence and updates the state structure.

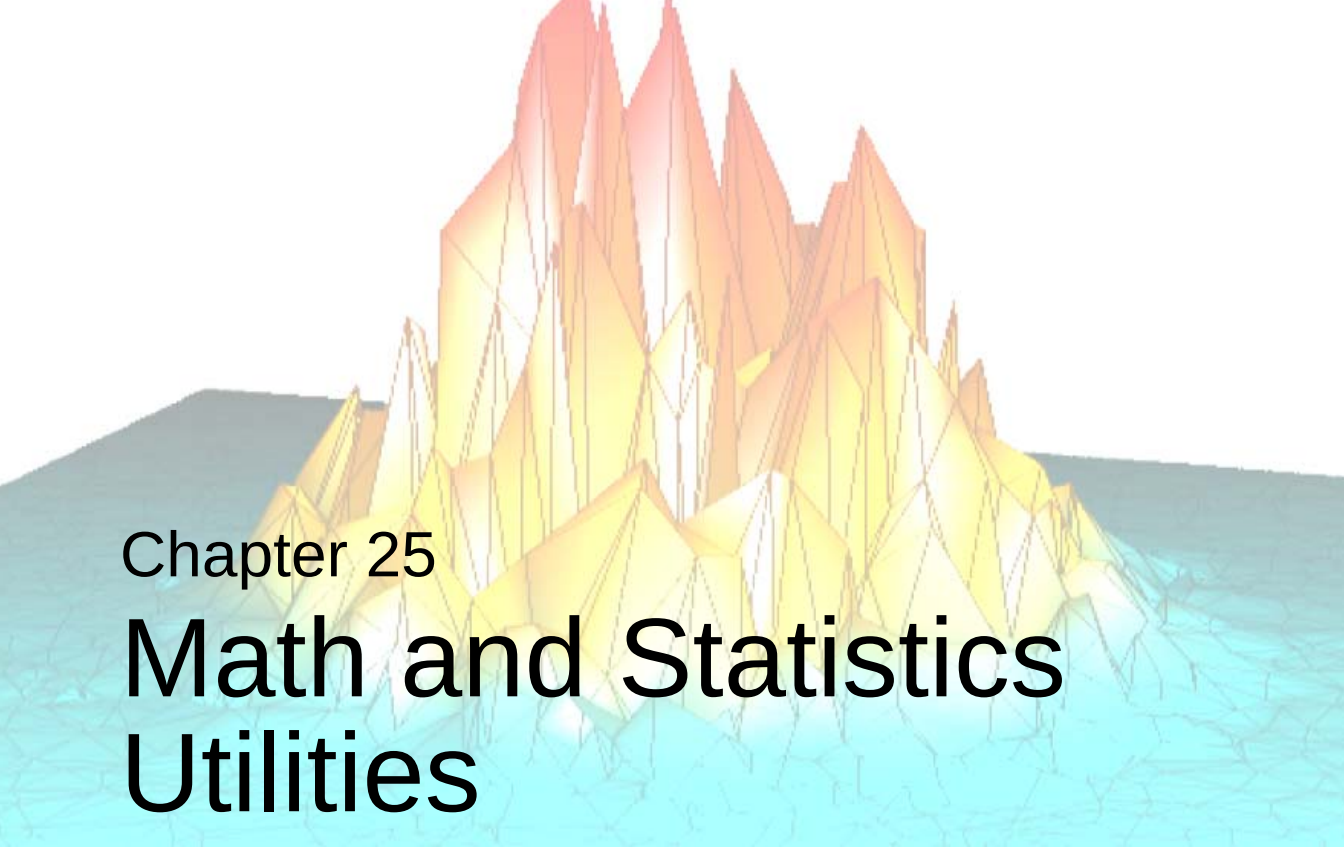
```
state = IMSL_FAURE_INIT(3)
p = IMSL_FAURE_NEXT_PT(5, state)
PM, p

0.333689      0.492659      0.0640654
```

0.667022	0.825992	0.397399
0.778133	0.270436	0.175177
0.111467	0.603770	0.508510
0.444800	0.937103	0.841843

Version History

6.4	Introduced



Chapter 25

Math and Statistics Utilities

This section contains the following topics:

[Overview: Math and Statistics Utilities . . 1146](#) [Math and Statistics Utilities Routines . . 1147](#)

Overview: Math and Statistics Utilities

This chapter describes general utility routines related to the IMSL library's mathematics and statistics routines. See [“Math and Statistics Utilities Routines”](#) on page 1147 for a list of the included routines.

Math and Statistics Utilities Routines

Dates

[IMSL_DAYSTODATE](#)—Days since epoch to date.

[IMSL_DATETODAYS](#)—Date to days since epoch.

Constants and Data Sets

[IMSL_CONSTANT](#)—Natural and mathematical constants.

[IMSL_MACHINE](#)—Machine constants.

[IMSL_STATDATA](#)—Commonly analyzed data sets.

Binomial Coefficient

[IMSL_BINOMIALCOEF](#)—Evaluates the binomial coefficient.

Geometry

[IMSL_NORM](#)—Vector norms.

Matrix Norm

[IMSL_MATRIX_NORM](#)—Real coordinate matrix.

Matrix Entry and Display

[PM](#)—Formatted output of arrays using the standard linear algebraic convention: “row” refers to the first index of the array and “column” refers to the second.

[RM](#)—Formatted input of arrays using the standard linear algebraic convention: “row” refers to the first index of the array and “column” refers to the second.

IMSL_DAYSTODATE

The IMSL_DAYSTODATE procedure gives the date corresponding to the number of days since January 1, 1900.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

IMSL_DAYSTODATE, *days*, *day*[, *month*[, *year*]]

Arguments

days

Number of days since January 1, 1900.

day

On return, this named variable is assigned the day of the date specified by *days*.

month

If present, on return, this named variable is assigned the month of the date specified by *days*.

year

If present, on return, this named variable is assigned the year of the date specified by *days*. The year 1950 corresponds to the year 1950 A.D., and the year 50 corresponds to year 50 A.D.

Discussion

The IMSL_DAYSTODATE procedure computes the date corresponding to the number of days since January 1, 1900. For a negative input value of *days*, the date computed is prior to January 1, 1900. This procedure is the inverse of the IDL Advanced Math and Stats IMSL_DATETODAYS function.

The Gregorian calendar's first day after October 4, 1502, which became October 15, 1582. Prior to that, the Julian calendar was in use.

Example

The following example uses `IMSL_DAYSTODATE` to compute the date for the 100th day of 1986. This is accomplished by first using `IMSL_DATETODAYS` to get the “day number” for December 31, 1985.

```
d0 = IMSL_DATETODAYS(31, 12, 1985)
IMSL_DAYSTODATE, d0 + 100, d, m, y
PM, d, m, y, TITLE = 'Day 100 of 1986 is (day-month-year)', $
      FORMAT = '(20x, i3, i4, i7)'
```

Day 100 of 1986 is (day-month-year)
10 4 1986

Version History

6.4	Introduced

IMSL_DATETODAYS

The IMSL_DATETODAYS function computes the number of days from January 1, 1900, to the given date.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

$$Result = \text{IMSL\_DATETODAYS}([day[, month[, year]]])$$

Return Value

Number of days from January 1, 1900, to the given date. If negative, it indicates the number of days prior to January 1, 1900.

Arguments

day

Day of the input date.

month

Month of the input date.

year

Year of the input date. The year 1950 corresponds to the year 1950 A.D., and the year 50 corresponds to year 50 A.D.

Discussion

The IMSL_DATETODAYS function returns the number of days from January 1, 1900, to the given date and returns negative values for days prior to January 1, 1900. A negative *year* can be used to specify B.C. Input dates in year 0 and for October 5, 1582, through October 14, 1582, inclusive, do not exist; consequently, in these cases, IMSL_DATETODAYS issues an error.

The Gregorian calendar starts the first day after October 4, 1582, which became October 15, 1582. Prior to that, the Julian calendar was in use.

Example

The following example uses `IMSL_DATETODAYS` to compute the number of days from January 15, 1986, to February 28, 1986.

```
d0 = IMSL_DATETODAYS(15, 1, 1986)
d1 = IMSL_DATETODAYS(28, 2, 1986)
PM, d1 - d0, TITLE = 'Number of days from 1/15/86 to 2/28/86'

Number of days from 1/15/86 to 2/28/86
44
```

Version History

6.4	Introduced

IMSL_CONSTANT

The `IMSL_CONSTANT` function returns the value of various mathematical and physical constants.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = `IMSL_CONSTANT`(*name* [, *units*] [, `/DOUBLE`])

Return Value

By default, returns the desired constant. If no value can be computed, NaN (Not a Number) is returned.

Arguments

name

Scalar string specifying the name of the desired constant. The case of the characters is not relevant when specifying *name*, i.e., character strings “PI”, “Pi”, “pI”, and “pi” are equivalent. Spaces and underscores are allowed and ignored.

units

Scalar string specifying the units of the desired constant. If empty, then Systeme International d’Unites (SI) units are assumed. The case of the characters is not relevant when specifying *units*, i.e., character strings “METER”, “Meter”, and “meter” are equivalent. Parameter *units* has the form “U1*U2*...*Um/V1/.../Vn,” where U_i and V_i are the names of basic units or the names of basic units raised to a power. Basic units must be separated by * or /. Powers are indicated by ^, as in “m^2” for m². Examples are “METER*KILOGRAM/SECOND”, “M*KG/S”, “METER”, or “M/KG^2”.

Keywords

DOUBLE

If present and nonzero, double precision is used.

Discussion

The names allowed are listed in [Table 25-1](#). Values marked with (mp) are exact (to machine precision). The references in the right-hand column are indicated by code numbers: (1) for Cohen and Taylor (1986), (2) for Liepman (1964), and (3) for precomputed mathematical constants. The supported units are listed in [Table 25-2](#).

Name	Description	Value	Ref.
amu	atomic mass unit	$1.6605655 \times 10^{-27}$ kg	1
ATM	standard atm. pressure	1.01325×10^5 N/m ² (mp)	2
AU	astronomical unit	1.496×10^{11} m	
Avogadro	Avogadro's number, N	6.022045×10^{23} 1/mole	1
Boltzman	Boltzman's constant, k	1.380662×10^{-23} J / K	1
C	speed of light, c	2.997924580×10^8 m/s	1
Catalan	Catalan's constant	0.915965... (mp)	3
E	base of natural logs, e	2.718... (mp)	3
ElectronCharge	electron charge, e	$1.6021892 \times 10^{-19}$ C	1
ElectronMass	electron mass, m_e	9.109534×10^{-31} kg	1
ElectronVolt	electron volt, ev	$1.6021892 \times 10^{-19}$ J	1
Euler	Euler's constant, γ	0.577... (mp)	3
Faraday	Faraday constant, F	9.648456×10^4 C/mole	1
FineStructure	fine structure, α	7.2973506×10^{-3}	1
Gamma	Euler's constant, γ	0.577... (mp)	3
Gas	gas constant, R_0	8.31441 J/mole/K	1

Table 25-1: Constant Names

Name	Description	Value	Ref.
Gravity	gravitational constant, G	$6.6720 \times 10^{-11} \text{ N m}^2 / \text{kg}^2$	1
Hbar	Planck's constant / 2π	$1.0545887 \times 10^{-34} \text{ J s}$	1
PerfectGasVolume	std. vol. ideal gas	$2.241383 \times 10^{-2} \text{ m}^3 / \text{mole}$	1
Pi	Pi, π	3.141... (mp)	3
Planck	Planck's constant, h	$6.626176 \times 10^{-34} \text{ J s}$	1
ProtonMass	proton mass, M_p	$1.6726485 \times 10^{-27} \text{ kg}$	1
Rydberg	Rydberg's constant, R_{infinity}	$1.097373177 \times 10^7 / \text{m}$	1
Speedlight	speed of light, c	$2.997924580 \times 10^8 \text{ m/s}$	1
StandardGravity	standard g	9.80665 m/s^2 (mp)	2
StandardPressure	standard atm. pressure	$1.01325 \times 10^5 \text{ N/m}^2$ (mp)	2
StefanBoltzman	Stefan-Boltzman, σ	$5.67032 \times 10^{-8} \text{ W/K}^4 / \text{m}^2$	1
WaterTriple	triple point of water	$2.7316 \times 10^2 \text{ K}$	2

Table 25-1: Constant Names (Continued)

The units allowed are as follows:

Unit	Description
time	day, hour = hr, min = minute, s = sec = second, year
frequency	Hertz = Hz
mass	AMU, g = gram, lb = pound, ounce = oz, slug
distance	Angstrom, AU, feet = foot, in = inch, m = meter = metre, micron, mile, mill, parsec, yard

Table 25-2: Supported Units

Unit	Description
area	acre
volume	l = liter = litre
force	dyne, N = Newton
energy	BTU, Erg, J = Joule
work	W = watt
pressure	ATM = atmosphere, bar
temperature	degC = Celsius, degF = Fahrenheit, degK = Kelvin
viscosity	poise, stoke
charge	Abcoulomb, C = Coulomb, statcoulomb
current	A = ampere, abampere, statampere
voltage	Abvolt, V = volt
magnetic induction	T = Tesla, Wb = Weber
other units	l, farad, mole, Gauss, Henry, Maxwell, Ohm

Table 25-2: Supported Units (Continued)

The metric prefixes listed in [Table 25-3](#) can be used with the previous units. The one- or two-letter prefixes can only be used with one-letter unit abbreviations.

Prefix	Definition	Value
a	atto	10^{-18}
f	femto	10^{-15}
p	pico	10^{-12}
n	nano	10^{-9}
u	micro	10^{-6}
m	milli	10^{-3}
c	centi	10^{-2}
d	deci	10^{-1}
dk	deca	10^2
k	kilo	10^3
	myria	10^4
	mega	10^6
g	giga	10^9
t	tera	10^{12}

Table 25-3: Supported Prefixes

There is no one-letter unit abbreviation for *myria* or *mega* since *m* means *milli*.

Examples

Example 1

In this example, Euler's constant γ is obtained and printed. Euler's constant is defined to be as follows:

$$\gamma = \lim_{n \rightarrow \infty} \left[\sum_{k=1}^{n-1} \frac{1}{k} - \ln n \right]$$


```
PM, IMSL_CONSTANT('gamma')  
  
0.577216
```

Example 2

In this example, the speed of light is obtained using several different units.

```
c1 = IMSL_CONSTANT('SpeedLight', 'meter/second')  
c2 = IMSL_CONSTANT('SpeedLight', 'mile/second')  
c3 = IMSL_CONSTANT('SpeedLight', 'cm/ns')  
PM, 'speed of light = ', c1, c2, c3, $  
    Title = '                meters/second    ' + $  
    'miles/second      cm/ns'  
  
meters/second    miles/second    cm/ns  
speed of light =  2.99792e+008    186282.    29.9792
```

Errors

Warning Errors

MATH_MASS_TO_FORCE—Conversion of units-of-mass to units-of-force required for consistency.

