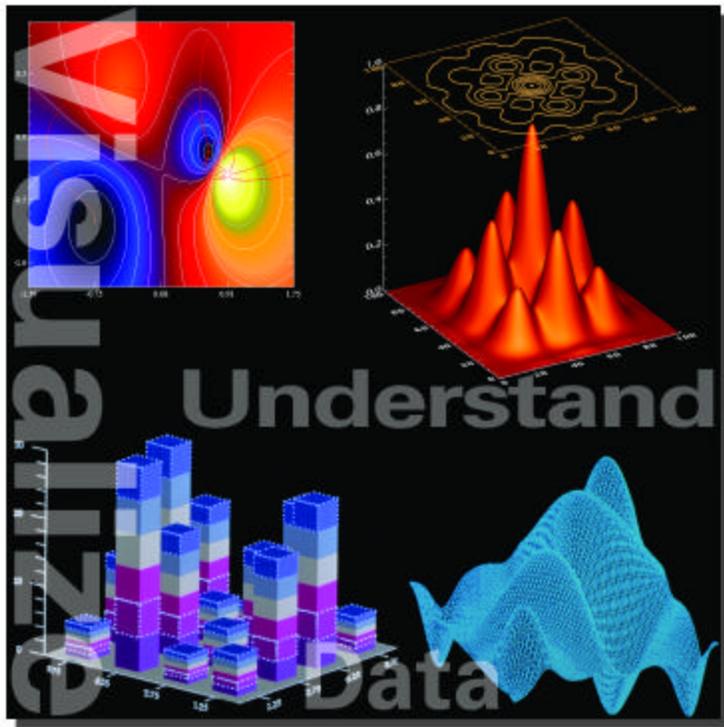




P V - W A V E 7 . 5[®]



I M S L S t a t i s t i c s R e f e r e n c e

HELPING CUSTOMERS **SOLVE** COMPLEX PROBLEMS

Visual Numerics, Inc.

Visual Numerics, Inc.
2500 Wilcrest Drive
Suite 200
Houston, Texas 77042-2579
United States of America
713-784-3131
800-222-4675
(FAX) 713-781-9260
<http://www.vni.com>
e-mail: info@boulder.vni.com

Visual Numerics, Inc.
7/F, #510, Sect. 5
Chung Hsiao E. Rd.
Taipei, Taiwan 110 ROC
+886-2-727-2255
(FAX) +886-2-727-6798
e-mail: info@vni.com.tw

Visual Numerics S.A. de C.V.
Cerrada de Berna 3, Tercer Piso
Col. Juarez
Mexico, D.F. C.P. 06600
Mexico

Visual Numerics, Inc. (France) S.A.R.L.
Tour Europe
33 place des Corolles
Cedex 07
92049 PARIS LA DEFENSE
FRANCE
+33-1-46-93-94-20
(FAX) +33-1-46-93-94-39
e-mail: info@vni-paris.fr

Visual Numerics International GmbH
Zettachring 10
D-70567 Stuttgart
GERMANY
+49-711-13287-0
(FAX) +49-711-13287-99
e-mail: info@visual-numerics.de

Visual Numerics, Inc., Korea
Rm. 801, Hanshin Bldg.
136-1, Mapo-dong, Mapo-gu
Seoul 121-050
Korea

Visual Numerics International, Ltd.
Suite 1
Centennial Court
East Hampstead Road
Bracknell, Berkshire
RG 12 1 YQ
UNITED KINGDOM
+01-344-458-700
(FAX) +01-344-458-748
e-mail: info@vniuk.co.uk

Visual Numerics Japan, Inc.
Gobancho Hikari Building, 4th Floor
14 Gobancho
Chiyoda-Ku, Tokyo, 102
JAPAN
+81-3-5211-7760
(FAX) +81-3-5211-7769
e-mail: vda-spirt@vnij.co.jp

© 1990-2001 by Visual Numerics, Inc. An unpublished work. All rights reserved. Printed in the USA.

Information contained in this documentation is subject to change without notice.

IMSL, PV-WAVE, Visual Numerics and PV-WAVE Advantage are either trademarks or registered trademarks of Visual Numerics, Inc. in the United States and other countries.

The following are trademarks or registered trademarks of their respective owners: Microsoft, Windows, Windows 95, Windows NT, Fortran PowerStation, Excel, Microsoft Access, FoxPro, Visual C, Visual C++ — Microsoft Corporation; Motif — The Open Systems Foundation, Inc.; PostScript — Adobe Systems, Inc.; UNIX — X/Open Company, Limited; X Window System, X11 — Massachusetts Institute of Technology; RISC System/6000 and IBM — International Business Machines Corporation; Java, Sun — Sun Microsystems, Inc.; HPGL and PCL — Hewlett Packard Corporation; DEC, VAX, VMS, OpenVMS — Compaq Computer Corporation; Tektronix 4510 Rasterizer — Tektronix, Inc.; IRIX, TIFF — Silicon Graphics, Inc.; ORACLE — Oracle Corporation; SPARCstation — SPARC International, licensed exclusively to Sun Microsystems, Inc.; SYBASE — Sybase, Inc.; HyperHelp — Bristol Technology, Inc.; dBase — Borland International, Inc.; MIFF — E.I. du Pont de Nemours and Company; JPEG — Independent JPEG Group; PNG — Aladdin Enterprises; XWD — X Consortium. Other product names and companies mentioned herein may be the trademarks of their respective owners.

IMPORTANT NOTICE: Use of this document is subject to the terms and conditions of a Visual Numerics Software License Agreement, including, without limitation, the Limited Warranty and Limitation of Liability. If you do not accept the terms of the license agreement, you may not use this documentation and should promptly return the product for a full refund. Do not make illegal copies of this documentation. No part of this documentation may be stored in a retrieval system, reproduced or transmitted in any form or by any means without the express written consent of Visual Numerics, unless expressly permitted by applicable law.

Table of Contents

Preface [vii](#)

Finding the Appropriate Routine [vii](#)

Documentation Organization [vii](#)

Typographical Conventions [viii](#)

Technical Support [x](#)

Introduction [xiii](#)

Starting PV-WAVE:IMSL Statistics [xiii](#)

Comparison of Matrices vs. Arrays [xiii](#)

Underflow and Overflow [xiv](#)

Missing Values [xiv](#)

User Errors [xiv](#)

Chapter 1: Basic Statistics [17](#)

Contents of Chapter [17](#)

Introduction [18](#)

SIMPLESTAT Function [19](#)

NORM1SAMP Function [25](#)

NORM2SAMP Function [29](#)

FREQTABLE Function [36](#)

SORTDATA Function [42](#)

RANKS Function [48](#)

Chapter 2: Regression [55](#)

Contents of Chapter [55](#)

Introduction [56](#)

REGRESSORS Function 70
MULTIREGRESS Function 77
MULTIPREDICT Function 93
ALLBEST Procedure 100
STEPWISE Procedure 109
POLYREGRESS Function 118
POLYPREDICT Function 125
NONLINREGRESS Function 132
HYPOTH_PARTIAL Function 141
HYPOTH_SCPH Function 147
HYPOTH_TEST Function 151
NONLINOPT Function 160
LNORMREGRESS Function 169

Chapter 3: Correlation and Covariance 189

Contents of Chapter 189
Introduction 189
COVARIANCES Function 190
PARTIAL_COV Function 194
POOLED_COV Function 199
ROBUST_COV Function 202

Chapter 4: Analysis of Variance 211

Contents of Chapter 211
Introduction 211
ANOVA1 Function 212
ANOVAFACT Function 221
MULTICOMP Function 230

ANOVANESTED Function 231

ANOVABALANCED Function 242

Chapter 5: Categorical and Discrete Data Analysis 259

Contents of Chapter 259

Introduction 259

CONTINGENCY Function 261

EXACT_ENUM Function 273

EXACT_NETWORK Function 275

CAT_GLM Function 280

Chapter 6: Nonparametric Statistics 295

Contents of Chapter 295

Introduction 295

SIGNTEST Function 296

WILCOXON Function 300

NCTRENDS Function 308

CSTRENDS Function 310

TIE_STATS Function 315

KW_TEST Function 317

FRIEDMANS_TEST Function 319

COCHRANQ Function 324

KTRENDS Function 326

Chapter 7: Goodness of Fit 333

Contents of Chapter 333

Introduction 334

CHISQTEST Function 334

NORMALITY Function 339
KOLMOGOROV1 Function 342
KOLMOGOROV2 Function 345
MVAR_NORMALITY Function 347
RANDOMNESS_TEST Function 352

Chapter 8: Time Series and Forecasting 363

Contents of Chapter 363
Introduction 364
ARMA Function 366
DIFFERENCE Function 382
BOXCOXTRANS Function 387
AUTOCORRELATION Function 391
PARTIAL_AC Function 395
LACK_OF_FIT Function 398
GARCH Function 401
KALMAN Procedure 406