Version History

6.4	Introduced

IMSL_MACHINE

The `IMSL_MACHINE` function returns information describing the computer's arithmetic.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_MACHINE( [, /DOUBLE] [, /FLOAT] )
```

Return Value

The information describing the computer's arithmetic is returned in a structure.

Keywords

DOUBLE

If present and nonzero, a structure containing the information describing the single-precision, floating-point arithmetic is returned.

FLOAT

If present and nonzero, a structure containing the information describing the single-precision, floating-point arithmetic is returned.

Discussion

The `IMSL_MACHINE` function returns information describing the computer's arithmetic. This can be used to make programs machine independent. The information returned by `IMSL_MACHINE` is in the form of a structure. A different structure is used for each type: integer, float, and double. Depending on how `IMSL_MACHINE` is called, a different structure is returned.

The default action of `IMSL_MACHINE` is to return the structure `IMACHINE` which contains integer information on the computer's arithmetic. By using either the

keywords *Float* or *Double*, information about the floating- or double-precision arithmetic is returned in structures FMACHINE or DMACHINE.

The contents of the these structures are described below.

Integer Information: IMACHINE

Assume that integers are represented in M -digit, base A form as:

$$\sigma \sum_{k=0}^M x_k A^k$$

where σ is the sign and $0 \leq x_k < A$ for $k = 0, \dots, M$. Then, [Table 25-4](#) describes the tags:

Tag	Definition
BITS_PER_CHAR	C , bits per character
INTEGER_BASE	A , the base
INTEGER_DIGITS	M_s , the number of base- A digits in a <i>short int</i>
MAX_INTEGER	$A^{M_s} - 1$, the largest <i>short int</i>
LONG_DIGITS	M_l , the number of base- A digits in a <i>long int</i>
MAX_LONG	$A^{M_l} - 1$, the largest <i>long int</i>

Table 25-4: Integer Tags

Assume that floating-point numbers are in N -digit, base B form as:

$$\sigma B^E \sum_{k=1}^N x_k B^{-k}$$

where σ is the sign and $0 \leq x_k < B$ for $k = 1, \dots, N$ for and $E_{\min} \leq E \leq E_{\max}$. Then, [Table 25-5](#) describes the tags:

Tag	Definition
FLOAT_BASE	B , the base
FLOAT_DIGITS	N_f , the number of base- B digits in <i>float</i>

Table 25-5: Floating Point Tags

Tag	Definition
FLOAT_MIN_EXP	$E_{\min_f}$, the smallest <i>float</i> exponent
FLOAT_MAX_EXP	$E_{\max_f}$, the largest <i>float</i> exponent
DOUBLE_DIGETS	N_d , the number of base- <i>B</i> digits in <i>double</i>
DOUBLE_MIN_EXP	$E_{\min_d}$, the largest <i>long int</i>
DOUBLE_MAX_EXP	$E_{\max_d}$, the number of base- <i>B</i> digits in <i>double</i>

Table 25-5: Floating Point Tags (Continued)

Floating- and Double-precision Information: FMACHINE and DMACHINE

Information concerning the floating- or double-precision arithmetic of the computer is contained in the structures FMACHINE and DMACHINE. These structures are returned into named variables by calling IMSL_MACHINE with the keywords *Float* for FMACHINE and *Double* for DMACHINE.

Assume that *float* numbers are represented in N_f -digit, base *B* form as:

$$\sigma B^E \sum_{k=1}^N x_k B^{-k}$$

where σ is the sign, $0 \leq x_k < B$ for $k = 1, 2, \dots, N_f$ and

$$E_{\min_f} \leq E \leq E_{\max_f}$$

Note that if we make the assignment `imach = IMSL_MACHINE()`, then $B = \text{imach.FLOAT\_BASE}$, $N_f = \text{imach.FLOAT\_DIGITS}$,

$$E_{\min_f} = \text{imach.FLOAT\_MIN\_EXP}$$

and:

$$E_{\max_f} = \text{imach.FLOAT\_MAX\_EXP}$$

The ANSI/IEEE 754-1985 standard for binary arithmetic uses NaN (Not a Number) as the result of various otherwise illegal operations, such as computing $0/0$. If the assignment `amach = IMSL_MACHINE(/Float)` is made, then on computers that do not support NaN, a value larger than `amach.MAX_POS` is returned in `amach.NAN`. On computers that do not have a special representation for infinity, `amach.POS_INF` contains the same value as `amach.MAX_POS`.

The structure IMACHINE is defined by [Table 25-6](#):

Tag	Definition
MIN_POS	$B^{E_{\min_f}-1}$, the smallest positive number
MAX_POS	$B^{E_{\max_f}}(1 - B^{-N_f})$, the largest number
MIN_REL_SPACE	$B - N_f$, the smallest relative spacing
MAX_REL_SPACE	$B^1 - N_f$, the largest relative spacing
LOG10_BASE	$\log_{10}(B)$
NAN	NaN
POS_INF	positive machine infinity
NEG_INF	negative machine infinity

Table 25-6: Floating or Double Precision Tags

The structure DMACHINE contains machine constants that define the computer's double arithmetic. Note that for *double*, if the assignment `imach = IMSL_MACHINE( )` is made, then:

$$B = \text{imach.FLOAT\_BASE}, N_f = \text{imach.DOUBLE\_DIGITS}$$

$$E_{\min_f} = \text{imach.DOUBLE\_MIN\_EXP}$$

and:

$$E_{\max_f} = \text{imach.DOUBLE\_MAX\_EXP}$$

Missing values in IDL Advanced Math and Stats procedures and functions are often indicated by NaN. There is no missing-value indicator for integers. Users usually have to convert from their missing value indicators to NaN.

Example

In this example, all values returned by `IMSL_MACHINE` are printed on a machine with IEEE (Institute for Electrical and Electronics Engineering) arithmetic.

```
i = IMSL_MACHINE()
f = IMSL_MACHINE(/FLOAT)
d = IMSL_MACHINE(/DOUBLE)
; Call HELP with the keyword STRUCTURE set to view the contents
; of the structures.
```

```
HELP, i, f, d, /STRUCTURE

** Structure IMACHINE, 13 tags, length=52:
  BITS_PER_CHAR LONG           8
  INTEGER_BASE LONG            2
  INTEGER_DIGITS LONG          15
  MAX_INTEGER LONG             32767
  LONG_DIGITS LONG             31
  MAX_LONGLONG 2147483647
  FLOAT_BASE LONG              2
  FLOAT_DIGITS LONG            24
  FLOAT_MIN_EXP LONG           -125
  FLOAT_MAX_EXP LONG           128
  DOUBLE_DIGITS LONG           53
  DOUBLE_MIN_EXP LONG          -1021
  DOUBLE_MAX_EXP LONG           1024
** Structure FMACHINE, 8 tags, length=32:
  MIN_POS FLOAT 1.17549e-38
  MAX_POS FLOAT 3.40282e+38
  MIN_REL_SPACE FLOAT 5.96046e-08
  MAX_REL_SPACE FLOAT 1.19209e-07
  LOG_10 FLOAT 0.301030
  NAN FLOAT NaN
  POS_INF FLOAT Inf
  NEG_INF FLOAT -Inf
** Structure DMACHINE, 8 tags, length=64:
  MIN_POS DOUBLE 2.2250739e-308
  MAX_POS DOUBLE 1.7976931e+308
  MIN_REL_SPACE DOUBLE 1.1102230e-16
  MAX_REL_SPACE DOUBLE 2.2204460e-16
  LOG_10 DOUBLE 0.30102998
  NAN DOUBLE NaN
  POS_INF DOUBLE Infinity
  NEG_INF DOUBLE -Infinity
```

Version History

6.4	Introduced

IMSL_STATDATA

The IMSL_STATDATA function retrieves commonly analyzed data sets.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_STATDATA(*choice*)

Return Value

An array containing the desired data set is returned.

Arguments

choice

Data set indicator. See [Table 25-7](#) for a list of values for *choice*.

choice	Number of Rows	Number of Columns	Description of Data Set
1	16	7	Longley
2	176	2	Wolfer sunspot
3	150	5	Fisher iris
4	144	1	Box and Jenkins Series G
5	13	5	Draper and Smith Appendix B
6	197	1	Box and Jenkins Series A

Table 25-7: choice Values

choice	Number of Rows	Number of Columns	Description of Data Set
7	296	2	Box and Jenkins Series J
8	100	4	Robinson Multichannel Time Series
9	113	34	Affi and Azen Data Set A

Table 25-7: choice Values (Continued)

Keyword

DOUBLE

If present and nonzero, double precision is used.

Discussion

The `IMSL_STATDATA` function retrieves a standard data set frequently cited in statistics text books or in this manual. [Table 25-8](#) gives the references for each data set:

choice	References
1	Longley (1967)
2	Anderson (1971, p. 660)
3	Fisher (1936); Mardia <i>et al.</i> (1979, Table 1.2.2)
4	Box and Jenkins (1976, p. 531)
5	Draper and Smith (1981, pp. 629–630)
6	Box and Jenkins (1976, p. 525)
7	Box and Jenkins (1976, pp. 532–533)

Table 25-8: Standard Data Set References

choice	References
8	Robinson (1967, p. 204)
9	Afifi and Azen (1979, pp. 16–22)

Table 25-8: Standard Data Set References (Continued)

Example

In this example, `IMSL_STATDATA` is used to copy the Draper and Smith (1981, Appendix B) data set into `X`.

```
x = IMSL_STATDATA(5)
PM, x
```

7.00000	26.0000	6.00000	60.0000	78.5000
1.00000	29.0000	15.0000	52.0000	74.3000
11.0000	56.0000	8.00000	20.0000	104.300
11.0000	31.0000	8.00000	47.0000	87.6000
7.00000	52.0000	6.00000	33.0000	95.9000
11.0000	55.0000	9.00000	22.0000	109.200
3.00000	71.0000	17.0000	6.00000	102.700
1.00000	31.0000	22.0000	44.0000	72.5000
2.00000	54.0000	18.0000	22.0000	93.1000
21.0000	47.0000	4.00000	26.0000	115.900
1.00000	40.0000	23.0000	34.0000	83.8000
11.0000	66.0000	9.00000	12.0000	113.300
10.0000	68.0000	8.00000	12.0000	109.400

Version History

6.4	Introduced

IMSL_BINOMIALCOEF

The IMSL_BINOMIALCOEF function evaluates the binomial coefficient.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

Result = IMSL_BINOMIALCOEF(*n*, *m* [, /DOUBLE])

Return Value

The binomial coefficient:

$$\binom{n}{m}$$

is returned.

Arguments

m

Second parameter of the binomial coefficient. Parameter *m* must be nonnegative.

n

First parameter of the binomial coefficient. Parameter *n* must be nonnegative.

Keywords

DOUBLE

If present and nonzero, double precision is used.

Discussion

The binomial function is defined to be:

with $n \geq m \geq 0$. Also, *n* must not be so large that the function overflows.

$$\binom{n}{m} = \frac{n!}{m!(n-m)!}$$

Example

In this example:

$$\binom{9}{5}$$

is computed and printed.

```
n = 9
m = 5
ans = IMSL_BINOMIALCOEF(n, m)
PRINT, 'binomial coefficient =', ans

binomial coefficient =      126.000
```

Version History

6.4	Introduced

IMSL_NORM

The IMSL_NORM function computes various norms of a vector or the difference of two vectors.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

```
Result = IMSL_NORM(x[, y] [, INDEX_MAX=variable] [, INF=value]  
[, ONE=value] )
```

Return Value

The requested norm of the input vector. If the norm cannot be computed, NaN is returned.

Arguments

x

Vector for which the norm is to be computed.

y

If present, IMSL_NORM computes the norm of $(x - y)$.

Keywords

INDEX_MAX

Named variable into which the index of the element of x with the maximum modulus is stored. If *Index_Max* is used, then the keyword *Inf* also must be used. If the parameter y is specified, then the index of $(x - y)$ with the maximum modulus is stored.

INF

If present and nonzero, computes the infinity norm $\max |x_i|$.

ONE

If present and nonzero, computes the 1-norm

$$\sum_{i=0}^{n-1} |x_i|$$

Discussion

By default, `IMSL_NORM` computes the Euclidean norm as follows:

$$\left(\sum_{i=0}^{n-1} x_i^2 \right)^{\frac{1}{2}}$$

If the keyword *One* is set, then the 1-norm:

$$\sum_{i=0}^{n-1} |x_i|$$

is returned. If the keyword *Inf* is set, the infinity norm $\max |x_i|$ is returned. In the case of the infinity norm, the index of the element with maximum modulus also is returned.

If the parameter *y* is specified, the computations of the norms described above are performed on $(x - y)$.

Examples**Example 1**

In this example, the Euclidean norm of an input vector is computed.

```
x = [ 1.0, 3.0, -2.0, 4.0 ]
n = IMSL_NORM(x)
PM, n, Title = 'Euclidean norm of x:'

Euclidean norm of x:
5.47723
```

Example 2

This example computes $\max |x_i - y_i|$ and prints the norm and index.

```
x = [1.0, 3.0, -2.0, 4.0]
y = [4.0, 2.0, -1.0, -5.0]
n = IMSL_NORM(x, y, /Inf, Index_Max = imax)
PM, n, Title = 'Infinity norm of (x-y):'
PM, imax, Title = 'Element of (x-y) with maximum modulus:'

Infinity norm of (x-y):
    9.00000
Element of (x-y) with maximum modulus:
    3
```

Version History

6.4	Introduced

IMSL_MATRIX_NORM

The IMSL_MATRIX_NORM function computes various norms of a rectangular matrix, a matrix stored in band format, and a matrix stored in coordinate format.

Note

This routine requires an IDL Advanced Math and Stats license. For more information, contact your sales or technical support representative.

Syntax

To compute various norms of a rectangular matrix:

```
Result = IMSL_MATRIX_NORM(a [, /DOUBLE] [, INF_NORM=value]  
[, ONE_NORM=value] [, SYMMETRIC=value])
```

To compute various norms of a matrix stored in band format:

```
Result = IMSL_MATRIX_NORM(n, nlca, nuca, a [, /DOUBLE]  
[, INF_NORM=value] [, ONE_NORM=value] [, SYMMETRIC=value])
```

To compute various norms of a matrix stored in coordinate format:

```
Result = IMSL_MATRIX_NORM(nrows, ncols, a [, /DOUBLE]  
[, INF_NORM=value] [, ONE_NORM=value] [, SYMMETRIC=value])
```

Return Value

The requested norm of the input matrix, by default, the Frobenius norm. If the norm cannot be computed, NaN is returned.

Arguments

a

Matrix for which the norm will be computed.

n

The order of matrix A.

ncols

The number of columns in matrix *A*.

nlca

Number of lower codiagonals of *A*.

nrows

The number of rows in matrix *A*.

nuca

Number of upper codiagonals of *A*.

Keywords**DOUBLE**

If present and nonzero, double precision is used.

INF_NORM

If present and nonzero, `IMSL_MATRIX_NORM` computes the infinity norm of matrix *A*.

ONE_NORM

If present and nonzero, `IMSL_MATRIX_NORM` computes the one norm of matrix *A*.

SYMMETRIC

If present and nonzero, matrix *A* is stored in symmetric storage mode. Keyword *Symmetric* can not be used with a rectangular matrix.

Discussion

By default, `IMSL_MATRIX_NORM` computes the Frobenius norm:

$$\|A\|_2 = \left[\sum_{i=0}^{m-1} \sum_{j=0}^{n-1} A_{ij}^2 \right]^{\frac{1}{2}}$$

If the keyword *One_Norm* is used, the one norm

$$\|A\|_1 = \max_{0 \leq j \leq n-1} \sum_{i=0}^{m-1} |A_{ij}|$$

is returned. If the keyword *Inf_Norm* is used, the infinity norm

$$\|A\|_\infty = \max_{0 \leq i \leq m-1} \sum_{j=0}^{n-1} |A_{ij}|$$

is returned.

Examples

Example 1

Compute the Frobenius norm, infinity norm, and one norm of matrix *A*.

```
a = TRANSPOSE([[1.0, 2.0, -2.0, 3.0], $
               [-2.0, 1.0, 3.0, 0.0], [0.0, 3.0, 1.0, -7.0], $
               [5.0, -2.0, 7.0, 6.0], [4.0, 3.0, 4.0, 0.0]])
frobenius_norm = IMSL_MATRIX_NORM(a)
inf_norm = IMSL_MATRIX_NORM(a, /INF_NORM)
one_norm = IMSL_MATRIX_NORM(a, /ONE_NORM)
PRINT, 'Frobenius norm = ', frobenius_norm
PRINT, 'Infinity norm = ', inf_norm
PRINT, 'One norm      = ', one_norm

Frobenius norm =      15.6844
Infinity norm  =      20.0000
One norm      =      17.0000
```

Example 2

Compute the Frobenius norm, infinity norm, and one norm of matrix *A*. Matrix *A* is stored in band storage mode.