Chapter 9: Multivariate Analysis 417

Contents of Chapter 417
Introduction 417
K_MEANS Function 419
PRINC_COMP Function 423
FACTOR_ANALYSIS Function 428
DISCR_ANALYSIS Procedure 437

Chapter 10: Survival Analysis 449

Contents of Chapter 449
Introduction 449

SURVIVAL_GLM Function [450](#)

Chapter 11: Probability Distribution Functions and Inverses [477](#)

Contents of Chapter [477](#)

NORMALCDF Function [478](#)

BINORMALCDF Function [480](#)

CHISQCDF Function [482](#)

FCDF Function [487](#)

TCDF Function [489](#)

GAMMACDF Function [492](#)

BETACDF Function [495](#)

BINOMIALCDF Function [497](#)

BINOMIALPDF Function [498](#)

HYPERGEOCDF Function [500](#)

POISSONCDF Function [502](#)

Chapter 12: Random Number Generation [505](#)

Contents of Chapter [505](#)

Introduction [506](#)

RANDOMOPT Procedure [510](#)

RANDOM_TABLE Procedure [514](#)

RANDOM Function [518](#)

RANDOM_NPP Function [537](#)

RANDOM_ORDER Function [540](#)

RAND_TABLE_2WAY Function [542](#)

RAND_ORTH_MAT Function [543](#)

RANDOM_SAMPLE Function [545](#)

RAND_FROM_DATA Function [547](#)

CONT_TABLE Procedure [549](#)

RAND_GEN_CONT Function [551](#)

DISCR_TABLE Function [553](#)

RAND_GEN_DISCR Function [557](#)

RANDOM_ARMA Function [560](#)

FAURE_INIT Function [564](#)

FAURE_NEXT_PT Function [567](#)

Chapter 13: Utilities [571](#)

Contents of Chapter [571](#)

MACHINE Function [573](#)

STATDATA Function [578](#)

BINOMIALCOEF Function [580](#)

BETA Function [581](#)

BETAI Function [583](#)

LNBETA Function [584](#)

GAMMA_ADV Function [585](#)

GAMMAI Function [587](#)

LNGAMMA Function [588](#)

CMAST_ERR_TRANS Function [589](#)

CMAST_ERR_STOP Function [591](#)

CMAST_ERR_PRINT Function [591](#)

Appendix A: References [A-1](#)

Appendix B: Summary of Routines [B-1](#)

Index [1](#)

Preface

PV-WAVE:IMSL Statistics is a powerful tool for mathematical, statistical, and scientific computing. This *PV-WAVE:IMSL Statistics Reference* documents the routines that support this functionality. Each function and procedure is designed for use in research as well as in technical applications.

Finding the Appropriate Routine

The *PV-WAVE:IMSL Statistics Reference* is organized into 13 chapters. Each chapter groups routines with similar computational or analytical capabilities. To locate the appropriate function for a given problem, refer to the *Contents of Chapter* subsection in the introduction to each chapter or the alphabetical *Summary of Routines* in Appendix B.

Often the quickest way to use this *PV-WAVE:IMSL Statistics Reference* is to find an example similar to your problem and mimic the example. Documented routines contain at least one example.

Documentation Organization

Each PV-WAVE Advantage routine conforms to established conventions in programming and documentation. The uniform design of these routines makes it easy to use more than one function or procedure in a given application. Also, the design consistency enables you to apply your experience with one PV-WAVE Advantage function to all other PV-WAVE Advantage functions.

This manual contains a concise description of each function and procedure. Each chapter begins with an introduction containing a *Contents of Chapter* that lists the routines discussed in that chapter and the corresponding page numbers. At least one example, including sample input and results, is provided for most routines. The documentation for each routine contains of the following information:

- **Routine Name** — procedure or function name with purpose statement
- **Usage** — calling sequence
- **Input/Output Parameters** — description of the parameters in the order of their occurrence

Input — parameter must be initialized; it is not changed by the function

Input/Output — parameter must be initialized; the routine returns output through the parameter; the parameter cannot be a constant or an expression

Output — no initialization is necessary; the routine returns output through this parameter; the parameter cannot be a constant or an expression

- **Returned Value** — value returned by the function
- **Keywords** — description of keywords available for a particular routine
- **Discussion** — discussion of the algorithm and references to detailed information
- **Examples** — one or more examples showing applications of this routine using the required parameters
- **Errors** — list of errors that may occur with a particular routine for which a user-defined action may be desired

References

References are listed alphabetically by author in Appendix A, *References*.

Typographical Conventions

The following typographical conventions are used in this guide:

- PV-WAVE Advantage code examples appear in a typewriter font. For example:

```
PLOT, temp, s02, Title = 'Air Quality'
```

- Comments for commands and program examples are shown in the following manner:

```
PLOT, temp, s02, Title = 'Air Quality'
; This is a comment for the PLOT command.
```

Comments are used often in this manual to explain code fragments and examples.

- PV-WAVE Advantage commands are not case sensitive. However, in this manual, variables are shown in lowercase italics (*myvar*), function and procedure names are shown in uppercase (XYOUTS), keywords are shown in mixed case italic (*XTitle*), and system variables are shown in regular mixed case type (!Version).
- A \$ at the end of a PV-WAVE Advantage line indicates that the current statement is continued on the following line.

This means, for instance, that *strings* are never split onto two lines without the addition of the string concatenation operator (+) or a comma in some cases. For example, the following lines would produce an error if entered literally in PV-WAVE Advantage:

```
WAVE> PLOT, x, y, Title = 'Average $
Air Temperatures by Two-Hour Periods'
; Note that the string is split onto two lines. This syntax would
; produce an error.
```

The correct way to enter these lines is:

```
WAVE> PLOT, x, y, Title = 'Average ' + $
'Air Temperatures by Two-Hour Periods'
; This is the correct way to split a string onto two command
; lines.
```

The string concatenation symbol (+) is used at the end of the first line, and the split portions of the string are enclosed by delimiters. This is the convention used in this reference whenever a string spans two lines. This is still only one command, even though multiple lines are used.

- Reserved words, such as FOR, IF, and CASE, are always shown in capital letters.

Technical Support

If you have problems installing, unlocking, or running your software, contact Visual Numerics Technical Support by calling:

Office Location	Phone Number
Corporate Headquarters Houston, Texas	713-784-3131
Boulder, Colorado	303-939-8920
France	+33-1-46-93-94-20
Germany	+49-711-13287-0
Japan	+81-3-5211-7760
Korea	+82-2-3273-2633
Mexico	+52-5-514-9730
Taiwan	+886-2-727-2255
United Kingdom	+44-1-344-458-700

Users outside the U.S., France, Germany, Japan, Korea, Mexico, Taiwan, and the U.K. can contact their local agents.

Please be prepared to provide the following information when you call for consultation during Visual Numerics business hours:

- Your license number, a six-digit number that can be found on the packing slip accompanying this order. (If you are evaluating the software, just mention that you are from an evaluation site.)
- The name and version number of the product. For example, PV-WAVE 7.0.
- The type of system on which the software is being run. For example, SPARCstation, IBM RS/6000, HP 9000 Series 700.
- The operating system and version number. For example, HP-UX 10.2 or IRIX 6.5.
- A detailed description of the problem.

FAX and E-mail Inquiries

Contact Visual Numerics Technical Support staff by sending a FAX to:

Office Location	FAX Number
Corporate Headquarters	713-781-9260
Boulder, Colorado	303-245-5301
France	+33-1-46-93-94-39
Germany	+49-711-13287-99
Japan	+81-3-5211-7769
Korea	+82-2-3273-2634
Mexico	+52-5-514-4873
Taiwan	+886-2-727-6798
United Kingdom	+44-1-344-458-748

or by sending E-mail to:

Office Location	E-mail Address
Boulder, Colorado	support@boulder.vni.com
France	support@vni-paris.fr
Germany	support@visual-numeric.de
Japan	vda-sprt@vnij.co.jp
Korea	support@vni.co.kr
Taiwan	support@vni.com.tw
United Kingdom	support@vniuk.co.uk

Electronic Services

Service	Address
General e-mail	<code>info@boulder.vni.com</code>
Support e-mail	<code>support@boulder.vni.com</code>
World Wide Web	<code>http://www.vni.com</code>
Anonymous FTP	<code>ftp.boulder.vni.com</code>
FTP Using URL	<code>ftp://ftp.boulder.vni.com/VNI/</code>
PV-WAVE Mailing List:	<code>Majordomo@boulder.vni.com</code>
To subscribe include:	<code>subscribe pv-wave YourEmailAddress</code>
To post messages	<code>pv-wave@boulder.vni.com</code>

Introduction

Starting PV-WAVE:IMSL Statistics

To start PV-WAVE:IMSL Statistics, you must first be running PV-WAVE. For detailed information on starting PV-WAVE, see the *PV-WAVE User's Guide* or the installation instructions.

At the WAVE> prompt, type:

```
@stat_startup
```

You will then see this message:

```
PV-WAVE:IMSL Statistics is initialized.
```

You are now ready to use PV-WAVE:IMSL Statistics.

Comparison of Matrices vs. Arrays

In this book, we use the following convention for 2D 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 command makes this easy to visualize:

```
a = INTARR( 4, 8 )      &      a(2,5) = 1      &      PM, a
  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0
  0  0  0  0  0  1  0  0
```

Underflow and Overflow

In most cases, PV-WAVE:IMSL Statistics 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.

PV-WAVE:IMSL Statistics 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 of the routines in this manual allow the data to contain missing values. These routines recognize as a missing value the special value referred to as “Not a Number” or NaN. The actual value varies on different computers, but it can be obtained by reference to function MACHINE.

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

User Errors

PV-WAVE:IMSL Statistics functions attempt to detect user errors and handle them in a way that provides as much information to the user as possible. To do this, five levels of *Informational Error* severity, in addition to the basic PV-WAVE:IMSL Statistics error-handling facility, are recognized. Following a call to a PV-WAVE:IMSL Statistics mathematical or statistical function, the system variables !Error and !Cmast_Err contain information concerning the current error state. The system variable !Error contains the error *number* of the last error. System variable !Cmast_Err is set to either zero, which indicates that an *Informational Error* did not occur, or to the error code of the last *Informational Error* that did occur.

Errors and Default Actions

When your application returns from a PV-WAVE:IMSL Statistics function, the system variable !Cmast_Err is set either to zero, which indicates that an informational error did not occur, or to the error code for the last *Informational Error* that did occur. Internally, there are five levels of *Informational Error* severity: note, alert, warning, fatal, and terminal. Although PV-WAVE:IMSL Statistics does not allow users to directly manipulate how these errors are interpreted internally, some control over the output of error messages is allowed. All informational error messages are printed by default. Setting the system variable !Quiet to a nonzero value suppresses output of notes, alerts, and warnings.

The system variable !Error remains active during all PV-WAVE:IMSL Statistics error states. But when an *Informational Error* occurs within a mathematical function, the system variable !Cmast_Err is used.

What Determines Error Severity

Although your input(s) may be mathematically correct, limitations of the computer's arithmetic and the algorithm itself can make it impossible to accurately compute an answer. In this case, the assessed degree of accuracy determines the severity of the error. In instances where the function computes several output quantities and some are not computable, an error condition exists. Its severity depends on an assessment of the overall impact of the error.

Functions for Error Handling

With respect to *Informational Errors*, you can interact with the PV-WAVE:IMSL Statistics error-handling system in two ways:

(1) change the default printing actions and (2) determine the code of an *Informational Error* in order to take corrective action. To change the default printing action, set the system variable !Quiet to a nonzero value. Use CMAST_ERR_TRANS to retrieve the integer code for an informational error.

Use of CMAST_ERR_TRANS to Determine Program Action

In the program segment below, the Cholesky factorization of a matrix is to be performed. If it is determined that the matrix is not nonnegative definite (and often this is not immediately obvious), the program takes a different branch:

```
x = CHNDFAC, a, fac
    ; Call CHNDFAC with a matrix that may not be nonnegative definite.
IF (CMAST_ERROR_TRANS($
    'MATH_NOT_NONNEG_DEFINITE') eq !Cmast_Err)) THEN ...
    ; Check the system variable !Cmast_Err to see if it contains the
    ; error code for the error NOT_NONNEG_DEFINITE.
```

Basic Statistics

Contents of Chapter

Simple Summary Statistics

Univariate summary statistics	SIMPLESTAT Function
Mean and variance inference for a single normal population	NORM1SAMP Function
Inferences for two normal populations	NORM2SAMP Function

Tabulate, Sort, and Rank

Tallies observations into a one-way frequency table	FREQTABLE Function
Sorts data with options to tally cases into a multiway frequency table	SORTDATA Function
Ranks, normal scores, or exponential scores	RANKS Function

Introduction

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 by using function 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.

SIMPLESTAT Function

Computes basic univariate statistics.

Usage

result = SIMPLESTAT(*x*)

Input Parameters

x — Data matrix. The data value for the *i*-th observation of the *j*-th variable should be in the matrix element (*i, j*).

Returned Value

result — 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.

i	Statistic Returned in Element (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.
9	number of observations (the counts)
10	lower confidence limit for the mean (assuming normality) The default is a 95-percent confidence interval.

i	Statistic Returned in Element (i, *)
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)

Input Keywords

Double — If present and nonzero, double precision is used.

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 $\text{Conf_Means} = 100.0 - 2.0(100.0 - c)$ (at least 50 percent).

Default: 95-percent confidence interval is computed

Conf_Variiances — 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 a one-sided confidence interval with confidence level c , set $\text{Conf_Means} = 100.0 - 2.0(100.0 - c)$ (at least 50 percent).

Default: 95-percent confidence interval is computed.

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.

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

Weights — One-dimensional array containing the weight for each observation.

Default: each observation has a weight of 1

Discussion

Function SIMPLESTAT 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

$$\text{MAD} / \Phi^{-1}(3/4)$$

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 = STATDATA(5)
  stats = SIMPLESTAT(x)
        ; Call 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

NORM1SAMP Function

Computes statistics for mean and variance inferences using a sample from a normal population.

Usage

result = NORM1SAMP(*x*)

Input Parameters

x — One-dimensional array containing the observed values.

Returned Value

result — The mean of the sample.

Input Keywords

Double — If present and nonzero, double precision is used

Conf_Mean — Confidence level (in percent) for two-sided interval estimate of the mean. 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

T_Null_Hyp — Null hypothesis value for *t* test for the mean.

Default: $T_Null_Hyp = 0.0$

Chi_Sq_Null_Hyp — Null hypothesis value for the chi-squared test for the variance.

Default: $Chi_Sq_Null_Hyp = 1.0$

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.

Output Keywords

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*.

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*.

Stdev — Named variable into which the standard deviation of the sample is stored.

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.

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 = ", 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.

```
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 = 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 =      2.00000
```

NORM2SAMP Function

Computes statistics for mean and variance inferences using samples from two independently normal populations.

Usage

result = NORM2SAMP(*x1*, *x2*)

Input Parameters

x1 — One-dimensional array containing the first sample.

x2 — One-dimensional array containing the second sample.

Returned Value

result — Difference in means of the mean of the second sample from the first sample.

Input Keywords

Double — If present and nonzero, double precision is used.

Conf_Mean — Confidence level for two-sided interval estimate of the mean of *x1* minus the mean of *x2*, in percent. 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: *Conf_Mean* = 95.0

T_Test_Null_Hyp — Null hypothesis value for the *t* test.

Default: *T_Test_Null_Hyp* = 0.0

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 com-

puted and is returned if *Ci_Ratio_Var* is specified. The confidence intervals are symmetric in probability.

Default: *Conf_Var* = 95.0

Chi_Sq_Null_Hyp — Null hypothesis value for the chi-squared test.

Default: *Chi_Sq_Null_Hyp* = 1.0

Output Keywords

Mean_X1 — Means of the first sample.

Mean_X2 — Means of the second sample.

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.

T_Test_Eq_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 of *T_Test_Null_Hyp*.) The first element contains the degrees of freedom, 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 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 of *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.

Pooled_Var — Named variable into which the pooled variance for the two samples is stored.

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.

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 description of *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.

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.

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.

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.

Discussion

Function NORM2SAMP 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 function NORM1SAMP on page 25.

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:

$$\bar{x}_1 = \left(\sum x_{1i} / n_1 \right), \quad \bar{x}_2 = \left(\sum x_{2i} \right) / n_2$$

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. (See, for example, 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

$$t' = (\bar{x}_1 - \bar{x}_2 - d) / s_d \quad ,$$

$$\text{where } s_d = \sqrt{(s_1^2/n_1) + (s_2^2/n_2)} \quad .$$

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.

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 below.