```
nlca = 1
nuca = 1
n = 4
a = [0.0, 2.0, 3.0, -1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 3.0, 4.0, 0.0]
frobenius_norm = IMSL_MATRIX_NORM(n, nlca, nuca, a)
inf_norm = IMSL_MATRIX_NORM(n, nlca, nuca, a, /INF_NORM)
one_norm = IMSL_MATRIX_NORM(n, nlca, nuca, a, /ONE_NORM)
PRINT, 'Frobenius norm = ', frobenius_norm
PRINT, 'Infinity norm = ', inf_norm
PRINT, 'One norm      = ', one_norm
```

```

Frobenius norm =      6.55744
Infinity norm  =      5.00000
One norm       =      8.00000

```

Example 3

Compute the Frobenius norm, infinity norm, and one norm of matrix A. Matrix A is stored in symmetric band storage mode.

```

nlca  =  2
nuca  =  2
n      =  6
a      =  [0.0, 0.0, 7.0, 3.0, 1.0, 4.0, $
           0.0, 5.0, 1.0, 2.0, 1.0, 2.0, 1.0, 2.0, 4.0, 6.0, 3.0, 1.0]
frobenius_norm =  IMSL_MATRIX_NORM(n, nlca, nuca, a, /SYMMETRIC)
inf_norm      =  IMSL_MATRIX_NORM(n, nlca, nuca, a, /INF_NORM, $
                        /SYMMETRIC)
one_norm      =  IMSL_MATRIX_NORM(n, nlca, nuca, a, /ONE_NORM, $
                        /SYMMETRIC)
PRINT, 'Frobenius norm = ', frobenius_norm
PRINT, 'Infinity norm = ', inf_norm
PRINT, 'One norm      = ', one_norm

Frobenius norm =      16.9411
Infinity norm  =      16.0000
One norm       =      16.0000

```

Example 4

Compute the Frobenius norm, infinity norm, and one norm of matrix A. Matrix A is stored in coordinate format.

```

nrows  =  6
ncols  =  6
a      =  REPLICATE(imsl_f_sp_elem, 15)
a(*).row =  [0, 1, 1, 1, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5]
a(*).col =  [0, 1, 2, 3, 2, 0, 3, 4, 0, 3, 4, 5, 0, 1, 5]
a(*).val =  [10.0, 10.0, -3.0, -1.0, 15.0, $
           -2.0, 10.0, -1.0, -1.0, -5.0, 1.0, -3.0, -1.0, -2.0, 6.0]
frobenius_norm =  IMSL_MATRIX_NORM(nrows, ncols, a)
inf_norm      =  IMSL_MATRIX_NORM(nrows, ncols, a, /INF_NORM)
one_norm      =  IMSL_MATRIX_NORM(nrows, ncols, a, /ONE_NORM)
PRINT, 'Frobenius norm = ', frobenius_norm
PRINT, 'Infinity norm = ', inf_norm
PRINT, 'One norm      = ', one_norm

Frobenius norm =      24.8395
Infinity norm  =      15.0000
One norm       =      18.0000

```

Example 5

Compute the Frobenius norm, infinity norm and one norm of matrix A. Matrix A is stored in symmetric coordinate format.

```
nrows = 6
ncols = 6
a = REPLICATE(imsf_sp_elem, 9)
a(*).row = [0, 0, 0, 1, 1, 2, 2, 4, 4]
a(*).col = [0, 2, 5, 3, 4, 2, 5, 4, 5]
a(*).val = [10.0, -1.0, 5.0, 2.0, 3.0, 3.0, 4.0, -1.0, 4.0]
frobenius_norm = IMSL_MATRIX_NORM(nrows, ncols, a, /SYMMETRIC)
inf_norm = IMSL_MATRIX_NORM(nrows, ncols, a, /INF_NORM, $
    /SYMMETRIC)
one_norm = IMSL_MATRIX_NORM(nrows, ncols, a, /ONE_NORM, $
    /SYMMETRIC)
PRINT, 'Frobenius norm = ', frobenius_norm
PRINT, 'Infinity norm = ', inf_norm
PRINT, 'One norm      = ', one_norm

Frobenius norm =      15.8745
Infinity norm  =      16.0000
One norm       =      16.0000
```

Version History

6.4	Introduced

PM

The PM procedure performs formatted output of arrays using the standard linear algebraic convention: “row” refers to the first index of the array and “column” refers to the second. By contrast, other IDL routines (such as PRINT) perform formatted output of arrays using the standard image processing convention: “column” refers to the first index of the array and “row” refers to the second.

The PM procedure is used extensively in the examples in the *IDL Advanced Math and Stats*. For multidimensional arrays, the syntax

```
PM, array
```

is equivalent to

```
PRINT, TRANSPOSE(array)
```

Syntax

```
PM, Array0 [, ... , Array19]
```

Arguments

Array_n

The arrays to be displayed. The PM routine can display up to 20 arrays.

Keywords

None.

Example

```
; Define an array arr
arr = [[1.0, 3.0], [0.0, 4.0], [2.0, 1.0]]
; Print using PM and PRINT
PM, arr & PRINT & PRINT, arr
```

IDL prints:

```
1.00000      0.000000      2.00000
3.00000      4.00000      1.00000

1.00000      3.00000
0.000000      4.00000
2.00000      1.00000
```

Version History

6.4	Introduced

RM

The RM procedure performs formatted input of arrays using the standard linear algebraic convention: “row” refers to the first index of the array and “column” refers to the second. By contrast, simply defining an array at the IDL command line creates an array the standard image processing convention: “column” refers to the first index of the array and “row” refers to the second.

The RM procedure is used extensively in the examples in the *IDL Advanced Math and Stats*. For multidimensional arrays, defining an array interactively using RM is equivalent to defining the same array using normal IDL syntax and then transposing the array.

Syntax

RM, Array, Rows, Columns

Arguments

Array

A named variable that will contain the array.

Rows

An integer specifying the number of rows in the array.

Columns

An integer specifying the number of columns in the array.

Note

If the user enters more data than will fit in the specified number of columns, the extra data is discarded.

Keywords

None

Example

Define a 3 row by 2 column array:

```
RM, arr, 3, 2
```

IDL prompts for input;

```
row 0: 1,4
row 1: 6,3
row 2: 9,9
```

Display the array using PM:

```
PM, arr
```

IDL Prints:

```
1.00000      4.00000
6.00000      3.00000
9.00000      9.00000
```

Display the array using PRINT:

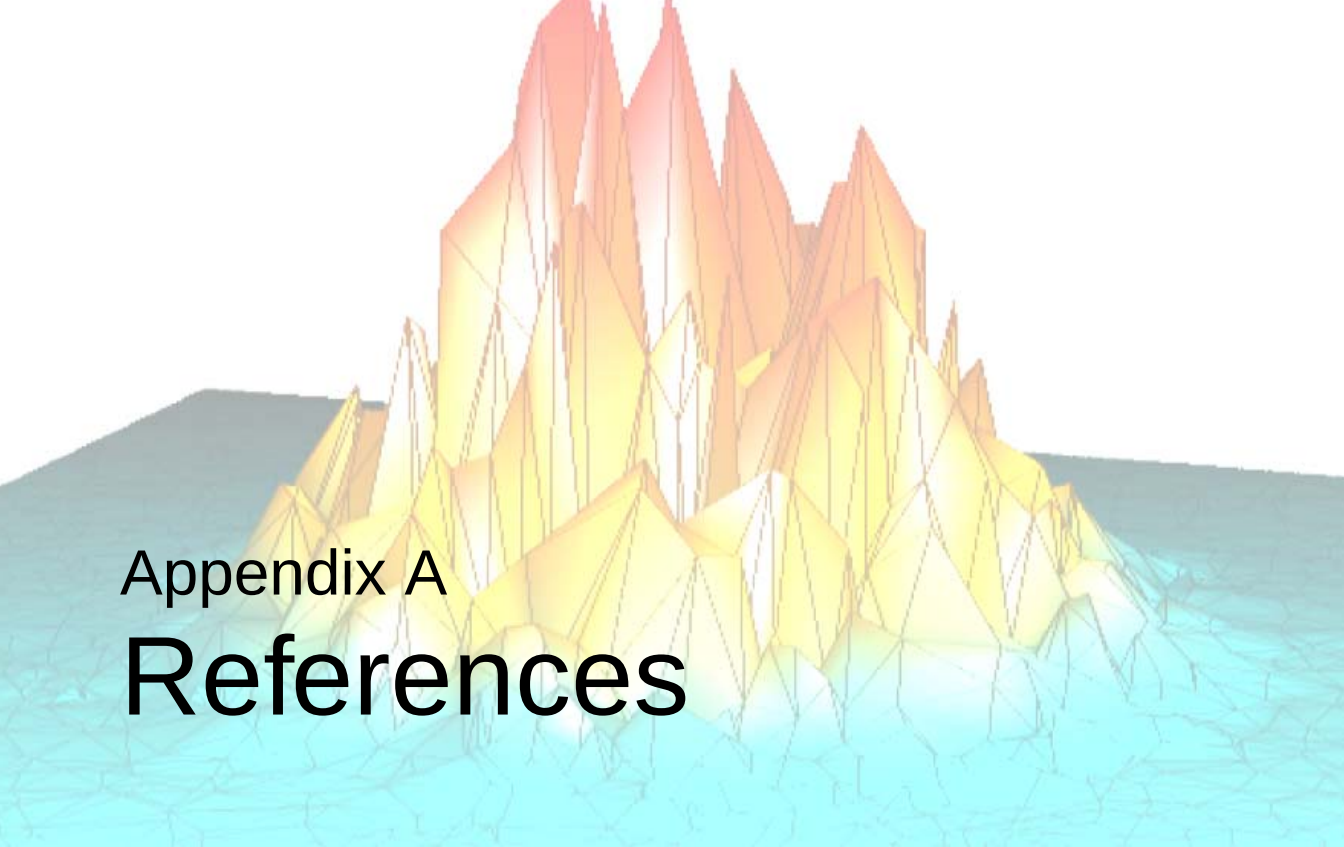
```
PRINT, arr
```

IDL Prints:

```
1.00000      6.00000      9.00000
4.00000      3.00000      9.00000
```

Version History

6.4	Introduced



Appendix A

References

Abramowitz, Milton, and Irene A. Stegun (editors) (1964), *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*, National Bureau of Standards, Washington, D.C.

Afifi, A.A., and S.P. Azen (1979), *Statistical Analysis: A Computer Oriented Approach*, 2d ed., Academic Press, New York.

Ahrens, J.H., and U. Dieter (1974), Computer methods for sampling from gamma, beta, Poisson, and binomial distributions, *Computing*, **12**, 223–246.

Akaike, H. (1978), A Bayesian analysis of the minimum AIC procedure, *Ann. Institute Statist. Mathematics.*, **30A**, 9–14.

Akaike, H. (1973), Information theory and an extension of maximum likelihood principle, *Proc. 2nd International Symposium on Information Theory*, Eds. B.N. Petrov and F. Csaki, 267–281.

Akima, H. (1978), A method of bivariate interpolation and smooth surface fitting for irregularly distributed data points, *ACM Transactions on Mathematical Software*, **4**, 148–159.

- Akima, H. (1970), A new method of interpolation and smooth curve fitting based on local procedures, *Journal of the ACM*, **17**, 589–602.
- Anderson, R.L., and T.A. Bancroft (1952), *Statistical Theory in Research*, McGraw-Hill Book Company, New York.
- Anderson, T.W. (1971), *The Statistical Analysis of Time Series*, John Wiley & Sons, New York.
- Ashcraft, C. (1987), *A vector implementation of the multifrontal method for large sparse symmetric positive definite systems*, Technical Report ETA-TR-51, Engineering Technology Applications Division, Boeing Computer Services, Seattle, Washington.
- Ashcraft, C., R. Grimes, J. Lewis, B. Peyton, and H. Simon (1987), Progress in sparse matrix methods for large linear systems on vector supercomputers. *Intern. J. Supercomputer Applic.*, **1**(4), 10-29.
- Atkinson, A.C. (1979), A family of switching algorithms for the computer generation of beta random variates, *Biometrika*, **66**, 141–145.
- Atkinson, A.C. (1985), *Plots, Transformations, and Regression*, Claredon Press, Oxford.
- Atkinson, Ken (1978), *An Introduction to Numerical Analysis*, John Wiley & Sons, New York.
- Barnett, A.R. (1981), An algorithm for regular and irregular Coulomb and Bessel functions of real order to machine accuracy, *Computer Physics Communication*, **21**, 297–314.
- Barrett, J.C., and M.J.R. Healy (1978), A remark on Algorithm AS 6: Triangular decomposition of a symmetric matrix, *Applied Statistics*, **27**, 379–380.
- Bays, Carter, and S.D. Durham (1976), Improving a poor random number generator, *ACM Transactions on Mathematical Software*, **2**, 59–64.
- Bishop, Yvonne M.M., Stephen E. Fienberg, and Paul W. Holland (1975), *Discrete Multivariate Analysis: Theory and Practice*, MIT Press, Cambridge, Mass.
- Blom, Gunnar (1958), *Statistical Estimates and Transformed Beta-Variables*, John Wiley & Sons, New York.
- de Boor, Carl (1978), *A Practical Guide to Splines*, Springer-Verlag, New York.
- Bosten, Nancy E., and E.L. Battiste (1974), Incomplete beta ratio, *Communications of the ACM*, **17**, 156–157.

- Box, George E.P., and Gwilyn M. Jenkins (1976), *Time Series Analysis: Forecasting and Control*, revised ed., Holden-Day, Oakland.
- Box, G.E.P., and P.W. Tidwell (1962), Transformation of the independent variables, *Technometrics*, **4**, 531–550.
- Brent, Richard P. (1973), *Algorithms for Minimization without Derivatives*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey.
- Brigham, E. Oran (1974), *The Fast Fourier Transform*, Prentice-Hall, Englewood Cliffs, New Jersey.
- Brown, Morton B., and Jacqueline K. Benedetti (1977), Sampling behavior and tests for correlation in two-way contingency tables, *Journal of the American Statistical Association*, **42**, 309–315.
- Brown, Morton E. (1983), MCDP4F, two-way and multiway frequency tables—measures of association and the log-linear model (complete and incomplete tables), in *BMDP Statistical Software, 1983 Printing with Additions*, (edited by W.J. Dixon), University of California Press, Berkeley.
- Carlson, R.E., and T.A. Foley (1991), The parameter R^2 in multiquadric interpolation, *Computer Mathematical Applications*, **21**, 29–42.
- Cheng, R.C.H. (1978), Generating beta variates with nonintegral shape parameters, *Communications of the ACM*, **21**, 317–322.
- Cohen, E. Richard, and Barry N. Taylor (1986), *The 1986 Adjustment of the Fundamental Physical Constants*, Codata Bulletin, Pergamon Press, New York.
- Conover, W.J. (1980), *Practical Nonparametric Statistics*, 2d ed., John Wiley & Sons, New York.
- Conover, W.J., and Ronald L. Iman (1983), *Introduction to Modern Business Statistics*, John Wiley & Sons, New York.
- Cook, R. Dennis, and Sanford Weisberg (1982), *Residuals and Influence in Regression*, Chapman and Hall, New York.
- Cooley, J.W., and J.W. Tukey (1965), An algorithm for the machine computation of complex Fourier series, *Mathematics of Computation*, **19**, 297–301.
- Cooper, B.E. (1968), Algorithm AS4, An auxiliary function for distribution integrals, *Applied Statistics*, **17**, 190–192.
- Craven, Peter, and Grace Wahba (1979), Smoothing noisy data with spline functions, *Numerische Mathematik*, **31**, 377–403.

- Crowe, Keith, Yuan-An Fan, Jing Li, Dale Neaderhouser, and Phil Smith (1990), *A direct sparse linear equation solver using linked list storage*, IMSL Technical Report 9006, IMSL, Houston.
- D'Agostino, Ralph B., and Michael A. Stevens (1986), *Goodness-of-Fit Techniques*, Marcel Dekker, New York.
- Davis, Philip F., and Philip Rabinowitz (1984), *Methods of Numerical Integration*, Academic Press, Orlando, Florida.
- Dallal, Gerald E. and Leland Wilkinson (1986), An analytic approximation to the distribution of Lilliefors's test statistic for normality, *The American Statistician*, **40**, 294–296.
- Dennis, J.E., Jr., and Robert B. Schnabel (1983), *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, Englewood Cliffs, New Jersey.
- Devore, Jay L (1982), *Probability and Statistics for Engineering and Sciences*, Brooks/Cole Publishing Company, Monterey, Calif.
- Dongarra, J.J., J.R. Bunch, C.B. Moler, and G.W. Stewart (1979), *LINPACK User's Guide*, SIAM, Philadelphia.
- Draper, N.R., and H. Smith (1981), *Applied Regression Analysis*, 2d ed., John Wiley & Sons, New York.
- Du Croz, Jeremy, P. Mayes, and G. Radicati (1990), Factorization of band matrices using Level-3 BLAS, *Proceedings of CONPAR 90-VAPP IV*, Lecture Notes in Computer Science, Springer, Berlin, 222.
- Duff, I. S., and J. K. Reid (1983), The multifrontal solution of indefinite sparse symmetric linear equations. *ACM Transactions on Mathematical Software*, **9**, 302–325.
- Duff, I. S., and J. K. Reid (1984), The multifrontal solution of unsymmetric sets of linear equations. *SIAM Journal on Scientific and Statistical Computing*, **5**, 633–641.
- Duff, I. S., A. M. Erisman, and J. K. Reid (1986), *Direct Methods for Sparse Matrices*, Clarendon Press, Oxford.
- Efroymson, M.A. (1960), Multiple regression analysis, *Mathematical Methods for Digital Computers*, Volume 1, (edited by A. Ralston and H. Wilf), John Wiley & Sons, New York, 191–203.
- Emmett, W.G. (1949), Factor analysis by Lawless method of maximum likelihood, *British Journal of Psychology, Statistical Section*, **2**, 90–97.

- Enright, W.H., and J.D. Pryce (1987), Two FORTRAN packages for assessing initial value methods, *ACM Transactions on Mathematical Software*, **13**, 1–22.
- Farebrother, R.W., and G. Berry (1974), A remark on Algorithm AS 6: Triangular decomposition of a symmetric matrix, *Applied Statistics*, **23**, 477.
- Fisher, R.A. (1936), The use of multiple measurements in taxonomic problems, *The Annals of Eugenics*, **7**, 179–188.
- Fishman, George S., and Louis R. Moore (1982), A statistical evaluation of multiplicative congruential random number generators with modulus $2^{31} - 1$, *Journal of the American Statistical Association*, **77**, 129–136.
- Forsythe, G.E. (1957), Generation and use of orthogonal polynomials for fitting data with a digital computer, *SIAM Journal on Applied Mathematics*, **5**, 74–88.
- Franke, R. (1982), Scattered data interpolation: Tests of some methods, *Mathematics of Computation*, **38**, 181–200.
- Furnival, G.M. and R.W. Wilson, Jr. (1974), Regressions by leaps and bounds, *Technometrics*, **16**, 499–511.
- Gautschi, Walter (1968), Construction of Gauss-Christoffel quadrature formulas, *Mathematics of Computation*, **22**, 251–270.
- Gear, C.W. (1971), *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice-Hall, Englewood Cliffs, New Jersey.
- Gentleman, W. Morven (1974), Basic procedures for large, sparse or weighted linear least squares problems, *Applied Statistics*, **23**, 448–454.
- George, A., and J. W. H. Liu (1981), *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Englewood Cliffs, New Jersey.
- Gill, P.E., W. Murray, M.A. Saunders, and M.H. Wright (1985), Model building and practical aspects of nonlinear programming, *Computational Mathematical Programming*, (edited by K. Schittkowski), NATO ASI Series, **15**, Springer-Verlag, Berlin, Germany.
- Girschick, M.A. (1939), On the sampling theory of roots of determinantal equations, *Annals of Mathematical Statistics*, **10**, 203–224.
- Goldfarb, D., and A. Idnani (1983), A numerically stable dual method for solving strictly convex quadratic programs, *Mathematical Programming*, **27**, 1–33.
- Golub, G.H. (1973), Some modified matrix eigenvalue problems, *SIAM Review*, **15**, 318–334.

- Golub, Gene H., and Charles F. Van Loan (1983), *Matrix Computations*, Johns Hopkins University Press, Baltimore, Md.
- Golub, G.H., and C.F. Van Loan (1989), *Matrix Computations*, 2d ed., The Johns Hopkins University Press, Baltimore, Maryland.
- Golub, G.H., and J.H. Welsch (1969), Calculation of Gaussian quadrature rules, *Mathematics of Computation*, **23**, 221–230.
- Goodnight, James H. (1979), A tutorial on the SWEEP operator, *The American Statistician*, **33**, 149–158.
- Gregory, Robert, and David Karney (1969), *A Collection of Matrices for Testing Computational Algorithms*, Wiley-Interscience, John Wiley & Sons, New York.
- Griffin, R., and K.A. Redish (1970), Remark on Algorithm 347: An efficient algorithm for sorting with minimal storage, *Communications of the ACM*, **13**, 54.
- Grosse, Eric (1980), Tensor spline approximation, *Linear Algebra and its Applications*, **34**, 29–41.
- Hageman, Louis A., and David M. Young (1981), *Applied Iterative Methods*, Academic Press, New York.
- Haldane, J.B.S. (1939), The mean and variance of χ^2 when used as a test of homogeneity, when expectations are small, *Biometrika*, **31**, 346.
- Hardy, R.L. (1971), Multiquadric equations of topography and other irregular surfaces, *Journal of Geophysical Research*, **76**, 1905–1915.
- Harman, Harry H. (1976), *Modern Factor Analysis*, 3d ed. revised, University of Chicago Press, Chicago.
- Hart, John F., E.W. Cheney, Charles L. Lawson, Hans J. Maehly, Charles K. Mesztenyi, John R. Rice, Henry G. Thacher, Jr., and Christoph Witzgall (1968), *Computer Approximations*, John Wiley & Sons, New York.
- Hartigan, John A. (1975), *Clustering Algorithms*, John Wiley & Sons, New York.
- Hartigan, J.A., and M.A. Wong (1979), Algorithm AS 136: A K-means clustering algorithm, *Applied Statistics*, **28**, 100–108.
- Hayter, Anthony J. (1984), A proof of the conjecture that the Tukey-Kramer multiple comparisons procedure is conservative, *Annals of Statistics*, **12**, 61–75.
- Healy, M.J.R. (1968), Algorithm AS 6: Triangular decomposition of a symmetric matrix, *Applied Statistics*, **17**, 195–197.
- Hemmerle, William J. (1967), *Statistical Computations on a Digital Computer*, Blaisdell Publishing Company, Waltham, Mass.

- Higham, Nicholas J. (1988), FORTRAN Codes for estimating the one-norm of a real or complex matrix, with applications to condition estimation, *ACM Transactions on Mathematical Software*, **14**, 381–396.
- Hildebrand, F.B. (1956), *Introduction to Numerical Analysis*, McGraw Hill.
- Hindmarsh, A.C. (1974), *GEAR: Ordinary Differential Equation System Solver*, Lawrence Livermore National Laboratory Report UCID-30001, Revision 3, Lawrence Livermore National Laboratory, Livermore, California.
- Hinkley, David (1977), On quick choice of power transformation, *Applied Statistics*, **26**, 67–69.
- Hill, G.W. (1970), Student's t -distribution, *Communications of the ACM*, **13**, 617–619.
- Hoaglin, David C., and Roy E. Welsch (1978), The hat matrix in regression and ANOVA, *The American Statistician*, **32**, 17–22.
- Hocking, R.R. (1972), Criteria for selection of a subset regression: Which one should be used?, *Technometrics*, **14**, 967–970.
- Huber, Peter J. (1981), *Robust Statistics*, John Wiley & Sons, New York.
- Hull, T.E., W.H. Enright, and K.R. Jackson (1976), *User's Guide for DVERK—A Subroutine for Solving Nonstiff ODEs*, Department of Computer Science Technical Report 100, University of Toronto.
- Irvine, Larry D., Samuel P. Marin, and Philip W. Smith (1986), Constrained interpolation and smoothing, *Constructive Approximation*, **2**, 129–151.
- Jackson, K.R., W.H. Enright, and T.E. Hull (1978), A theoretical criterion for comparing Runge-Kutta formulas, *SIAM Journal of Numerical Analysis*, **15**, 618 – 641.