Scores for Standard Group	Scores for Experimental Group
72	111
75	118
77	128
80	138
104	140
110	150
125	163
	164
	169

```

x1 = [72, 75, 77, 80, 104, 110, 125]
  x2 = [111, 118, 128, 138, 140, 150, 163, $
        164, 169]
PRINT, "difference of means = ", 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 = 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

FREQTABLE Function

Tallies observations into a one-way or two-way frequency table.

Usage

result = FREQTABLE(*x*, *nxbins* [, *y*, *nybins*])

Input Parameters

x — One-dimensional array containing the observations for the first variable.

nxbins — Number of intervals (bins) for *x*.

y — (Optional) One-dimensional array containing the observations for the second variable.

nybins — (Optional) Number of intervals (bins) for *y*.

Returned Value

result — One-dimensional or two-dimensional array containing the counts.

Input Keywords

Double — If present and nonzero, double precision is used.

IF Two Positional Arguments Are Used:

Lower_Bound — Used with *Upper_Bound* to specify two semi-infinite intervals that 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 keyword *Upper_Bound* also must be specified with this keyword. Parameter *nxbins* must be greater than or equal to 3 for this option.

Upper_Bound — Used along with *Lower_Bound* to specify two semi-infinite intervals that 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 keyword *Lower_Bound* must also be specified with this keyword. Parameter *nxbins* must be greater than or equal to 3 for this option.

Cutpoints — 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. Parameter *nxbins* must be greater than 3 for this option. If *Cutpoints* is used, then no other keywords should be specified.

Class_Marks — 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 $(Class_Marks(1) - Class_Marks(0))$. Parameter *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:

Lower_Bound — Used with *Upper_Bound* to specify intervals of equal lengths. See the *Discussion* section for details.

Upper_Bound — Used with *Lower_Bound* to specify intervals of equal lengths. See the *Discussion* section for details.

Cutpoints — Specifies a one-dimensional array of cutpoints (boundaries). 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.

Class_Marks — 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.

Discussion

If Two Positional Arguments Are Used:

The default action of `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. Parameter $nxbins$ must be greater than or equal to 3 for this option.

If 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 mid-points of each of the $nxbins$, and each interval is taken to have the following length:

$$(Class_Marks(1) - Class_Marks(0))$$

Parameter $nxbins$ must be greater than or equal to 2 for this option.

If 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. Parameter $nxbins$ 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 the user 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.

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 = FREQTABLE(x, 10)
     ; Call 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 FREQTABLE
```

```
table = FREQTABLE(x, nxbins, y, nybins)
PM, table, Format = "(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 = FREQTABLE(x, nxbins, y, nybins, Upper_Bound = up, $
                 Lower_Bound = lb)
PM, table, Format = "(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 = 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 = 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

```

SORTDATA Function

Sorts observations by specified keys, with option to tally cases into a multiway frequency table.

Usage

result = SORTDATA(*x*, *n_keys*)

Input Parameters

x — One- or two-dimensional array containing the observations to be sorted.

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 keyword *Indices_Keys*).

Returned Value

result — The sorted array.

Input Keywords

Double — If present and nonzero, double precision is used.

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

Frequencies — One-dimensional array containing the frequency for each observation in *x*.

Default: *Frequencies* (*) = 1

Ascending — If present and nonzero, the sort is in ascending order. (Default) Keywords *Ascending* and *Descending* cannot be used together.

Descending — If present and nonzero, the sort is in descending order. Keywords *Ascending* and *Descending* cannot be used together.

Output Keywords

Permutation — Named variable into which a one-dimensional array containing the rearrangement (permutation) of the observations (rows) is stored.

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. 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. Keywords *Table_N*, *Table_Values*, and *Table_Bal* must be used together.

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. Keywords *Table_N*, *Table_Values*, and *Table_Bal* must be used together.

N_List_Cells — Named variable into which the number of nonempty cells is stored. Keywords *N_List_Cells*, *List_Cells*, and *Table_Unbal* must be used together.

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. Keywords *N_List_Cells*, *List_Cells*, and *Table_Unbal* 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. 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 $N_ELEMENTS(N_Cells)$.

Discussion

Function SORTDATA can perform both a key sort and/or tabulation of frequencies into a multiway frequency table.

Sorting

Function SORTDATA 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

Function SORTDATA 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 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 continu-

ous can be grouped first. The function FREQTABLE can be used to group variables and determine the frequencies of groups.

When *Table_N*, *Table_Values*, and *Table_Bal* are specified, 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.

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 = 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, 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 = 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      1.00000
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	1.00000	8.00000
2.00000	2.00000	2.00000	9.00000
1.00000	1.00000	1.00000	9.00000

```
y = SORTDATA(x, 2, Permutation = permutation)
```

```
; Use 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

RANKS Function

Computes the ranks, normal scores, or exponential scores for a vector of observations.

Usage

result = RANKS(*x*)

Input Parameters

x — One-dimensional array containing the observations to be ranked.

Returned Value

result — A one-dimensional array containing the rank (or optionally, a transformation of the rank) of each observation.

Input Keywords

Double — If present and nonzero, double precision is used.

Average_Tie, or

Highest, or

Lowest, or

Random_Split — At most, one of these keywords can be set to a nonzero value to change the method used to assign a score to tied observations.

Keyword	Method
<i>Average_Tie</i>	average of the scores of the tied observations (default)
<i>Highest</i>	highest score in the group of ties
<i>Lowest</i>	lowest score in the group of ties

Keyword	Method
<i>Random_Split</i>	tied observations are randomly split using a random-number generator

Fuzz — Value used to determine when two items are tied. If $ABS(x(I) - x(J))$ is less than or equal to *Fuzz*, then $x(I)$ and $x(J)$ are said to be tied.

Default: *Fuzz* = 0.0

Ranks, or

Blom_Scores, or

Tukey_Scores, or

Vdw_Scores, or

Exp_Norm_Scores, or

Savage_Scores — At most, one of these keywords can be set to a nonzero value to specify the type of values returned.

Keyword	Result
<i>Ranks</i>	ranks (default)
<i>Blom_Scores</i>	Blom version of normal scores
<i>Tukey_Scores</i>	Tukey version of normal scores
<i>Vdw_Scores</i>	Van der Waerden version of normal scores
<i>Exp_Norm_Scores</i>	expected value of normal order statistics (for tied observations, the average of the expected normal scores)
<i>Savage_Scores</i>	Savage scores (expected value of exponential order statistics)

Discussion

Ties

If the assignment $RANK = RANKS(x)$ 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 $RANK(i) = 1$. If both $x(i)$ and $x(j)$ have

the same smallest value, then the output value depends on the option used to break ties.

Keyword	Result
<i>Average_Tie</i>	$result(i) = result(j) = 1.5$
<i>Highest</i>	$result(i) = result(j) = 2.0$
<i>Lowest</i>	$result(i) = result(j) = 1.0$
<i>Random_Split</i>	$result(i) = 1.0$ and $result(j) = 2.0$ or, randomly, $result(i) = 2.0$ and $result(j) = 1.0$

When the ties are resolved randomly, function 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 the function 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, NORMALCDF (with 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 $n = \text{N_ELEMENTS}(x)$. 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 = RANKS(x)
    ; Call 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

Regression

Contents of Chapter

Multiple Linear Regression

Generates regressors for a
general linear model..... [REGRESSORS Function](#)

Fits a multiple linear regression
model and optionally produces
summary statistics
for a regression model [MULTIREGRESS Function](#)

Computes predicted values,
confidence intervals,
and diagnostics [MULTIPREDICT Function](#)

Variable Selection

All best regressions [ALLBEST Procedure](#)

Stepwise regression [STEPWISE Procedure](#)

Polynomial and Nonlinear Regression

Fits a polynomial
regression model..... [POLYREGRESS Function](#)

Computes predicted values,
confidence intervals,
and diagnostics [POLYPREDICT Function](#)

Fits a nonlinear regression model. [NONLINREGRESS Function](#)

Multivariate Linear Regression—Statistical Inference and Diagnostics

Construction of a completely testable hypothesis [HYPOTH_PARTIAL Function](#)

Sums of cross products for a multivariate hypothesis [HYPOTH_SCPH Function](#)

Tests for the multivariate linear hypothesis [HYPOTH_TEST Function](#)

Polynomial and Nonlinear Regression

Fit a nonlinear regression model using Powell's algorithm [NONLINOPT Function](#)

Alternatives to Least Squares Regression

LAV, Lpnorm, and LMV criteria regression [LNORMREGRESS Function](#)

Introduction

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 .

Function MULTIREGRESS (page 77) 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 MULTIREGRESS, function MULTIPREDICT (page 93) computes predicted values, confidence intervals, and case statistics for the fitted model. The information about the fit is communicated from 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 ALLBEST (page 100), which computes all best-subset regressions, or by STEPWISE (page 109), 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

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \dots + \beta_k x_i^k + \varepsilon_i \quad i = 1, 2, \dots, n$$

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; $\beta_0, \beta_1, \dots, \beta_k$ are the regression coefficients; and the ε_i 's are independently distributed normal errors each with mean zero and variance σ^2 .

Function POLYREGRESS (page 118) fits a polynomial regression model with the option of determining the degree of the model and also produces summary information. Function POLYPREDICT (page 125) computes predicted values, confidence intervals, and case statistics for the model fit by POLYREGRESS.

The information about the fit is communicated from POLYREGRESS to 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:

1. a single continuous variable
2. a single classification variable
3. several different classification variables
4. several continuous variables, some of which may be the same
5. 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 `REGRESSORS` (page 70) to specify all regressors except the intercept. The second stage calls `MULTIREGRESS` (page 77), 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 follows:

Option	<i>intcep</i>	Action
No intercept	0	An intercept is not in the model.
Intercept (default)	1	An intercept is in the model.

The remaining parameters and keywords (*n_continuous*, *n_class*, *Class_Columns*, *Var_Effects*, and *Indices_Effects*) are defined for function `REGRESSORS`. All have defaults except for *n_continuous* and *n_class*, both of which must be specified. (See the documentation for `REGRESSORS` on page 70 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 follows:

Column 0	Column 1	Column 2	Column 3
11.23	1.23	1.0	5.0

Column 0	Column 1	Column 2	Column 3
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
4.89	9.31	2.0	1.0
9.12	3.71	2.0	1.0

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 follows:

Regression Functions	<i>intcep</i>	<i>n_class</i>	<i>Class_</i> <i>Columns</i>	<i>Var_</i> <i>Effects</i>	<i>Indices_</i> <i>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

Functions for Fitting the Model

Function MULTIREGRESS (page 77) fits a multiple general linear model, where regressors for the general linear model have been generated using function REGRESSORS (page 70).

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, $R_j^2_{(1, 2, \dots, j-1)}$ 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 the user 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 function 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 .

Function `NONLINREGRESS` (page 132) 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

Function MULTIREGRESS (page 77) 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. Function POLYREGRESS (page 118) 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 $\text{rank}(X) = r$, 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 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 the following table (here, *Aov* replaces *Anova_Table*):

Model Analysis of Variance Table

Source of Variation	Degrees of Freedom	Sum of Squares	Mean Square	F	p-value
Regression	DFR = $Aov(0)$	SSR = $Aov(3)$	MSR = $Aov(6)$	$Aov(8)$	$Aov(9)$
Error	DFE = $Aov(1)$	SSE = $Aov(4)$	$s^2 = Aov(7)$		
Total	DFT = $Aov(2)$	SST = $Aov(5)$			

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:

$$SST = \sum^n w_i (y_i - \bar{y})^2$$

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:

$$SST = \sum^n w_i y_i^2$$

The error sum of squares is given as follows:

$$SSE = \sum^n w_i (y_i - \hat{y}_i)^2$$

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:

$$R^2 = 100(\text{SSR}/\text{SST}) = 100(1 - \text{SSE}/\text{SST})$$

$$R_a^2 = 100\left(1 - \frac{s^2}{\text{SST}/\text{DFT}}\right)$$

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 ($\text{CV} = \text{Anova_Table}(14)$) is expressed as a percentage and defined by

$$\text{CV} = 100s/\bar{y}.$$

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 function POLYREGRESS (page 118). 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: MULTIPREDICT (page 93) for linear and nonlinear regressions and POLYPREDICT (page 125) 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 $\text{rank}(X) = r$, 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

$$h_i = [x_i^T (X^T W X)^- x_i] w_i$$

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

$$t_i = e_i \sqrt{\frac{w_i}{s_i^2(1 - h_i)}}$$

and t_i follows a t distribution with $n - r - 1$ degrees of freedom.

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:

$$D_i = \frac{w_i h_i e_i^2}{r s^2 (1 - h_i)^2}$$

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. Function `LNORMREGRESS` 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* (page 170) 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* (page 169). 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 the user 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* (page 170). 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 function `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.

REGRESSORS Function

Generates regressors for a general linear model.

Usage

result = REGRESSORS(*x*, *n_class*, *n_continuous*)

Input Parameters

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.

Returned Value

result — A two-dimensional array containing the regressor variables generated from *x*.

Input Keywords

Double — If present and nonzero, double precision is used.

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]

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.

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.

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:

Dummy_Method	Method
(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.

Discussion

Function REGRESSORS 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, 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

$$A_1, A_2, \dots, A_{m_1} \text{ and } B_1, B_2, \dots, B_{m_2}.$$

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 REGRESSORS are given below.

<i>Dummy_Method</i>	Regressors
(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, 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 corre-

sponds 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

$$\text{NEF} = n_{\text{class}} + 2 * n_{\text{continuous}} + \begin{pmatrix} \text{NVAR} \\ 2 \end{pmatrix}$$

effects, where $\text{NVAR} = n_{\text{continuous}} + n_{\text{class}}$. 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., $x(\text{NVAR} - 1)$). The next $n_{\text{continuous}}$ effects correspond to squares of the continuous variables. The last

$$\begin{pmatrix} \text{NVAR} \\ 2 \end{pmatrix}$$

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*.

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 = REGRESSORS(x, 2, 0)
; Call 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, 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

$$\mu + \alpha_i + \beta_j + \gamma_{ij} + \delta x_{ij} + \zeta_i x_{ij} + \eta_j x_{ij} + \theta_{ij} x_{ij} \quad i = 1, 2; \quad j = 1, 2, 3$$

$$\text{where } \alpha_2 = \beta_3 = \gamma_{21} = \gamma_{22} = \gamma_{23} = \zeta_2 = \eta_3 = \theta_{21} = \theta_{22} = \theta_{23} = 0 .$$

```

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 = REGRESSORS(x, 2, 1, Dummy_Method = 1, $
      Var_Effects = var_effects, $
      Indices_Effects = indices_effects)
      ; Call 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	1.00	0.00	0.00	0.00	3.33
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

MULTIREGRESS Function

Fits a multiple linear regression model using least squares and optionally compute summary statistics for the regression model.

Usage

result = MULTIREGRESS(*x*, *y*)

Input Parameters

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) variables(s). The *i*-th column of *y* contains the *i*-th dependent variable.

Returned Value

result — If keyword *No_Intercept* is not used, 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.

Input Keywords

Double — If present and nonzero, double precision is used.

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.

Tolerance — Tolerance used in determining linear dependence. For MULTIGRESS, $Tolerance = 100 \times \epsilon$, where ϵ is machine precision, is the default choice.

Frequencies — One-dimensional array containing the frequency for each observation.

Default: $Frequencies(*) = 1$

Weights — One-dimensional array containing the weight for each observation.

Default: $Weights(*) = 1$

Predict_Info — Named variable into which the one-dimensional byte array containing information needed by MULTIPREDICT (page 93) 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 MULTIPREDICT.

Output Keywords

Rank — Named variable into which the rank of the fitted model is stored.

Coef_Covariances — Named variable into which the $m \times m \times n_{dependent}$ array containing the estimated variances and covariances of the estimated regression coefficients is stored. Here, m is the number of regression coefficients in the model. If *No_Intercept* is specified, $m = N_ELEMENTS(x(0, *))$; otherwise, $m = (N_ELEMENTS(x(0, *)) + 1)$.

XMean — Named variable into which the array containing the estimated means of the independent variables is stored.

Residual — Named variable into which the array containing the residuals is stored.

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 given as follows:

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)

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:

Column	Description
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

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.

Discussion

Function MULTIREGRESS fits a multiple linear regression model with or without an intercept.

By default, the multiple linear regression model is

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_k x_{ik} + \varepsilon_i \quad i = 0, 2, \dots, n$$

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 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.

Function MULTIREGRESS 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, 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 (uncen-

tered) 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

$$\sqrt{1 - R_{i-1, 2, \dots, i-1}^2}$$

is less than or equal to *Tolerance*. Here, $R_{i-1, 2, \dots, i-1}$ 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.

Function MULTIREGRESS also can be used to compute summary statistics from a fitted general linear model. The model 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 each independently distributed with mean zero and variance σ^2 . Function MULTIREGRESS 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, MULTIREGRESS produces summary statistics from the weighted least-squares fit. More generally, if the variance-covariance matrix of ε is σ^2V , MULTIREGRESS can be used to produce summary statistics from the generalized least-squares fit. Function MULTIREGRESS 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

$$\left(R\hat{\beta} \right)^2.$$

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.,

$$\left(R\hat{\beta} \right)^2$$

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^2GG^T . (G is a g_3 inverse of R , represented by

$$R^{g_3},$$

the result $R^{g_3}R^{g_3T}$

is a symmetric g_2 inverse of $R^TR = X^TX$. See Sallas and Lionti 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^TR 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 $Coef_Covariances = (SSE/DFE) (R^T R)^{-1}$.

Otherwise, variances and covariances of estimable functions of the regression coefficients can be obtained using $Coef_Covariances$ and $Coef_Covariances = (SSE/DFE) (GDG^T)$. 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 .