- Jenkins, M.A. (1975), Algorithm 493: Zeros of a real polynomial, *ACM Transactions on Mathematical Software*, **1**, 178–189.
- Jenkins, M.A., and J.F. Traub (1970), A three-stage algorithm for real polynomials using quadratic iteration, *SIAM Journal on Numerical Analysis*, **7**, 545–566.
- John, Peter W.M. (1971), *Statistical Design and Analysis of Experiments*, Macmillan Company, New York.
- Jöhnk, M.D. (1964), Erzeugung von Betaverteilten und Gammaverteilten Zufallszahlen, *Metrika*, **8**, 5–15.

- Jöreskog, K.G. (1977), Factor analysis by least squares and maximum-likelihood methods, *Statistical Methods for Digital Computers*, (edited by Kurt Enslein, Anthony Ralston, and Herbert S. Wilf), John Wiley & Sons, New York, 125–153.
- Kaiser, H.F. (1963), Image analysis, *Problems in Measuring Change*, (edited by C. Harris), University of Wisconsin Press, Madison, Wis.
- Kaiser, H.F., and J. Caffrey (1965), Alpha factor analysis, *Psychometrika*, **30**, 1–14.
- Kendall, Maurice G., and Alan Stuart (1973), *The Advanced Theory of Statistics, Volume 2: Inference and Relationship*, 3rd ed., Charles Griffin & Company, London.
- Kendall, Maurice G., and Alan Stuart (1979), *The Advanced Theory of Statistics, Volume 2: Inference and Relationship*, 4th ed., Oxford University Press, New York.
- Kendall, Maurice G., Alan Stuart, and J. Keith Ord (1983), *The Advanced Theory of Statistics, Volume 3: Design and Analysis, and Time Series*, 4th. ed., Oxford University Press, New York.
- Kennedy, William J., Jr. and James E. Gentle (1980), *Statistical Computing*, Marcel Dekker, New York.
- Kinnucan, P., and H. Kuki (1968), *A Single Precision Inverse Error Function Subroutine*, Computation Center, University of Chicago.
- Kirk, Roger E. (1982), *Experimental Design: Procedures for the Behavioral Sciences*, 2d ed., Brooks/Cole Publishing Company, Monterey, Calif.
- Knuth, Donald E. (1981), *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*, 2d ed., Addison-Wesley, Reading, Mass.
- Lawley, D.N., and A.E. Maxwell (1971), *Factor Analysis as a Statistical Method*, 2d ed., Butterworth, London.
- Learmonth, G.P., and P.A.W. Lewis (1973), *Naval Postgraduate School Random Number Generator Package LLRANDOM, NPS55LW73061A*, Naval Postgraduate School, Monterey, Calif.
- Leavenworth, B. (1960), Algorithm 25: Real zeros of an arbitrary function, *Communications of the ACM*, **3**, 602.
- Lehmann, E.L. (1975), *Nonparametrics: Statistical Methods Based on Ranks*, Holden-Day, San Francisco.
- Levenberg, K. (1944), A method for the solution of certain problems in least squares, *Quarterly of Applied Mathematics*, **2**, 164–168.
- Lewis, P.A.W., A.S. Goodman, and J.M. Miller (1969), A pseudorandom number generator for the System/360, *IBM Systems Journal*, **8**, 136–146.

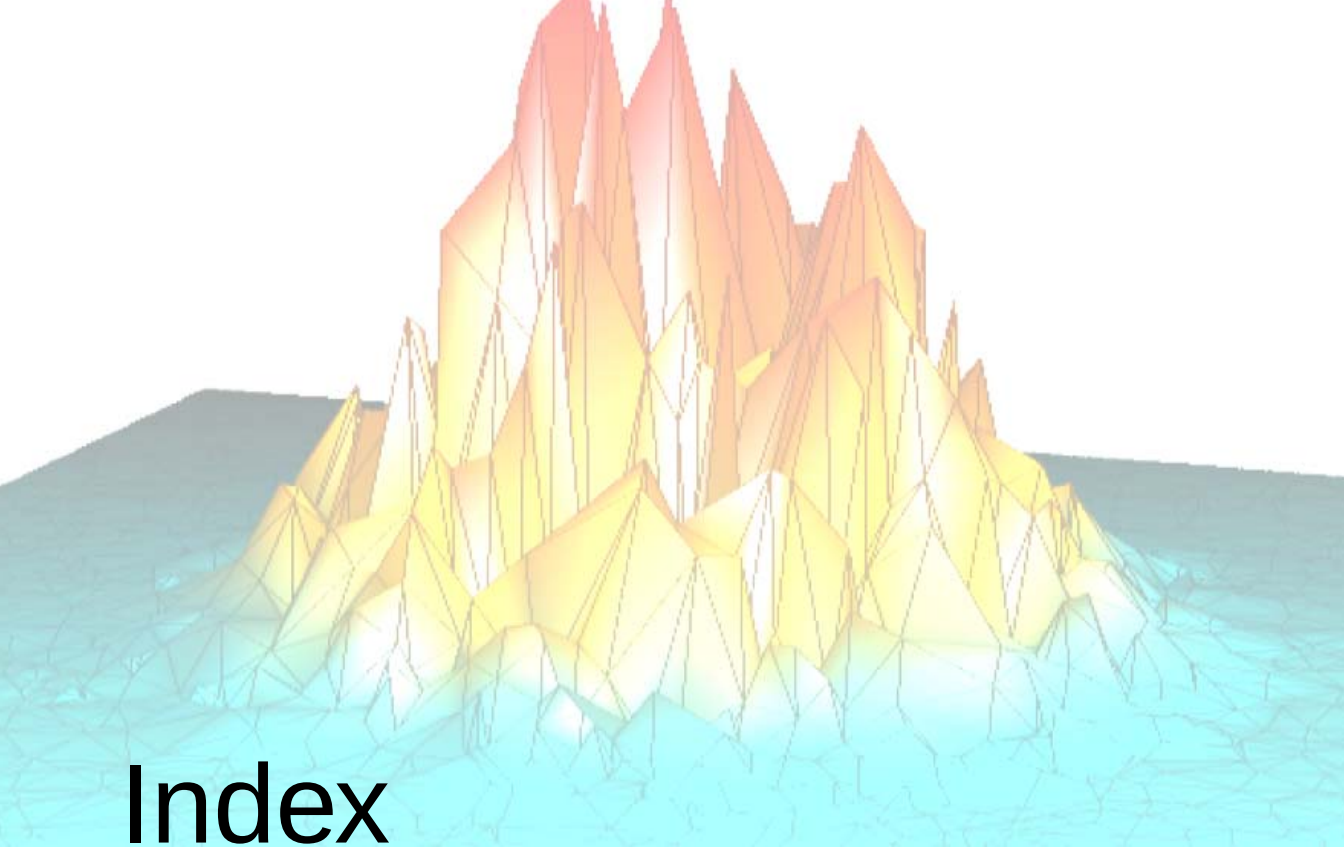
- Liepmann, David S. (1964), Mathematical constants, *Handbook of Mathematical Functions*, Dover Publications, New York.
- Lilliefors, H.W. (1967), On the Kolmogorov-Smirnov test for normality with mean and variance unknown, *Journal of the American Statistical Association*, **62**, 534–544.
- Liu, J. W. H. (1986), On the storage requirement in the out-of-core multifrontal method for sparse factorization. *ACM Transactions on Mathematical Software*, **12**, 249–264.
- Liu, J. W. H. (1987), *A collection of routines for an implementation of the multifrontal method*, Technical Report CS-87-10, Department of Computer Science, York University, North York, Ontario, Canada.
- Liu, J. W. H. (1989), The multifrontal method and paging in sparse Cholesky factorization. *ACM Transactions on Mathematical Software*, **15**, 310–325.
- Liu, J. W. H. (1990), *The multifrontal method for sparse matrix solution: theory and practice*, Technical Report CS-90-04, Department of Computer Science, York University, North York, Ontario, Canada.
- Longley, James W. (1967), An appraisal of least-squares programs for the electronic computer from the point of view of the user, *Journal of the American Statistical Association*, **62**, 819–841.
- Maindonald, J.H. (1984), *Statistical Computation*, John Wiley & Sons, New York.
- Mardia, K.V., J.T. Kent, J.M. Bibby (1979), *Multivariate Analysis*, Academic Press, New York.
- Marquardt, D. (1963), An algorithm for least-squares estimation of nonlinear parameters, *SIAM Journal on Applied Mathematics*, **11**, 431–441.
- Martin, R.S., and J.H. Wilkinson (1971), The Modified *LR* algorithm for complex Hessenberg matrices, *Volume II: Linear Algebra Handbook*, Springer, New York.
- Micchelli, C.A. (1986), Interpolation of scattered data: Distance matrices and conditionally positive definite functions, *Constructive Approximation*, **2**, 11–22.
- Micchelli, C.A., T.J. Rivlin, and S. Winograd (1976), The optimal recovery of smooth functions, *Numerische Mathematik*, **26**, 279–285.
- Micchelli, C.A., Philip W. Smith, John Swetits, and Joseph D. Ward (1985), Constrained L_p approximation, *Constructive Approximation*, **1**, 93–102.
- Müller, D.E. (1956), A method for solving algebraic equations using an automatic computer, *Mathematical Tables and Aids to Computation*, **10**, 208–215.

- Milliken, George A., and Dallas E. Johnson (1984), *Analysis of Messy Data, Volume 1: Designed Experiments*, Van Nostrand Reinhold, New York.
- Miller, Rupert G., Jr. (1980), *Simultaneous Statistical Inference*, 2d ed., Springer-Verlag, New York.
- Moré, Jorge, Burton Garbow, and Kenneth Hillstom (1980), *User Guide for MINPACK-1*, Argonne National Laboratory Report ANL 80-74, Argonne, Illinois.
- Murtagh, Bruce A. (1981), *Advanced Linear Programming: Computation and Practice*, McGraw-Hill, New York.
- Murty, Katta G. (1983), *Linear Programming*, John Wiley and Sons, New York.
- Nelson, Peter (1989), Multiple Comparisons of Means Using Simultaneous Confidence Intervals, *Journal of Quality Technology*, **21**, 232–241.
- Neter, John, and William Wasserman (1974), *Applied Linear Statistical Models*, Richard D. Irwin, Homewood, Ill.
- Neter, John, William Wasserman, and Michael H. Kutner (1983), *Applied Linear Regression Models*, Richard D. Irwin, Homewood, Illinois.
- Owen, D.B. (1962), *Handbook of Statistical Tables*, Addison-Wesley Publishing Company, Reading, Massachusetts.
- Owen, D.B. (1965), A special case of the bivariate non-central t distribution, *Biometrika*, **52**, 437–446.
- Parlett, B.N. (1980), *The Symmetric Eigenvalue Problem*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey.
- Petro, R. (1970), Remark on Algorithm 347: An efficient algorithm for sorting with minimal storage, *Communications of the ACM*, **13**, 624.
- Piessens, R., E. deDoncker-Kapenga, C.W. Überhuber, and D.K. Kahaner (1983), *QUADPACK*, Springer-Verlag, New York.
- Powell, M.J.D. (1978), A fast algorithm for nonlinearly constrained optimization calculations, in *Numerical Analysis Proceedings, Dundee 1977, Lecture Notes in Mathematics*, (edited by G. A. Watson), **630**, Springer-Verlag, Berlin, Germany, 144–157.
- Powell, M.J.D. (1985), On the quadratic programming algorithm of Goldfarb and Idnani, *Mathematical Programming Study*, **25**, 46–61.
- Powell, M.J.D. (1983), *ZQPCVX a FORTRAN subroutine for convex quadratic programming*, DAMTP Report 1983/NA17, University of Cambridge, Cambridge, England.

- Reinsch, Christian H. (1967), Smoothing by spline functions, *Numerische Mathematik*, **10**, 177–183.
- Rice, J.R. (1983), *Numerical Methods, Software, and Analysis*, Mcguire-Hill, New York.
- Rietman, Edward (1989), *Exploring the Geometry of Nature*, Windcrest Books, Blue Ridge Summit, Pennsylvania.
- Robinson, Enders A. (1967), *Multichannel Time Series Analysis with Digital Computer Programs*, Holden-Day, San Francisco.
- Royston, J.P. (1982a), An extension of Shapiro and Wilk's W test for normality to large samples, *Applied Statistics*, **31**, 115–124.
- Royston, J.P. (1982b), The W test for normality, *Applied Statistics*, **31**, 176–180.
- Royston, J.P. (1982c), Expected normal order statistics (exact and approximate), *Applied Statistics*, **31**, 161–165.
- Saad, Y., and M. H. Schultz (1986), GMRES: A generalized minimum residual algorithm for solving nonsymmetric linear systems, *SIAM Journal of Scientific and Statistical Computing*, **7**, 856–869.
- Sallas, William M., and Abby M. Lioni (1988), *Some useful computing formulas for the nonfull rank linear model with linear equality restrictions*, IMSL Technical Report 8805, IMSL, Houston.
- Savage, I. Richard (1956), Contributions to the theory of rank order statistics—the two-sample case, *Annals of Mathematical Statistics*, **27**, 590–615.
- Schittkowski, K. (1980), Nonlinear programming codes, *Lecture Notes in Economics and Mathematical Systems*, **183**, Springer-Verlag, Berlin, Germany.
- Schittkowski, K. (1983), On the convergence of a sequential quadratic programming method with an augmented Lagrangian line search function, *Mathematik Operations for Schung and Statistik, Serie Optimization*, **14**, 197–216.
- Schittkowski, K. (1986), NLPQL: A FORTRAN subroutine solving constrained nonlinear programming problems, (edited by Clyde L. Monma), *Annals of Operations Research*, **5**, 485–500.
- Schmeiser, Bruce (1983), Recent advances in generating observations from discrete random variates, *Computer Science and Statistics: Proceedings of the Fifteenth Symposium on the Interface*, (edited by James E. Gentle), North-Holland Publishing Company, Amsterdam, 154–160.
- Schmeiser, Bruce W., and A.J.G. Babu (1980), Beta variate generation via exponential majorizing functions, *Operations Research*, **28**, 917–926.

- Schmeiser, Bruce, and Voratas Kachitvichyanukul (1981), *Poisson Random Variate Generation*, Research Memorandum 81-4, School of Industrial Engineering, Purdue University, West Lafayette, Ind.
- Schmeiser, Bruce W., and Ram Lal (1980), Squeeze methods for generating gamma variates, *Journal of the American Statistical Association*, **75**, 679–682.
- Schwartz, G. (1978), Estimating the dimension of a model, *Ann. Statist.*, **6**, 461–464.
- Searle, S.R. (1971), *Linear Models*, John Wiley & Sons, New York.
- Shampine, L.F. (1975), Discrete least-squares polynomial fits, *Communications of the ACM*, **18**, 179–180.
- Shampine, L.F., and C.W. Gear (1979), A user's view of solving stiff ordinary differential equations, *SIAM Review*, **21**, 1–17.
- Singleton, R.C. (1969), Algorithm 347: An efficient algorithm for sorting with minimal storage, *Communications of the ACM*, **12**, 185–187.
- Smith, B.T., J.M. Boyle, J.J. Dongarra, B.S. Garbow, Y. Ikebe, V.C. Klema, and C.B. Moler (1976), *Matrix Eigensystem Routines—EISPACK Guide*, Springer-Verlag, New York.
- Smith, P.W. (1990), On knots and nodes for spline interpolation, *Algorithms for Approximation II*, J.C. Mason and M.G. Cox, Eds., Chapman and Hall, New York.
- Snedecor, George W., and William G. Cochran (1967), *Statistical Methods*, 6th ed., Iowa State University Press, Ames, Iowa.
- Spurrer, John D., and Steven P. Isham (1985), Exact simultaneous confidence intervals for pairwise comparisons of three normal means, *Journal of the American Statistical Association*, **80**, 438–442.
- Stewart, G.W. (1973), *Introduction to Matrix Computations*, Academic Press, New York.
- Stoer, J. (1985), Principles of sequential quadratic programming methods for solving nonlinear programs, *Computational Mathematical Programming*, (edited by K. Schittkowski), NATO ASI Series, **15**, Springer-Verlag, Berlin, Germany.
- Stoline, Michael R. (1981), The status of multiple comparisons: simultaneous estimation of all pairwise comparisons in one-way ANOVA designs, *The American Statistician*, **35**, 134–141.
- Strecok, Anthony J. (1968), On the calculation of the inverse of the error function, *Mathematics of Computation*, **22**, 144–158.

- Stroud, A.H., and D.H. Secrest (1963), *Gaussian Quadrature Formulae*, Prentice-Hall, Englewood Cliffs, New Jersey.
- Temme, N.M. (1975), On the numerical evaluation of the modified Bessel function of the third kind, *Journal of Computational Physics*, **19**, 324–337.
- Thompson, I.J., and A.R. Barnett (1987), Modified Bessel functions $I_\nu(z)$ and $K_\nu(z)$ of real order and complex argument, to selected accuracy, *Computer Physics Communication*, **47**, 245–257.
- Tukey, John W. (1962), The future of data analysis, *Annals of Mathematical Statistics*, **33**, 1–67.
- Velleman, Paul F., and David C. Hoaglin (1981), *Applications, Basics, and Computing of Exploratory Data Analysis*, Duxbury Press, Boston.
- Walker, H. F. (1988), Implementation of the GMRES method using Householder transformations, *SIAM Journal of Scientific and Statistical Computing*, **9**, 152–163.
- Watkins, David S., L. Elsner (1991), Convergence of algorithm of decomposition type for the eigenvalue problem, *Linear Algebra Applications*, **143**, 19–47.
- Weisberg, S. (1985), *Applied Linear Regression*, 2nd edition, John Wiley & Sons, New York.



Index

A

- abampere, [1155](#)
- Abcoulomb, [1155](#)
- Abvolt, [1155](#)
- acre, [1155](#)
- Adams' method, [333](#)
 - implicit, [337](#)
- Airy functions, [526](#)
 - IMSL_AIRY_AI, [526](#)
 - IMSL_AIRY_BI, [528](#)
- alpha-factor analysis method, [988](#)