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 MULTIREGRESS to compute the coefficients.
```

```
coefs = 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 = 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 MULTIREGRESS to fit data with both simple linear regression and second order regression. The results are plotted along with confidence bands and residual plots.

```
PRO 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 = MULTIREGRESS(x, y, $
    Predict_Info = predict_info)
y_hat = MULTIPREDICT(predict_info, x, Residual = residual, Y =
    y)
y_hat = 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 = MULTIREGRESS(x2, y, Predict_Info = predict_info)
y_hat = MULTIPREDICT(predict_info, x2, Residual = residual, Y =
    y)
y_hat = MULTIPREDICT(predict_info, $
    [[z], [z * z]], Ci_Ptw_New_Sample = ci)
```

```

PLOT, x, y, Title = '2nd order regression', $
      Psym = 4, XRange = [0.0, 6.0]
      ; Plot the second-order regression.

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', $
      XRange = [0.0, 6.0], YRange = [-6, 6]
      ; Plot the residual.

OPLOT, z, line
END

```

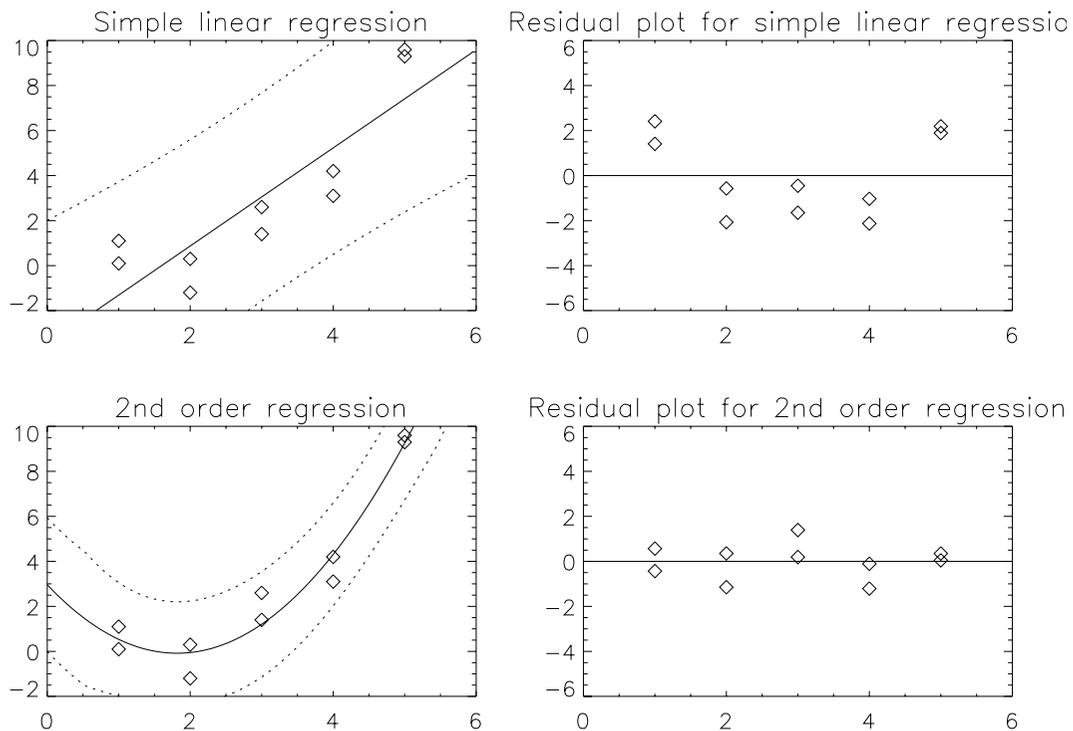


Figure 2-1 Plots of fit with confidence bands and residuals.

Example 4: Two-variable, Second-degree Fit

In this example, MULTIREGRESS is used to compute a two variable second-degree fit to data.

```
PRO 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 = 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
    END
END

!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

```

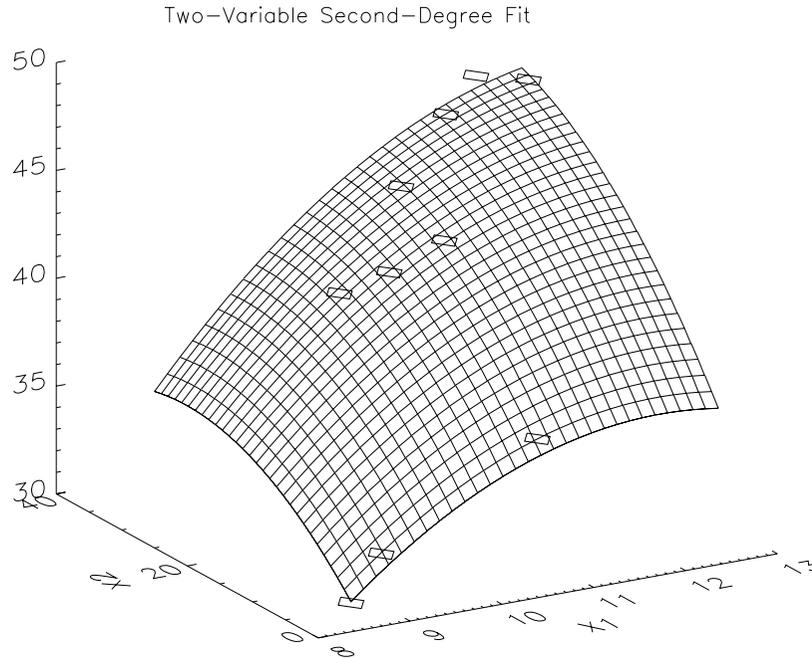


Figure 2-2 Two-variable, second-degree fit.

Warning Errors

STAT_RANK_DEFICIENT — Model is not full rank. There is not a unique least-squares solution.

MULTIPREDICT Function

Computes predicted values, confidence intervals, and diagnostics after fitting a regression model.

Usage

result = MULTIPREDICT(*predict_info*, *x*)

Input Parameters

predict_info — One-dimensional byte array containing information computed by MULTIREGRESS (page 77) 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 MULTIREGRESS.

x — Two-dimensional array containing the combinations of independent variables in each row for which calculations are to be performed.

Returned Value

result — One-dimensional array of length N_ELEMENTS (*x*(*, 0)) containing the predicted values.

Input Keywords

Double — If present and nonzero, double precision is used.

Weights — One-dimensional array containing the weight for each row of *x*. The computed prediction interval uses $SSE / (DFE * Weights(1))$ for the estimated variance of a future response.

Default: *Weights* (*) = 1

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

Y — Array of length N_ELEMENTS (*x*(*, 0)) containing the observed responses.

Output 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.

Leverage — Named variable into which the one-dimensional array of length N_ELEMENTS (x^* , 0) containing the leverages is stored.

NOTE Y must be specified if any of the following keywords are specified:

Residual — Named variable into which the one-dimensional array of length N_ELEMENTS (x^* , 0) containing the residuals is stored.

Std_Residual — Named variable into which the one-dimensional array of length N_ELEMENTS (x^* , 0) containing the standardized residuals is stored.

Del_Residual — Named variable into which the one-dimensional array of length N_ELEMENTS (x^* , 0) containing the deleted residuals is stored.

Cooks_D — Named variable into which the one-dimensional array of length N_ELEMENTS (x^* , 0) containing the Cook's D statistics is stored.

Dffits — Named variable into which the one-dimensional array of length N_ELEMENTS (x^* , 0) containing the DFFITS statistics is stored.

Discussion

The general linear model used by function 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, function 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:

$$h_i s^2 / w_i, \text{ where } s^2 = \text{SSE} / \text{DFE}$$

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 .

Example 1

This example calls MULTIPREDICT to compute predicted values after calling 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 = MULTIREGRESS(x, y, $
    Predict_Info = predict_info)
    ; Call MULTIREGRESS to compute the fit.

predicted = MULTIPREDICT(predict_info, x)
    ; Call 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 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 = MULTIREGRESS(x, y, $
    Anova_Table      = anova_table, $
    T_Tests          = t_tests,      $
    Predict_Info     = predict_info, $
    Residual         = residual)
    ; Call MULTIREGRESS to compute the fit.

predicted = MULTIPREDICT(predict_info, x, $
    Ci_scheffe       = ci_scheffe, $
    Y                = y,           $
    Dffits           = dffits)
    ; Call MULTIPREDICT.

print_results, anova_table, t_tests, y, $
    predicted, ci_scheffe, residual, dffits
    ; Print the results.

    * * 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

```

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.

ALLBEST Procedure

Selects the best multiple linear regression models.

Usage

ALLBEST, x , y

Input Parameters

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.

Input Keywords

Double — If present and nonzero, double precision is used.

Weights — One-dimensional array of length N_ELEMENTS ($x(*, 0)$) containing the weight for each row of x .

Default: $Weights(*) = 1$

Frequencies — One-dimensional array of length N_ELEMENTS ($x(*, 0)$) containing the frequency for each row of x .

Default: $Frequencies(*) = 1$

Max_Subset — The R^2 criterion is used, where subset sizes 1, 2, ..., Max_Subset are examined. This option is the default with $Max_Subset = N_ELEMENTS(x(0, *))$. Keywords Max_Subset , $Adj_R_Squared$, and $Mallows_Cp$ cannot be used together.

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.

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: *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: *Max_N_Good* = 10

Cov_Nobs — Number of observations associated with array *Cov_Input*. Keywords *Cov_Input* and *Cov_Nobs* 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 function COVARIANCES. Parameters x and y , and keywords *Frequencies* and *Weights* are not accessed when this option is specified. Normally, ALLBEST computes *Cov_Input* from the input data matrices x and y . However, there may be cases when the user wants to calculate the covariance matrix and manipulate it before calling ALLBEST. See the *Discussion* section for a discussion of such cases.

Keywords *Cov_Input* and *Cov_Nobs* must be used together.

Output Keywords

Idx_Criteria — Named variable into which the one-dimensional array of length NSIZE containing the locations in *Criteria* 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 $i = 0, 1, \dots, \text{NSIZE} - 2$, element numbers *Idx_Criteria*(i), *Idx_Criteria* (i) +

1, ..., $Idx_Criteria(i + 1) - 1$ of *Criteria* correspond to the $(i + 1)$ -st subset size. Keywords *Criteria* and *Idx_Criteria* must be used together.

Criteria — Named variable into which the one-dimensional array of length $\max(Idx_Criteria(NSIZE - 1), N_ELEMENTS(x(0, *)))$ containing in its first $Idx_Criteria(NSIZE - 1)$ elements the criterion values for each subset considered, in increasing subset size order, is stored. Keywords *Criteria* and *Idx_Criteria* 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 $Idx_Vars(i), Idx_Vars(i) + 1, \dots, Idx_Vars(i + 1) - 1$ 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 $Idx_Vars(NSIZE - 1)$ 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.

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, $NTBEST$ 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 $i = 0, 1, \dots, NTBEST$, rows $Idx_Coefs(i), Idx_Coefs(i) + 1, \dots, Idx_Coefs(i + 1) - 1$ of *Coefs* correspond to the $(i + 1)$ -st regression. Keywords *Coefs* and *Idx_Coefs* must be used together.

Coefs — Named variable into which the two-dimensional array of size $(Idx_Coefs(NTBEST)) \times 5$ 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):

Column	Description
0	variable number

Column	Description
1	coefficient estimate
2	estimated standard error of the estimate
3	<i>t</i> -statistic for the test that the coefficient is 0
4	<i>p</i> -value for the two-sided <i>t</i> test

Keywords *Coefs* and *Idx_Coefs* must be used together.

Discussion

Procedure ALLBEST finds the best subset regressions for a regression problem with

$$n_candidate = (N_ELEMENTS (x (0, *)))$$

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 the user 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.

Procedure ALLBEST 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 the user may wish to input the variance-covariance matrix rather than allow the procedure ALLBEST to calculate it. This can be accomplished using keyword *Cov_Input*. Three situations in which the user 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

Procedure ALLBEST 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 ALLBEST (for $n_candidate > 24$; single precision) and ALLBEST (for $n_candidate > 49$; double precision) can produce incorrect results.
2. Procedure ALLBEST 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 the user wants 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 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 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

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 ALLBEST.

PRINT
PM, idx_coefs, Title = "Idx_Coefs:"

PRINT
PM, Coefs, Title = "Coefs"

PRINT

ntbest = max_n_best
      ; Next, describe how Coefs is to be broken apart by regressions
      ; based on values of Idx_Coefs. Note: NTBEST is defined under the
      ; 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
  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 variables(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 variables(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 variables(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

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.

STEPWISE Procedure

Builds multiple linear regression models using forward, backward, or stepwise selection.

Usage

STEPWISE, *x*, *y*

Input Parameters

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.

Input Keywords

Double — If present and nonzero, double precision is used.

Weights — One-dimensional array containing the weight for each row of *x*.

Default: *Weights* (*) = 1

Frequencies — One-dimensional array containing the frequency for each row of *x*.

Default: *Frequencies* (*) = 1

First_Step or

Inter_Step or

Last_Step or

All_Steps — One or none of these options can be specified. If none of these is specified, the action defaults to *All_Steps*.

Keyword	Action
<i>First_Step</i>	This is the first invocation; additional calls will be made. Initialization and stepping is performed.
<i>Inter_Step</i>	This is an intermediate invocation. Stepping is performed.
<i>Last_Step</i>	This is the final invocation. Stepping and wrap-up computations are performed.
<i>All_Steps</i>	This is the only invocation. Initialization, stepping, and wrap-up computations are performed.

N_Steps — For nonnegative *N_Steps*, *N_Steps* steps are taken. If *N_STEPS* = -1, stepping continues until completion.

Default: *N_STEPS* = 1

Keyword *N_Steps* is not referenced if *All_Steps* is used.

Forward or

Backward or

Stepwise — One or none of these options can be specified. If none is specified, the action defaults to *Backward*.

Keyword	Action
<i>Forward</i>	An attempt is made to add a variable to the model. A variable is added if its <i>p</i> -value is less than <i>P_In</i> . During initialization, only the forced variables enter the model.
<i>Backward</i>	An attempt is made to remove a variable from the model. A variable is removed if its <i>p</i> -value exceeds <i>P_Out</i> . During initialization, all candidate independent variables enter the model.

Keyword	Action
<i>Stepwise</i>	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.

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: $P_Out = 0.10$

Tolerance — Tolerance used in determining linear dependence.

Default: $Tolerance = 100 * \epsilon$, where ϵ is machine precision.

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

$Level(N_ELEMENTS(x(0, *)))$

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*.

Cov_Nobs — The number of observations associated with array Cov_Input . 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 function COVARIANCES. Parameters *x* and *y*, and keywords *Frequencies* and *Weights* are not accessed when this option is specified. Normally, ALLBEST computes *Cov_Input* from the input data matrices *x* and *y*. However, there may be cases when the user wants to calculate the covariance matrix and manipulate it before calling ALLBEST. See the *Discussion* section for a discussion of such cases.

Keywords *Cov_Input* and *Cov_Nobs* must be used together.

Output Keywords

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:

Element	Analysis of Variance Statistic
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	<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

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.

Column	Description
0	coefficient estimate
1	estimated standard error of the coefficient estimate
2	<i>t</i> -statistic for the test that the coefficient is zero
3	<i>p</i> -value for the two-sided <i>t</i> 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:

$$1.0 - (1.0 / VIF)$$

Iend — Named variable into which an integer which indicates whether additional steps are possible is stored.

<i>Iend</i>	Meaning
0	Additional steps may be possible.
1	No additional steps are possible.

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*).

Swept (<i>i</i>)	Status of <i>i</i>-th variable.
-1	Variable <i>i</i> is not in model.
1	Variable <i>i</i> is in model.

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*).

History (<i>i</i>)	Status of <i>i</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 <i>k</i> -th step.
$k < 0.0$	Variable was deleted from model during the <i>k</i> -th step.

Cov_Swept — Named variable into which the two-dimensional array of size $N_ELEMENTS(x(0, *)) + 1 \times (N_ELEMENTS(x(0, *)) + 1)$ 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).

Discussion

Procedure STEPWISE builds a multiple linear regression model using forward, backward, or forward stepwise (with a backward glance) selection. Procedure STEPWISE is designed so the user can monitor, and perhaps change, the variables added (deleted) to (from) the model after each step. In this case, multiple calls to STEPWISE (using keywords *First_Step*, *Inter_Step*, or *Last_Step*) are made. Alternatively, 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 Efroymson (1960). Procedure STEPWISE 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 ALLBEST, page 100) 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
    ; Define some labels for anova_table.

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]

STEPWISE, x, y, Anova_Table = anova_table, $
      Coef_T_Tests = t, swept = s
      ; Backward stepwise regression.

print_results, anova_table, t, s
      ; Print the analysis of variance table.

      * * 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.

```

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.

POLYREGRESS Function

Performs a polynomial least-squares regression.

Usage

result = POLYREGRESS(*x*, *y*, *degree*)

Input Parameters

x — One-dimensional array containing the independent variable.

y — One-dimensional array containing the dependent variable.

degree — Degree of the polynomial.

Returned Value

result — An array of size *degree* + 1 containing the coefficients of the fitted polynomial.

Input Keywords

Double — If present and nonzero, double precision is used.

Weight — Array containing the vector of weights for the observation. If this option is not specified, all observations have equal weights of 1.

Predict_Info — Named variable into which the one-dimensional byte array containing information needed by function 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 POLYPREDICT.

Output Keywords

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 as follows:

Element	Description
($i, 0$)	degrees of freedom
($i, 1$)	sum of squares
($i, 2$)	F -statistic
($i, 3$)	p -value

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 as follows:

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

XMean — Named variable into which the mean of x is stored.

XVariance — Named variable into which the variance of x is stored.

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:

Element	Analysis of Variance Statistic
0	degrees of freedom for the model
1	degrees of freedom for error
2	total (corrected) degrees of freedom

Element	Analysis of Variance Statistic
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.

Ssq_Pure_Error — Named variable into which the sum of squares for pure error is stored.

Residual — Named variable into which the array containing the residuals is stored.

Discussion

Function POLYREGRESS computes estimates of the regression coefficients in a polynomial (curvilinear) regression model. In addition to the computation of the fit, 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 the user for the original model (power form).

Function POLYREGRESS 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).

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. A graph of the results also is given.

$x = [0, 0, 1, 1, 2, 2, 4, 4, 5, 5, 6, 6, 7, 7]$

$y = [508.1, 498.4, 568.2, 577.3, 651.7, \$$

$657.0, 755.3, 758.9, 787.6, 792.1, 841.4, \$$

```

831.8, 854.7, 871.4]
; Define the data vectors.
coefs = 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

```

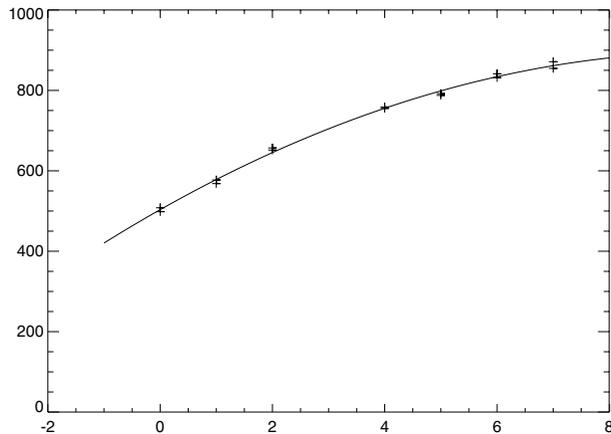


Figure 2-3 Plot of least-squares regression.