- alternatives to least squares regression
 - IMSL_LIMSL_NORMREGRESS, [703](#)
- ampere, [1155](#)
- AMU, [1154](#)
- analysis of variance, [747](#)
 - factorial design, [760](#)
 - general linear model, [600](#)
 - IMSL_ANOVA1, [750](#)
 - IMSL_ANOVABALANCED, [781](#)
 - IMSL_ANOVAFACT, [760](#)
 - IMSL_ANOVANESTED, [772](#)
 - IMSL_MULTICOMP, [769](#)
 - n-way design, [760](#)
 - one-way design, [754](#)
- Angstrom, [1154](#)
- ANOVA. *See* analysis of variance
- ANSI/IEEE 754-1985, [1160](#)
- approximations
 - scattered data, [195](#)
 - smooth cubic splines, [254](#)
- arbitrary dimension quadrature
 - IMSL_INTFCN_QMC, [319](#)
- arbitrary dimension quadrature, [27](#), [283](#)
 - IMSL_INTFCNHYPE, [315](#)

area, [1155](#)
 association, measures of, [802](#)
 astronomical unit, [1153](#)
 asymptotic variances, [978](#)
 ATM, [1155](#)
 atmosphere, [1155](#)
 atmospheric pressure, standard, [1153](#)
 atomic mass unit, [1153](#)
 atto, [1156](#)
 AU, [1154](#)
 autoregressive parameters, [919](#)
 Avogadro's number, [1153](#)

B

backcasting, [915](#)
 backward difference operator, [930](#)
 backward glance, [645](#)
 backward selection, [639](#)
 balanced experimental design, [781](#)
 band storage mode, [1171](#)
 bar, [1155](#)
 base of natural logs, [1153](#)
 basic statistics
 IMSL_FREQTABLE, [563](#)
 IMSL_NORM1SAMP, [550](#)
 IMSL_NORM2SAMP, [555](#)
 IMSL_RANKS, [577](#)
 IMSL_SIMPLESTAT, [544](#)
 IMSL_SORTDATA, [570](#)
 basic uniform generator, [1066](#), [1066](#)
 basis function, [234](#)
 Bessel function
 first kind, [500](#)
 modified
 first kind, [498](#)
 second kind, [502](#)
 second kind, [504](#)
 Bessel functions, [506](#), [506](#), [508](#)
 Bessel functions with real order and complex argument

 IMSL_BESSI, [498](#)
 IMSL_BESSI_EXP, [506](#)
 IMSL_BESSJ, [500](#)
 IMSL_BESSK, [502](#)
 IMSL_BESSK_EXP, [508](#)
 IMSL_BESSY, [504](#)
 beta distribution, [1088](#), [1097](#)
 beta function
 real, [484](#)
 incomplete, [489](#)
 logarithmic, [487](#)
 binomial coefficient, [1166](#)
 binomial distribution, [1088](#)
 binomial distributions, [1107](#), [1109](#), [1112](#),
 [1115](#), [1123](#), [1127](#)
 binomial probability, [834](#)
 bisection process, [286](#)
 bivariate quadrature, [27](#), [283](#)
 bivariate quintic polynomial, [263](#)
 Blom normal scores, [581](#)
 Boltzman's constant, [1153](#)
 Bonferroni method, [756](#)
 b-spline and cubic spline evaluation and interpolation
 IMSL_SPINTEG, [230](#)
 IMSL_SPVALUE, [224](#)
 b-spline interpolation
 IMSL_BSINTERP, [210](#)
 IMSL_BSKNOTS, [219](#)
 B-splines, [193](#)
 least-squares approximations
 one-dimensional, [240](#), [242](#)
 one-dimensional, [212](#), [231](#)
 two-dimensional, [231](#)
 BTU, [1155](#)

C

Catalan's constant, [1153](#)
 categorical and discrete data analysis
 IMSL_CAT_GLM, [817](#)

- IMSL_CONTINGENCY, 796
- IMSL_EXACT_ENUM, 809
- IMSL_EXACT_NETWORK, 812
- cauchy distribution, 1089
- Cauchy principle value, 306, 307
- Celsius, 1155
- centi, 1156
- characteristic roots, 977
- characteristic vectors, 977
- charge, 1155
- Chebyshev moments, 301, 304
- chi-square distribution, 1089
- chi-squared
 - analysis, 796
 - goodness-of-fit test, 876, 878
 - measures relating to, 800
 - statistic, 800
 - test, 551, 557, 797
- chi-squared statistics, 794
- chi-squared test, 874
- Cholesky factorization
 - symmetric nonnegative definite, 112
 - symmetric positive definite, 93
- classification model, one-way, 750
- classification variables, 600
- Clenshaw-Curtis formula, 295, 301
- cluster analysis, 968, 971
- Cochran Q test, 865
- coefficient of variation, 548
- concavity, 205
- condition numbers, 80, 86, 175
- confidence intervals, 622, 657, 657
 - Bonferroni method, 751, 752, 753, 753, 754, 754, 756
 - Dunn-Sidak method, 751, 752, 753, 753, 754, 754, 755
 - means, 623
 - One-at-a-Time t (Fisher's LSD) method, 751, 752, 753, 753, 754, 754, 757
 - prediction, 623
 - Scheffé method, 623, 751, 752, 753, 753, 754, 754, 756
 - Tukey method, 751, 752, 753, 753, 754, 754, 755
 - Tukey-Kramer method, 751, 752, 753, 753, 754, 754, 755
- constants
 - computer, 1158
 - mathematical and physical, 1152
- constants and date sets
 - IMSL_MACHINE, 1158
- constraints, 248
- contants and date sets
 - IMSL_CONSTANT, 1152
- contingency coefficient, 797, 800, 898, 898, 899, 900, 900
- contingency tables, 809
 - two-way, 796
- continuous variables, 600
- convolution, discrete of 1D arrays, 390
- Cook's D statistics, 623, 658
- Cooley-Tukey algorithm, 374
- coordinate format, 1171
- copyrights, 2
- Cornish-Fisher expansion, 1047
- correlation and covariance
 - IMSL_COVARIANCES, 722
 - IMSL_PARTIAL_COV, 728
 - IMSL_POOLED_COV, 734
 - IMSL_ROBUST_COV, 738
- correlation coefficient
 - multiple, 609
- correlation matrix, 722, 723, 1109, 1115
- correlation, discrete of 1D arrays, 395
- correlations, 722, 728
- cosine Fresnel integrals, 522
- Coulomb, 1155
- counts, 545, 570
- covariances, 722
 - sample, 978
- Cox and Stuart sign test, 849
- Cramer's V, 800

cross-validation, [255](#)
 cubic spline interpolation
 IMSL_CSINTERP, [200](#)
 IMSL_CSSHAPE, [205](#)
 cubic splines, [193](#), [200](#)
 approximations
 smooth, [254](#)
 interpolation
 endpoint conditions, [200](#)
 shape-preserving, [205](#)
 smoothing, [195](#), [195](#)
 current, [1155](#)
 curvilinear regression, [652](#)

D

data sets, statistical
 retrieving, [1163](#)
 dates
 epoch to date, [1148](#)
 IMSL_DATETODAYS, [1150](#)
 IMSL_DAYSTODATE, [1148](#)
 number of days, [1150](#)
 deca, [1156](#)
 deci, [1156](#)
 degrees of freedom
 for error, [608](#), [640](#), [650](#)
 for the model, [608](#), [640](#), [650](#)
 total corrected, [608](#), [640](#), [650](#)
 Delaunay triangulation, [263](#)
 derivatives, [326](#)
 DFFITS statistics, [595](#), [624](#), [659](#)
 diagnostics, [622](#), [657](#)
 differential equations
 IMSL_ODE, [333](#)
 IMSL_PDE_MOL, [351](#)
 IMSL_POISSON2D, [366](#)
 differential equations, ordinary (IMSL_ODE)
 general, [329](#)
 mildly stiff, [337](#)
 Rossler system, [343](#)

Runge-Kutta method, [338](#)
 differentiation
 IMSL_FCN_DERIV, [326](#)
 discrete Fourier cosine transformation, [377](#),
 [377](#)
 discrete uniform distribution, [1093](#)
 distance, [1154](#)
 distribution functions
 beta probability, [1053](#)
 binomial distribution, [1056](#)
 binomial probability, [1058](#)
 chi-squared, noncentral, [1038](#)
 F distribution, [1043](#)
 gamma distribution, [1050](#)
 hypergeometric, [1060](#)
 normal
 bivariate, [1035](#)
 Gaussian, [1032](#)
 inverse, [1032](#)
 inverse, [1032](#)
 Poisson, [1063](#)
 Student's t, [1046](#)
 dummy variables, [601](#)
 Dunn-Sidak method, [755](#)
 dyne, [1155](#)

E

eigenexpansion, [178](#)
 eigensystem analysis, [968](#)
 IMSL_EIG, [178](#)
 IMSL_EIGSYMGEN, [183](#)
 IMSL_GENEIG, [186](#)
 eigenvalues, [186](#)
 eigenvalues and eigenvectors
 accuracy, [174](#)
 error analysis, [174](#)
 general, [178](#)
 generalized, reformulating, [175](#)
 symmetric positive definite, [183](#)
 eigenvectors, [186](#)

electron charge, [1153](#)
 electron mass, [1153](#)
 electron volt, [1153](#)
 elliptic integrals, [510](#), [514](#), [516](#)
 IMSL_ELE, [512](#)
 IMSL_ELK, [510](#)
 IMSL_ELRC, [520](#)
 IMSL_ELRD, [516](#)
 IMSL_ELRF, [514](#)
 IMSL_ELRJ, [518](#)
 endpoint conditions, [200](#)
 energy, [1155](#)
 equality/inequality constraints, [458](#)
 Equation, [400](#)
 Erg, [1155](#)
 Erlang distribution, [1051](#)
 error function
 real, [477](#)
 complementary, [480](#)
 error functions
 IMSL_BETA, [484](#)
 IMSL_BETAI, [489](#)
 IMSL_ERF, [477](#)
 IMSL_ERFC, [480](#)
 IMSL_LNBETA, [487](#)
 errors
 alert, [18](#)
 fatal, [18](#)
 Euler's constant, [1153](#)
 excess, coefficient of, [544](#), [547](#)
 exponential distribution, [1087](#)
 exponential mix distribution, [1090](#)
 exponential order statistics, [582](#)
 exponential scores, [577](#)
 export restrictions, [2](#)

F

F test statistic, [557](#), [560](#)
 factor analysis, [969](#), [981](#)
 factorial design, balanced, [760](#)

factorization
 Cholesky, [93](#)
 LU, [85](#)
 SVD, [104](#)
 factor-loading estimates, [981](#)
 Fahrenheit, [1155](#)
 farad, [1155](#)
 Faraday constant, [1153](#)
 fast Fourier transforms, [374](#)
 complex
 one-dimensional, [380](#)
 continuous vs. discrete, [374](#)
 real
 one-dimensional, [411](#), [411](#), [411](#)
 fatal errors, [18](#)
 Faure, [1137](#), [1141](#)
 Faure sequence, [1136](#), [1140](#), [1140](#)
 faure_next_point, [1140](#)
 femto, [1156](#)
 FFT. *See* fast Fourier transforms
 fine structure, [1153](#)
 finite differences, forward, [670](#)
 first-order IMSL_ODEs, [330](#)
 Fisher's LSD, [757](#)
 fixed points, [323](#)
 force, [1155](#)
 forecasts
 IMSL_GARCH, [952](#)
 forward finite differences, [670](#)
 forward selection, [639](#)
 Fourier sine and cosine transforms, [303](#)
 frequencies
 resolvable, [380](#)
 resolving dominant, [384](#)
 frequency, [1154](#)
 frequency tables
 multiway, [570](#)
 one-way, [563](#)
 frequency tabulation, [573](#)
 Fresnel integrals
 IMSL_FRESNEL_COSINE, [522](#)

IMSL_FRESNEL_SINE, 524
 Friedman's test, 860
 F-statistic, 608

G

gamma distribution, 1087
 gamma function
 real
 incomplete, 495
 logarithmic, 491
 gamma functions, 493
 IMSL_GAMMAI, 495
 IMSL_LNGAMMA, 491
 gamma statistic, 798
 gas constant, 1153
 Gauss, 1155
 Gauss Legendre quadrature, 3-point, 324
 Gauss quadrature
 IMSL_INTFCN, 322
 Gauss quadrature, 282, 322
 10-point, 286
 Gauss-Kronrod rules, 307
 21-point, 286
 7/15, 301
 Gauss-Lobatto quadrature, 322
 points and weights, 323
 Gauss-Radau quadrature, 322
 points, 323
 Gauss-Seidel method, 241
 Gear's method, 333
 general discrete distribution, 319, 1058, 1104,
 1107, 1112, 1118, 1120, 1123, 1127, 1127,
 1128
 general distributions, 874
 general goodness-of-fit tests
 IMSL_CHISQTEST, 876
 IMSL_KOLMOGOROV2, 889
 IMSL_KOLMOGROV1, 886
 IMSL_MVAR_NORMALITY, 892
 IMSL_NORMALITY, 882
 general linear models, 588, 600
 Generalized Autoregressive Conditional Het-
 eroskedastic, 952
 generalized categorical models
 IMSL_CAT_GLM, 817
 generalized eigensystem problems
 IMSL_EIGSYMGEN, 183
 IMSL_GENEIG, 186
 generalized feedback shift register method,
 1066
 generalized inverses, 104
 generalized linear models, 794
 generators
 basic uniform, 1066, 1066
 shuffled, 1067
 geometric distribution, 1090
 geometry
 IMSL_NORM, 1168
 vector norms, 1168
 GFSR, 1072
 GFSR generator, 1067
 GFSR method, 1066
 giga, 1156
 Givens transformations, 611
 globally adaptive scheme, 286
 Goodman and Kruskal τ , 803
 for columns, 798
 for rows, 798
 goodness-of-fit
 IMSL_CHISQTEST, 876
 IMSL_KOLMOGOROV2, 889
 IMSL_KOLMOGROV1, 886
 IMSL_MVAR_NORMALITY, 892
 IMSL_NORMALITY, 882
 IMSL_RANDOMNESS_TEST, 897
 goodness-of-fit tests, 874
 gravitational constant, 1154
 Gray code, 1138, 1142
 Gregorian calendar, 1151
 gridded data, 216
 G-squared test, 797

H

Hardy multiquadratic, 268
 Henry, 1155
 Hertz, 1154
 hypergeometric distribution, 1091
 hyper-rectangle, 315, 319

I

ideal gas, standard volume, 1154
 IEEE arithmetic, 1161
 ill-conditioning, 66
 image analysis method, 985, 988
 IMSL_AIRY_AI function, 526
 IMSL_AIRY_BI function, 528
 IMSL_ALLBEST procedure, 630
 IMSL_ANOVA1 function, 750
 IMSL_ANOVABALANCED function, 781
 IMSL_ANOVAFAC function, 760
 IMSL_ANOVANESTED function, 772
 IMSL_ARMA
 least-squares procedure, 915
 method-of-moments procedure, 915
 stationary, 931
 IMSL_ARMA function, 913
 IMSL_ARMA models
 IMSL_ARMA, 913
 IMSL_AUTOCORRELATION, 940
 IMSL_BOXCOXTRANS, 935
 IMSL_DIFFERENCE, 929
 IMSL_GARCH, 952
 IMSL_KALMAN, 957
 IMSL_LACK_OF_FIT, 948
 IMSL_PARTIAL_AC, 945
 IMSL_AUTOCORRELATION function, 940
 IMSL_BESSI function, 498
 IMSL_BESSI_EXP function, 506
 IMSL_BESSJ function, 500
 IMSL_BESSK function, 502
 IMSL_BESSK_EXP function, 508
 IMSL_BESSY function, 504
 IMSL_BETA function, 484
 IMSL_BETACDF function, 1053
 IMSL_BETAI function, 489
 IMSL_BINOMIALCDF function, 1056
 IMSL_BINOMIALCOEF function, 1166
 IMSL_BINOMIALPDF function, 1058
 IMSL_BINORMALCDF function, 1035
 IMSL_BOXCOXTRANS function, 935
 IMSL_BSINTERP function, 210
 IMSL_BSKNOTS function, 219
 IMSL_BSLSQ function, 238
 IMSL_CAT_GLM function, 817
 IMSL_CHFAC procedure, 93
 IMSL_CHISQCDF function, 1038
 IMSL_CHISQTEST function, 876
 IMSL_CHNNDFAC procedure, 112
 IMSL_CHNND SOL function, 108
 IMSL_CHSOL function, 89
 IMSL_COCHRANQ function, 865
 IMSL_CONLSQ function, 248
 IMSL_CONSTANT function, 1152
 IMSL_CONSTRAINED_NLP function, 465
 IMSL_CONT_TABLE procedure, 1118
 IMSL_CONTINGENCY function, 796
 IMSL_CONVOLID function, 390
 IMSL_CORRID function, 395
 IMSL_COVARIANCES function, 722
 IMSL_CSINTERP function, 200
 IMSL_CSSHAPE function, 205
 IMSL_CSSMOOTH function, 254
 IMSL_CSTRENDS function, 849
 IMSL_DATETODAYS function, 1150
 IMSL_DAYSTODATE procedure, 1148
 IMSL_DIFFERENCE function, 929
 IMSL_DISCR_ANALYSIS procedure, 992
 IMSL_DISCR_TABLE function, 1123
 IMSL_EIG function, 178
 IMSL_EIGSYMGEN function, 183
 IMSL_ELE function, 512
 IMSL_ELK function, 510

- IMSL_ELRC function, [520](#)
- IMSL_ELRD function, [516](#)
- IMSL_ELRF function, [514](#)
- IMSL_ELRJ function, [518](#)
- IMSL_ERF function, [477](#)
- IMSL_ERFC function, [480](#)
- IMSL_EXACT_ENUM function, [809](#)
- IMSL_EXACT_NETWORK function, [812](#)
- IMSL_FACTOR_ANALYSIS function, [981](#)
- IMSL_FAURE_INIT function, [1136](#)
- IMSL_FAURE_NEXT_PT function, [1140](#)
- IMSL_FCDF function, [1043](#)
- IMSL_FCN_DERIV function, [326](#)
- IMSL_FCNLSQ function, [234](#)
- IMSL_FFTCOMP function, [377](#)
- IMSL_FFTINIT function, [387](#)
- IMSL_FMIN function, [425](#)
- IMSL_FMINV function, [433](#)
- IMSL_FREQTABLE function, [563](#)
- IMSL_FRESNEL_COSINE function, [522](#)
- IMSL_FRESNEL_SINE function, [524](#)
- IMSL_FRIEDMANS_TEST function, [860](#)
- IMSL_GAMMA_ADV function, [493](#)
- IMSL_GAMMACDF function, [1050](#)
- IMSL_GAMMAI function, [495](#)
- IMSL_GARCH function, [952](#)
- IMSL_GENEIG procedure, [186](#)
- IMSL_GQUAD procedure, [322](#)
- IMSL_HYPERGEOCDF function, [1060](#)
- IMSL_HYPOTH_PARTIAL function, [675](#)
- IMSL_HYPOTH_SCPH function, [681](#)
- IMSL_HYPOTH_TEST function, [686](#)
- IMSL_INTFCN function, [284](#)
 - algebraic-logarithmic singularities, [294](#)
 - Cauchy principle value, [306](#)
 - fourier sine and cosine transforms, [303](#)
 - Gauss-Kronrod rules, [288](#)
 - infinite or semi-infinite interval, [297](#)
 - sine and cosine factors, [300](#)
 - singular points given, [291](#)
 - smooth functions using nonadaptive rule, [309](#)
 - two-dimensional iterated integrals, [312](#)
- IMSL_INTFCN_QMC function, [319](#)
- IMSL_INTFCNHYPHER function, [315](#)
- IMSL_INV function, [77](#)
- IMSL_K_MEANS function, [971](#)
- IMSL_KALMAN procedure, [957](#)
- IMSL_KELVIN_BEI0 function, [533](#)
- IMSL_KELVIN_BER0 function, [531](#)
- IMSL_KELVIN_KEI0 function, [537](#)
- IMSL_KELVIN_KERO function, [535](#)
- IMSL_KOLMOGOROV1 function, [886](#)
- IMSL_KOLMOGOROV2 function, [889](#)
- IMSL_KTRENDS function, [868](#)
- IMSL_KW_TEST function, [857](#)
- IMSL_LACK_OF_FIT function, [948](#)
- IMSL_LAPLACE_INV function, [398](#)
- IMSL_LINLSQ function, [116](#)
- IMSL_LINPROG function, [449](#)
- IMSL_LNBETA function, [487](#)
- IMSL_LNGAMMA function, [491](#)
- IMSL_LNORMREGRESS function, [703](#)
- IMSL_LUFAC procedure, [85](#)
- IMSL_LUSOL function, [79](#)
- IMSL_MACHINE function, [1158](#)
- IMSL_MATRIX_NORM function, [1171](#)
- IMSL_MINCONGEN function, [458](#)
- IMSL_MULTICOMP function, [769](#)
- IMSL_MULTIPREDICT function, [622](#)
- IMSL_MULTIREGRESS function, [607](#)
- IMSL_MVAR_NORMALITY function, [892](#)
- IMSL_NCTRENDS function, [846](#)
- IMSL_NLINLSQ function, [441](#)
- IMSL_NONLINOPT function, [694](#)
- IMSL_NONLINREGRESS function, [665](#)
- IMSL_NORM function, [1168](#)
- IMSL_NORM1SAMP function, [550](#)
- IMSL_NORM2SAMP function, [555](#)
- IMSL_NORMALCDF function, [1032](#)
- IMSL_NORMALITY function, [882](#)
- IMSL_ODE function, [333](#)
- IMSL_ODE. *See* differential equations

- IMSL_PARTIAL_AC function, [945](#)
- IMSL_PARTIAL_COV function, [728](#)
- IMSL_PDE_MOL function, [351](#)
- IMSL_POISSON2D function, [366](#)
- IMSL_POISSONCDF function, [1063](#)
- IMSL_POLYPREDICT function, [657](#)
- IMSL_POLYREGRESS function, [649](#)
- IMSL_POOLED_COV function, [734](#)
- IMSL_PRINC_COMP function, [976](#)
- IMSL_QRFAC procedure, [100](#)
- IMSL_QRSOL function, [96](#)
- IMSL_QUADPROG function, [454](#)
- IMSL_RADBE function, [277](#)
- IMSL_RADBF function, [266](#)
- IMSL_RAND_FROM_DATA function, [1115](#)
- IMSL_RAND_GEN_CONT function, [1120](#)
- IMSL_RAND_GEN_DISCR function, [1127](#)
- IMSL_RAND_ORTH_MAT function, [1109](#)
- IMSL_RAND_TABLE_2WAY function, [1107](#)
- IMSL_RANDOM function, [1080](#)
- IMSL_RANDOM_ARMA function, [1131](#)
- IMSL_RANDOM_NPP function, [1100](#)
- IMSL_RANDOM_ORDER function, [1104](#)
- IMSL_RANDOM_SAMPLE function, [1112](#)
- IMSL_RANDOM_TABLE procedure, [1076](#)
- IMSL_RANDOMNESS_TEST function, [897](#)
- IMSL_RANDOMOPT procedure, [1071](#)
- IMSL_RANKS function, [577](#)
- IMSL_REGRESSORS function, [600](#)
- IMSL_ROBUST_COV function, [738](#)
- IMSL_SCAT2DINTERP function, [262](#)
- IMSL_SIGNTEST function, [834](#)
- IMSL_SIMPLESTAT function, [544](#)
- IMSL_SMOOTHDATA1D function, [258](#)
- IMSL_SORTDATA function, [570](#)
- IMSL_SP_BDFAC procedure, [139](#)
- IMSL_SP_BDPDFAC function, [155](#)
- IMSL_SP_BDPDSOL function, [152](#)
- IMSL_SP_BDSOL function, [135](#)
- IMSL_SP_CG function, [163](#)
- IMSL_SP_GMRES function, [159](#)
- IMSL_SP_LUFAC function, [128](#)
- IMSL_SP_LUSOL function, [122](#)
- IMSL_SP_MVMUL function, [167](#)
- IMSL_SP_PDFAC function, [148](#)
- IMSL_SP_PDSOL function, [143](#)
- IMSL_SPINTEG function, [230](#)
- IMSL_SPVALUE function, [224](#)
- IMSL_STATDATA function, [1163](#)
- IMSL_STEPWISE procedure, [639](#)
- IMSL_SURVIVAL_GLM function, [1006](#)
- IMSL_SVDCOMP function, [104](#)
- IMSL_TCDF function, [1046](#)
- IMSL_TIE_STATS function, [855](#)
- IMSL_WILCOXON function, [837](#)
- IMSL_ZEROFCN function, [413](#)
- IMSL_ZEROPOLY function, [410](#)
- IMSL_ZEROSYS function, [418](#)
- incomplete gamma function, [495](#)
- indicator variables, [601](#)
- inferences about the mean, [559](#)
- initial value problem (IVP), [329](#), [333](#)
 - nonstiff, [329](#)
 - stiff, [329](#)
- integrals
 - n-dimensional iterated, [316](#)
 - two-dimensional iterated, [312](#), [313](#)
- integration, [319](#)
 - arbitrary dimension quadrature, [27](#), [283](#)
 - Gauss quadrature, [282](#), [322](#)
 - multivariate
 - general, [281](#)
 - hyper-rectangle, [315](#)
 - two-dimensional, [312](#), [313](#)
 - spline, one or two-dimensional, [230](#)
 - univariate / bivariate
 - Cauchy principle, [306](#), [307](#)
 - Gauss-Kronrod rules, [288](#), [289](#)
 - general, [280](#)
 - infinite or semi-infinite interval, [297](#), [298](#)
 - nonadaptive rule, [309](#), [310](#)
 - sine or cosine factor, [300](#), [301](#)

- sine or cosine transform, [303](#), [304](#)
- smooth function, [309](#), [310](#)
- with algebraic-logarithmic singularities, [294](#), [295](#)
- with singularity points, [291](#), [292](#)
- interpolation
 - cubic spline
 - endpoint conditions, [200](#)
 - shape preserving, [205](#)
 - scattered data, [195](#)
 - radial-basis fit, [277](#)
 - radial-basis functions, [266](#)
 - user-supplied, [272](#)
 - smooth bivariate, [262](#)
 - three-dimensional fit, [273](#)
 - spline
 - knot sequence, [219](#)
 - one-dimensional, [215](#)
 - two-dimensional, [213](#)
- interpolation and approximation
 - IMSL_BSINTERP, [210](#)
 - IMSL_BSKNOTS, [219](#)
 - IMSL_BSLSQ, [238](#)
 - IMSL_CONLSQ, [248](#)
 - IMSL_CSINTERP, [200](#)
 - IMSL_CSSHAPE, [205](#)
 - IMSL_CSSMOOTH, [254](#)
 - IMSL_FCMLSQ, [234](#)
 - IMSL_RADBE, [277](#)
 - IMSL_RADBF, [266](#)
 - IMSL_SCAT2DINTERP, [262](#)
 - IMSL_SMOOTHDATA1D, [258](#)
 - IMSL_SPINTEG, [230](#)
 - IMSL_SPVALUE, [224](#)
- inverse
 - complementary error function, [480](#)
 - error function, [477](#)
 - g3, [614](#)
 - generalized, [614](#)
 - Moore-Penrose, [614](#)
- inverse matrix, [77](#)

IVP. *See* initial value problem

J

- Jacobian matrix, [443](#), [666](#)
- Jenkins-Traub three-stage algorithm, [410](#)
- Joule, [1155](#)
- Julian calendar, [1149](#), [1151](#)

K

- Kalman filtering, [957](#)
- Kappa analysis, [794](#)
- kappa statistic, [798](#), [804](#)
- Kelvin, [1155](#)
- Kelvin functions
 - IMSL_KELVIN_BEI0, [533](#)
 - IMSL_KELVIN_BER0, [531](#)
 - IMSL_KELVIN_KEI0, [537](#)
 - IMSL_KELVIN_KER0, [535](#)
- Kendall's τ_b , [798](#)
- key sort, [572](#)
- kilo, [1156](#)
- K-means analysis, [971](#)
- knot sequence, [212](#)
- knots, [212](#), [219](#)
- Kolmogorov one-sample test, [886](#)
- Kolmogorov two-sample test, [889](#)
- Kruskal-Wallis test, [798](#), [804](#)
- k-sample trends test, [868](#)
- kurtosis, [544](#), [547](#)

L

- lack-of-fit statistics, [651](#)
- lack-of-fit tests, [595](#)
- Least Absolute Value, [598](#)
- Least Maximum Value, [598](#)
- Least Squares
 - Alternatives

- Least Absolute Value, [598](#)
- Least Maximum Value, [598](#)
- Lp Norm, [598](#)
- least squares approximation and smoothing
 - IMSL_BSLSQ, [238](#)
 - IMSL_CONLSQ, [248](#)
 - IMSL_CSSMOOTH, [254](#)
 - IMSL_FCNLSQ, [234](#)
 - IMSL_SMOOTHDATA1D, [258](#)
- least-squares fit, [26](#), [195](#), [199](#), [258](#), [607](#), [781](#), [846](#), [849](#), [855](#), [860](#), [886](#), [889](#)
- B-spline
 - one-dimensional, [240](#)
 - two-dimensional, [242](#)
- spline
 - constrained, [248](#)
 - one- or two-dimensional, [238](#)
- user-supplied function, [234](#)
- weighted, [616](#)
- least-squares method, [986](#)
 - generalized, [985](#)
 - unweighted, [985](#)
 - weighted, [592](#)
- least-squares solution, [65](#)
- Lebesgue measure, [1137](#), [1141](#)
- legalities, [2](#)
- Levenberg-Marquardt algorithm, modified, [441](#), [670](#)
- leverages, [595](#), [624](#)
- Lilliefors test, [882](#), [883](#)
- linear constraints, [116](#), [251](#)
- linear dependence, [591](#)
- linear eigensystem problems
 - IMSL_EIG, [178](#)
- linear equations with full matrices
 - IMSL_CHFAC, [93](#)
 - IMSL_CHSOL, [89](#)
 - IMSL_LUFAC, [85](#)
 - IMSL_LUSOL, [79](#)
- linear least squares with full matrices
 - IMSL_CHNNDFAC, [112](#)
 - IMSL_CHNND SOL, [108](#)
 - IMSL_LINLSQ, [116](#)
 - IMSL_QRFAC, [100](#)
 - IMSL_QRSOL, [96](#)
 - IMSL_SVDCOMP, [104](#)
- linear least-squares problem, [116](#)
- linear programming problems, [449](#)
- linear regression
 - multiple, [586](#)
 - simple, [586](#)
- linear system solution
 - general, [64](#), [64](#), [79](#)
 - Hermitian positive definite, [91](#)
 - matrix factorization, [64](#)
 - multiple right-hand sides, [65](#), [82](#)
 - symmetric nonnegative definite, [108](#)
 - symmetric positive definite, [89](#)
- linear systems
 - IMSL_CHFAC, [93](#)
 - IMSL_CHNNDFAC, [112](#)
 - IMSL_CHNND SOL, [108](#)
 - IMSL_CHSOL, [89](#)
 - IMSL_INV, [77](#)
 - IMSL_LINLSQ, [116](#)
 - IMSL_LUFAC, [85](#)
 - IMSL_LUSOL, [79](#)
 - IMSL_QRFAC, [100](#)
 - IMSL_QRSOL, [96](#)
 - IMSL_SP_BDFAC, [139](#)
 - IMSL_SP_BDPDFAC, [155](#)
 - IMSL_SP_BDPDSOL, [152](#)
 - IMSL_SP_BDSOL, [135](#)
 - IMSL_SP_CG, [163](#)
 - IMSL_SP_GMRES, [159](#)
 - IMSL_SP_LUFAC, [128](#)
 - IMSL_SP_LUSOL, [122](#)
 - IMSL_SP_MVMUL, [167](#)
 - IMSL_SP_PDFAC, [148](#)
 - IMSL_SP_PDSOL, [143](#)
 - IMSL_SVDCOMP, [104](#)
- linear trend test, [804](#)

- linearly constrained minimization
 - IMSL_LINPROG, 449
 - IMSL_QUADPROG, 454
- linearly dependent regressors, 612
- logarithm, gamma function, 491
- logarithm, real beta function, 487
- logarithmic distribution, 1091
- lognormal distribution
 - random numbers
 - lognormal distribution, 1092
- low-discrepancy, 1138, 1142
- low-discrepancy sequences
 - IMSL_FAURE_INIT, 1136
 - IMSL_FAURE_NEXT_PT, 1140
- Lp Norm, 598
- LU factorization, 85

M

- MAD. *See* median absolute deviation
- magnetic induction, 1155
- Mallows Cp criterion, 633
- Mann-Whitney U test, 840
- mass, 1154
- mathematical constants, 1152
- matrices, sparse. *See* sparse matrices
- matrix
 - notation, 66
- matrix factorization, 64
- matrix inversion
 - IMSL_INV, 77
 - linear system solution, 64
- matrix norm
 - IMSL_MATRIX_NORM, 1171
- maximum, 544, 547
- maximum likelihood estimates, 962
- maximum likelihood method, 985, 986
- Maxwell, 1155
- McNemar test, 799, 805
- mean, 544, 547, 550, 608, 640, 650
 - exact, 797, 898, 898, 899, 900, 900
 - for two normal populations, 555
 - inferences about, 559
 - lower confidence limit, 545
 - normal population, 550
 - return value, 552
 - upper confidence limit, 545
- mean square
 - error, 608, 640, 650
 - model, 608, 640, 650
- measures of
 - association, 801
 - prediction, 802
 - uncertainty, 802
- measures of association, 794
- median, 548
- median absolute deviation, 548
- mega, 1156
- method of provisional means, 725
- micro, 1156
- micron, 1154
- mill, 1154
- milli, 1156
- minimization, 421, 458, 465
 - linearly constrained, 422
 - quadratic programming, 454
 - simplex algorithm, 449
 - nonlinearly constrained, 423
 - unconstrained, 422
 - nonlinear least squares, 441
 - quasi-Newton method, 433, 436
 - univariate, 425
- minimum, 544, 547
- missing values, 17, 598
- models
 - general linear, 600
 - multiple linear regression, 607, 630
 - nonlinear regression, 592, 665
 - polynomial, 587
 - polynomial regression, 657
 - regression, 622
- modified Bessel function, 498

- mole, 1155
- Moore-Penrose inverses, 110, 114, 614
- Müller's method, 413
- multiple linear regression
 - IMSL_MULTIPREDICT, 622
 - IMSL_REGRESSORS, 600
 - MULTIGRESS, 607
- multiple linear regression models, 586, 600, 607, 630, 639, 781, 846, 849, 855, 860, 886, 889
- multiple right-hand sides, 65
- multiple-comparisons test
 - Student-Newman-Keuls, 769
- multiplicative congruential generator, 1066
- multiplicative generator, 1066
- multivariate analysis
 - cluster analysis, 971
 - factor analysis, 981
 - IMSL_DISCR_ANALYSIS, 992
 - IMSL_FACTOR_ANALYSIS, 981
 - IMSL_K_MEANS, 971
 - IMSL_PRINC_COMP, 976
 - principal components, 976
- multivariate distribution, 1115
- multivariate linear regression - statistical inference and diagnostics
 - IMSL_HYPOTH_PARTIAL, 675
 - IMSL_HYPOTH_SCPH, 681
 - IMSL_HYPOTH_TEST, 686
- multivariate normal distribution, 1083, 1088, 1098
- multivariate quadrature, 281
- multiway frequency table, 570
- myria, 1156
- Newton's Method, 206, 419
- nino, 1156
- Noether test, 846
- noncentral chi-squared distribution function, 1038
- nonlinear equations
 - IMSL_ZEROFCN, 413
 - IMSL_ZEROPOLY, 410
 - IMSL_ZEROSYS, 418
- nonlinear least-squares problems, 441
- nonlinear programming problem, 465
- nonlinear regression models, 592, 665
- nonlinearly constrained minimization, 465
 - IMSL_MINCONGEN, 458
- nonparametric statistics
 - IMSL_COCHRANQ_TEST, 865
 - IMSL_CSTRENDS, 849
 - IMSL_FRIEDSMANS_TEST, 860
 - IMSL_KTRENDS, 868
 - IMSL_KW_TEST, 857
 - IMSL_NCTRENDS, 846
 - IMSL_SIGNTEST, 834
 - IMSL_TIE_STATS, 855
 - IMSL_WILCOXON, 837
- nonstiff IVPs, 329
- nonuniform generators, 1068
- normal distribution, 1086
- normal populations
 - mean, 550
 - variances, 550
- normal scores, 577
- normality test, 882
- not-a-knot condition, 200, 215, 226
- numerical ranking, 577
- Nyquist phenomenon, 380

N

- NaN (Not a Number), 17, 598
- natural logs, base, 1153
- n-dimensional iterated integrals, 316
- negative binomial, 1092

O

- observations, number of, 545
- Ohm, 1155
- one sample tests - nonparametric statistics

IMSL_CSTRENDS, 849
 IMSL_NCTRENDS, 846
 IMSL_SIGNTTEST, 834
 IMSL_TIE_STATS, 855
 IMSL_WILCOXON, 837
 One-at-a-Time t method, 757
 one-way classification model, 750
 one-way frequency table, 563
 optimal prediction, 798, 798
 optimization
 IMSL_CONSTRAINED_NLP, 465
 IMSL_FMIN, 425
 IMSL_FMINV, 433
 IMSL_LINPROG, 449
 IMSL_MINCONGEN, 458
 IMSL_NLINLSQ, 441
 IMSL_QUADPROG, 454
 ordinary differential equation. *See* differential equations
 over-determined system, 65
 overflow, 17

P

parsec, 1154
 partial correlations, 728
 partial covariances, 728
 partial differential equations, 331
 partial pivoting, 80
 Paterson rules, nested, 310
 periodic interpolant, 201
 phi, 797, 800, 898, 898, 899, 900, 900
 physical constants, 1152
 Pi, 1154
 pico, 1156
 piecewise polynomials, 192, 195, 231
 Planck's constant, 1154
 PM procedure, 1176
 poise, 1155
 Poisson distribution, 1087, 1097
 polynomial and nonlinear regression

IMSL_NONLINOPT, 694
 IMSL_NONLINREGRESS, 665
 IMSL_POLYPREDICT, 657
 IMSL_POLYREGRESS, 649
 polynomial models, 587
 polynomial regression models, 657
 pooled variances, 557
 Powell hybrid algorithm, 418
 predicted values, 622, 657, 657, 667
 prediction coefficient, 802
 pressure, 1155
 principal components, 976
 principal components method, 985, 985
 principal factor method, 985, 985
 probability distribution functions and inverses
 IMSL_BETACDF, 1053
 IMSL_BINOMIALCDF, 1056
 IMSL_BINOMIALPDF, 1058
 IMSL_BINORMALCDF, 1035
 IMSL_CHISQCDF, 1038
 IMSL_FCDF, 1043
 IMSL_GAMMACDF, 1050
 IMSL_HYPERGEOCDF, 1060
 IMSL_NORMALCDF, 1032
 IMSL_POISSONCDF, 1063
 IMSL_TCDF, 1046
 product moment correlation, 798
 proton mass, 1154
 provisional means, method of, 725
 pseudorandom number generators, 874
 pseudorandom numbers, 1100, 1115, 1118, 1120, 1123, 1127, 1128
 pseudorandom order statistics, 1104, 1105, 1105
 pseudorandom orthogonal matrix, 1109
 pseudorandom sample, 1112
 p-values, 608, 640, 650, 801

Q

QP. *See* quadratic programming

QR factorization
 linear least squares, 96
 real matrix, 100

quadrature

 IMSL_FCN_DERIV, 326
 IMSL_GQUAD, 322
 IMSL_INTFCN, 284
 IMSL_INTFCN_QMC, 319
 IMSL_INTFCNHYPHER, 315

quadratic programming

 convex problems, 423, 455
 dual algorithm, 423, 455
 linearly constrained, 454

quadrature points and weights, 322

quasi-Monte Carlo, 319

quasi-Newton method, 436

R

R matrix, 591, 614, 668

R2 criterion, 608, 633, 640, 650

 adjusted, 608, 630, 640, 650

radial-basis fit, 266

radial-basis functions, 195

random number generation

 IMSL_CONT_TABLE, 1118
 IMSL_DISCR_TABLE, 1123
 IMSL_FAURE_INIT, 1136
 IMSL_FAURE_NEXT_PT, 1140
 IMSL_RAND_FROM_DATA, 1115
 IMSL_RAND_GEN_CONT, 1120
 IMSL_RAND_GEN_DISCR, 1127
 IMSL_RAND_ORTH_MAT, 1109
 IMSL_RAND_TABLE_2WAY, 1107
 IMSL_RANDOM, 1080
 IMSL_RANDOM_ARMA, 1131
 IMSL_RANDOM_NPP, 1100
 IMSL_RANDOM_ORDER, 1104
 IMSL_RANDOM_SAMPLE, 1112
 IMSL_RANDOM_TABLE, 1076
 IMSL_RANDOMPT, 1071

random numbers, 1066

 beta distribution, 1088, 1097

 binomial distribution, 1088

 cauchy distribution, 1089

 chi-squared distribution, 1089

 control the seed, 1071

 discrete uniform distribution, 1093

 exponential distribution, 1087

 exponential mix distribution, 1090

 gamma distribution, 1087

 generate pseudorandom numbers, 1080

 geometric distribution, 1090

 hypergeometric distribution, 1091

 IMSL_CONT_TABLE, 1118

 IMSL_DISCR_TABLE, 1123

 IMSL_RAND_FROM_DATA, 1115

 IMSL_RAND_GEN_CONT, 1120

 IMSL_RAND_GEN_DISCR, 1127

 IMSL_RAND_ORTH_MAT, 1109

 IMSL_RAND_TABLE_2WAY, 1107

 IMSL_RANDOM, 1080

 IMSL_RANDOM_NPP, 1100

 IMSL_RANDOM_ORDER, 1104

 IMSL_RANDOM_SAMPLE, 1112

 IMSL_RANDOM_TABLE, 1076

 IMSL_RANDOMPT, 1071

 logarithmic distribution, 1091

 multivariate normal distribution, 1083, 1088,
 1098

 negative binomial, 1092

 normal distribution, 1086

 Poisson distribution, 1087, 1097

 select the form, 1071

 Student's t distribution, 1093

 triangular distribution, 1093

 von Mises distribution, 1093

 Weibull distribution, 1094

randomness test, 897

range, 544, 547

ranks, 104, 577

real beta function, 484

real complementary error function, [480](#)
 real error function, [477](#)
 real incomplete beta function, [489](#)
 rectangular matrix, [1171](#)
 regression
 all best, [630](#)
 curvilinear, [649](#)
 general linear model, [600](#)
 IMSL_ALLBEST, [630](#)
 IMSL_HYPOTH_PARTIAL, [675](#)
 IMSL_HYPOTH_SCPH, [681](#)
 IMSL_HYPOTH_TEST, [686](#)
 IMSL_LNORMREGRESS, [703](#)
 IMSL_MULTIPREDICT, [622](#)
 IMSL_NONLINOPT, [694](#)
 IMSL_NONLINREGRESS, [665](#)
 IMSL_POLYPREDICT, [657](#)
 IMSL_POLYREGRESS, [649](#)
 IMSL_REGRESSORS, [600](#)
 IMSL_STEPWISE, [639](#)
 MULTIGRESS, [607](#)
 multiple linear, [607](#)
 nonlinear, [665](#)
 polynomial least-squares, [649](#)
 simple linear, [586](#)
 stepwise, [639](#)
 regression coefficients, [610](#), [630](#), [642](#), [665](#)
 regression models, [586](#), [622](#)
 regression simple linear, [607](#)
 regressors, [600](#)
 residuals, [624](#), [659](#), [668](#)
 deleted, [595](#), [624](#), [659](#)
 jackknife, [595](#)
 standardized, [595](#), [624](#), [659](#)
 resolvable frequencies, [380](#)
 RM procedure, [1178](#)
 root of a system, [408](#)
 root of a system of equations
 IMSL_ZEROSYS, [418](#)
 Runge-Kutta-Verner method
 fifth-order, [330](#), [333](#)

 sixth-order, [330](#), [333](#)
 Rydberg's constant, [1154](#)

S

sample covariance, [978](#)
 Satterthwaite's procedure, [560](#)
 Savage scores, [579](#)
 scaling results of IMSL_RANDOM, [1097](#)
 scattered data
 approximation, [195](#)
 interpolation, [195](#)
 scattered data interpolation
 IMSL_RADBE, [277](#)
 IMSL_RADBF, [266](#)
 IMSL_SCAT2DINTERP, [262](#)
 Scheffé confidence intervals, [623](#)
 Scheffé method, [756](#)
 shape-preserving cubic splines, [205](#)
 Shapiro-Wilk W test, [882](#), [883](#)
 shuffled generators, [1067](#)
 shuffling, [1071](#)
 sign test, [834](#)
 simple summary statistics
 IMSL_NORM1SAMP, [550](#)
 IMSL_NORM2SAMP, [555](#)
 IMSL_SIMPLESTAT, [544](#)
 simplex algorithm, [449](#)
 sine Fresnel integrals, [524](#)
 single value decomposition (SVD), [65](#), [104](#)
 singularity, [66](#)
 skewness, coefficient of, [544](#), [547](#)
 slug, [1154](#)
 smooth data
 cubic spline interpolant, [259](#)
 error detection, [258](#)
 smoothed data, [258](#)
 smoothing parameter, [254](#)
 smoothing spline, [255](#)
 Snedecor's F random variable, [1044](#)
 Somers' D

- for columns, 798
- for rows, 798
- sorting, 570, 573
 - key, 572
- sparse matrices
 - band storage format, 71
 - Cholesky factorization of symmetric positive definite, 155
 - compressed sparse column format, 74
 - conjugate gradient method, 163
 - direct methods, 67
 - general band system linear equation solution, 135
 - IMSL_SP_BDFAC, 139
 - IMSL_SP_BDPDFAC, 155
 - IMSL_SP_BDPDSOL, 152
 - IMSL_SP_BDSOL, 135
 - IMSL_SP_CG, 163
 - IMSL_SP_GMRES, 159
 - IMSL_SP_LUFAC, 128
 - IMSL_SP_LUSOL, 122
 - IMSL_SP_MVMUL, 167
 - IMSL_SP_PDFAC, 148
 - IMSL_SP_PDSOL, 143
 - introduction, 67
 - iterative methods, 68
 - linear equation solution, 122
 - LU factorization of, 128
 - LU factorization of band storage matrix, 139
 - matrix storage modes, 68
 - matrix-vector product of sparse matrix and dense vector, 167
 - positive definite system, 148
 - restarted generalized minimum residual method, 159
 - sparse coordinate storage format, 68
 - storage formats, choosing, 73
 - symmetric positive definite system, 152
 - symmetric positive definite system solution, 143
 - utilities, 68
- Spearman rank correlation, 798
- special functions, 473
 - IMSL_AIRY_AI, 526
 - IMSL_AIRY_BI, 528
 - IMSL_BESSI, 498
 - IMSL_BESSI_EXP, 506
 - IMSL_BESSJ, 500
 - IMSL_BESSK, 502
 - IMSL_BESSK_EXP, 508
 - IMSL_BESSY, 504
 - IMSL_BETA, 484
 - IMSL_BETAI, 489
 - IMSL_ELE, 512
 - IMSL_ELK, 510
 - IMSL_ELRC, 520
 - IMSL_ELRD, 516
 - IMSL_ELRF, 514
 - IMSL_ELRJ, 518
 - IMSL_ERF, 477
 - IMSL_ERFC, 480
 - IMSL_FRESNEL_COSINE, 522
 - IMSL_FRESNEL_SINE, 524
 - IMSL_GAMMAI, 495
 - IMSL_KELVIN_BEI0, 533
 - IMSL_KELVIN_BER0, 531
 - IMSL_KELVIN_KEI0, 537
 - IMSL_KELVIN_KER0, 535
 - IMSL_LNBETA, 487
 - IMSL_LNGAMMA, 491
- speed of light, 1153
- splines, 193
 - approximation
 - smooth cubic, 254
 - cubic, 193
 - evaluation, 224
 - integration
 - one- or two-dimensional, 230
 - interpolation
 - knot-sequence, 219
 - one-dimensional, 215
 - two-dimensional, 213

- least squares
 - constrained, 248
 - one- or two-dimensional, 238
- smoothing, 255
- structures for, 195
- subspace, 239
- tensor-product, 194, 211
- standard atmospheric pressure, 1153
- standard deviation, 544, 551, 552, 557, 608, 640, 650
 - exact, 797, 898, 898, 899, 900, 900
- standard errors, 801
- standard errors, for characteristic roots, 977
- standard gravity, 1154
- standard volume ideal gas, 1154
- statampere, 1155
- statcoulomb, 1155
- state vector, 957
- statespace model, 957
- stationary IMSL_ARMA, 931
- statistics in the two-way contingency table
 - IMSL_CONTINGENCY, 796
 - IMSL_EXACT_ENUM, 809
 - IMSL_EXACT_NETWORK, 812
- Stefan-Boltzman, 1154
- stepwise selection, 639
- stiff IVPs, 329
- stochastic processes
 - IMSL_RANDOM_ARMA, 1131
- stoke, 1155
- Stuart's tc, 798
- Student's t distribution, 1093
- Student's t distribution function, 1046
- Student-Newman-Keuls multiple-comparisons test, 769
- subspace, 234
- sum of squares
 - for error, 608, 640, 650
 - for the model, 608, 640, 650
 - sequential, 613, 651
 - total corrected, 608, 640, 650

- summary statistics, 592, 607
- sum-of-squares and crossproducts matrix, 723
- sums-of-squares
 - within-cluster, 973
- survival analysis
 - IMSL_SURVIVAL_GLM, 1006
- symmetric positive definite system, 89

T

- t test statistic, 552, 558, 559
- tabular data, 281
- tabulate, sort, and rank
 - IMSL_FREQTABLE, 563
 - IMSL_RANKS, 577
 - IMSL_SORTDATA, 570
- temperature, 1155
- tensor-product splines, 194, 211
- tera, 1156
- Tesla, 1155
- test for linear trend, 804
- test for normality, 882
- tests for randomness, 874
 - IMSL_RANDOMNESS_TEST, 897
- tie statistics, 855
- time, 1154
- time constraints, 330
- time series
 - autoregressive parameters, 913
 - backward differences, 930
 - Box-Jenkins forecasts, 922
 - difference, 929
 - moving average parameters, 913
- time series and forecasting
 - IMSL_ARMA, 913
 - IMSL_AUTOCORRELATION, 940
 - IMSL_BOXCOXTRANS, 935
 - IMSL_DIFFERENCE, 929
 - IMSL_GARCH, 952
 - IMSL_KALMAN, 957
 - IMSL_LACK_OF_FIT, 948

IMSL_PARTIAL_AC, 945
 trademarks, 2
 transformations, 597
 transforms
 IMSL_CONVOLID, 390
 IMSL_CORRID, 395
 IMSL_FFTCOMP, 377
 IMSL_FFTINIT, 387
 IMSL_LAPLACE_INV, 398
 triangular distribution, 1093
 triple point of water, 1154
 trust region, 670
 Tucker reliability coefficient, 984
 Tukey method, 755
 Tukey normal scores, 581
 Tukey-Kramer method, 755
 two or more samples tests - nonparametric statistics
 IMSL_COCHRANQ_TEST, 865
 IMSL_FRIEDMANS_TEST, 860
 IMSL_KTRENDS, 868
 IMSL_KW_TEST, 857
 two-dimensional iterated integrals, 312, 313

U

uncertainty
 coefficients, 798, 804
 measures of, 802
 unconstrained minimization
 IMSL_FMIN, 425
 IMSL_FMINV, 433
 IMSL_NLINLSQ, 441
 unit circle, 1084
 univariate and bivariate quadrature
 IMSL_INTFCN, 284
 univariate quadrature, 27, 283
 univariate statistics, 544, 817, 1006, 1006
 utilities

IMSL_CONSTANT, 1152
 IMSL_DATETODAYS, 1150
 IMSL_DAYSTODATE, 1148
 IMSL_MACHINE, 1158
 IMSL_MATRIX_NORM, 1171
 IMSL_NORM, 1168
 utility functions, 1145

V

Van der Waerden normal scores, 581
 variable selection, 587, 630, 639
 IMSL_ALLBEST, 630
 IMSL_STEPWISE, 639
 variables
 classification, 600
 continuous, 600
 dummy, 601
 indicator, 601
 variance-covariance matrix, 723, 1081
 variances, 544, 547, 550, 722
 asymptotic, 978
 for two normal populations, 555
 inferences about, 560
 inflation factor, 609
 lower confidence limit, 545
 normal population, 550
 upper confidence limit, 545
 variation, coefficient of, 544, 548, 608, 640, 650
 vector norms, 1168
 1-norm, 1169, 1169
 Euclidean, 1169
 infinity, 1169
 viscosity, 1155
 volt, 1155
 voltage, 1155
 volume, 1155
 von Mises distribution, 1093

W

water, triple point, [1154](#)
watt, [1155](#)
Weber, [1155](#)
Weibull distribution, [1094](#)
weighted least-squares fit, [592](#), [613](#), [616](#)
Wilcoxon rank sum test, [837](#)
Wilson-Hilferty approximation, [1039](#)
work, [1155](#)

Z

zeroes of a polynomial
 IMSL_ZEROPOLY, [410](#)
zeros of a function, [408](#)
 IMSL_ZEROFCN, [413](#)
 Muller's method, [413](#)
zeros of a polynomial, [408](#)
 Jenkins-Traub three-stage algorithm, [410](#)
zeros of a system, [418](#)