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 the 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 = POLYREGRESS(x, y, 2, $
    Anova_Table = anova_table)
    ; Call 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

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.

POLYPREDICT Function

Computes predicted values, confidence intervals, and diagnostics after fitting a polynomial regression model.

Usage

result = POLYPREDICT(*predict_info*, *x*)

Input Parameters

predict_info — One-dimensional byte array containing information computed by function 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 POLYREGRESS.

x — One-dimensional array containing the values of the independent variable for which calculations are to be performed.

Returned Value

result — One-dimensional array containing the predicted values.

Input Keywords

Double — If present and nonzero, double precision is used.

Weights — One-dimensional array containing the weight for each element of *x*. The computed prediction interval uses $SSE / (DFE * Weights(i))$ for the estimated variance of a future response.

Default: *Weights* (*) = 1

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

Y — Array of length N_ELEMENTS (*x*) containing the observed responses.

Output Keywords

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.

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_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.

Leverage — Named variable into which the one-dimensional array of length N_ELEMENTS(x) containing the leverages is stored.

NOTE Y must be specified if any of the following keywords are specified.

Residual — Named variable into which the one-dimensional array of length N_ELEMENTS(x) containing the residuals is stored.

Std_Residual — Named variable into which the one-dimensional array of length N_ELEMENTS(x) containing the standardized residuals is stored.

Del_Residual — Named variable into which the one-dimensional array of length N_ELEMENTS(x) containing the deleted residuals is stored.

Cooks_D — Named variable into which the one-dimensional array of length N_ELEMENTS(x) containing the Cook's D statistics is stored.

Dffits — Named variable into which the one-dimensional array of length N_ELEMENTS(x) containing the DFFITS statistics is stored.

Discussion

Function POLYPREDICT assumes a polynomial model

$$y_i = \beta_0 + \beta_1 x_i + \dots, \beta_k x_i^k + \varepsilon_i \quad i = 1, 2, \dots, n$$

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 , function 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 POLYPREDICT (function POLYREGRESS, page 118).

Results from function POLYREGRESS, which produces the fit using orthogonal polynomials, are used for input by the array *predict_info*. The fitted model from 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:

$$h_i = w_i \sum_{j=0}^k d_j^{-1} p_j^2(z_i)$$

The estimated variance of

$$\hat{y}_i$$

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*.

Example 1

A polynomial model is fit to data using function POLYREGRESS (page 118), then POLYPREDICT is used to compute predicted values.

```
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 = POLYREGRESS(x, y, degree, $
  Predict_Info = predict_info, $
  x2 = 8 * FINDGEN(100)/99)
      ; Call POLYREGRESS using keyword Predict_Info.
predicted = POLYPREDICT(predict_info, x2)
      ; Call POLYPREDICT with Predict_Info.
PLOT, x, y, Psym = 4
      ; Plot the results.
OPLOT, x2, predicted
```

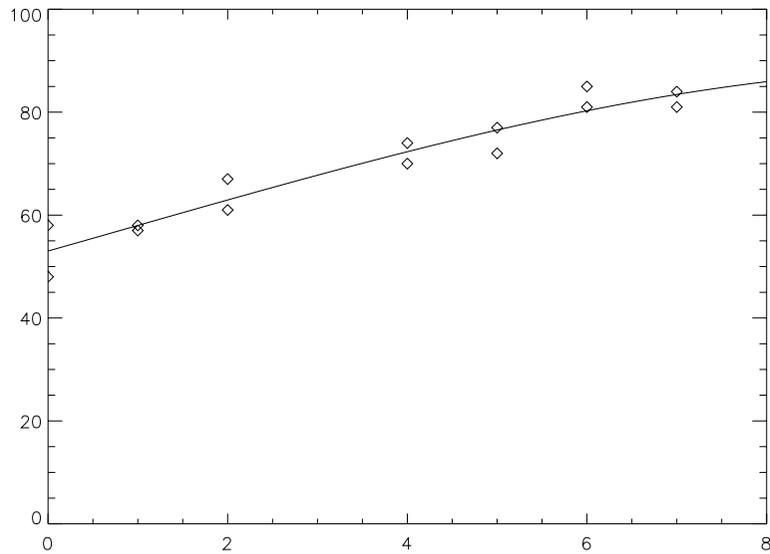


Figure 2-4 Plot of original data with predicted values.

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.

```
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 = POLYREGRESS(x, y, degree, $
    Anova_Table      = anova_table, $
    predict_info     = predict_info)
; Call POLYREGRESS to compute the fit.
predicted = POLYPREDICT(predict_info, x, $
    Ci_Scheffe = ci_scheffe, $
    Y = y, Dffits = dffits)
; Call 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, 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

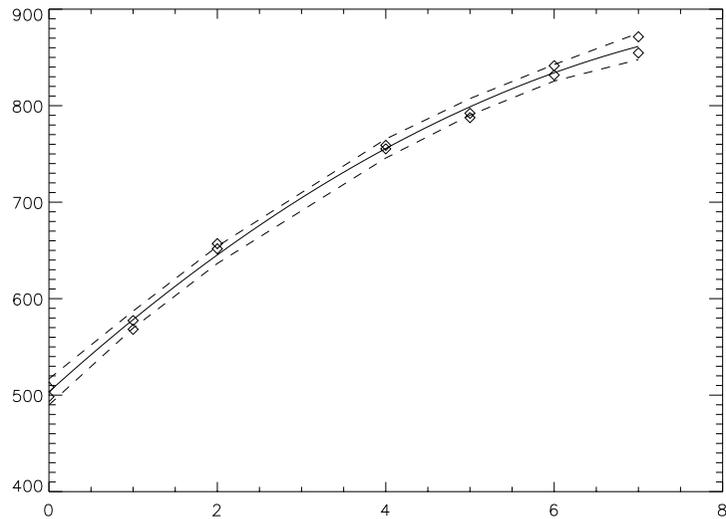


Figure 2-5 Predicted values with confidence bands.

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.

NONLINREGRESS Function

Fits a nonlinear regression model.

Usage

result = NONLINREGRESS(*fcn*, *n_parameters*, *x*, *y*)

Input Parameters

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.

Returned Value

result — A one-dimensional array of length *n_parameters* containing a solution,

$$\hat{\theta},$$

for the nonlinear regression coefficients.

Input Keywords

Double — If present and nonzero, double precision is used.

Theta_Guess — Array with $n_parameters$ components containing an initial guess.

Default: $Theta_Guess(*) = 0$

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 $j = 1, 2, \dots, n_parameters$.

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: $Theta_Scale(*) = 1$

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 $g = \nabla F(\theta)$, $t = Theta_Scale$, 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:

$$Grad_Eps = \sqrt{\epsilon} \ (\sqrt[3]{\epsilon} \text{ in double}),$$

where ϵ is the machine precision

Step_Eps — Scaled step tolerance. The j -th component of the scaled step from points θ and θ_j' is computed as

$$\frac{|\theta_j - \theta_j'|}{\max(|\theta_j|, 1/t_j)}$$

where $t = \text{Theta_Scale}$.

Default: $\text{Step_Eps} = \epsilon^{2/3}$, where ϵ is the machine precision

Sse_Rel_Eps — Relative SSE function tolerance.

Default: $\text{Sse_Rel_Eps} = \max(10^{-10}, \epsilon^{2/3})$, $\max(10^{-20}, \epsilon^{2/3})$ in double, where ϵ is the machine precision

Abs_Eps_Sse — Absolute SSE function tolerance.

Default: $\text{Abs_Eps_Sse} = \max(10^{-20}, \epsilon^2)$, $\max(10^{-40}, \epsilon^2)$ in double, where ϵ is the machine precision

Max_Step — Maximum allowable step size.

Default:

$\text{Max_Step} = 1000 \max(\epsilon_1, \epsilon_2)$, where $\epsilon_1 = (t^T \theta_0)^{1/2}$,

$\epsilon_2 = \|t\|_2$, $t = \text{Theta_Scale}$, and $\theta_0 = \text{Theta_Guess}$

Trust_Region — Size of initial trust region radius. The default is based on the initial scaled Cauchy step.

N_Digit — Number of good digits in the function.

Default: machine dependent

Itmax — Maximum number of iterations.

Default: $\text{Itmax} = 100$

Max_Sse_Evals — Maximum number of SSE function evaluations.

Default: $\text{Max Sse Evals} = 400$

Max_Jac_Evals — Maximum number of Jacobian evaluations.

Default: $\text{Max Jac Evals} = 400$

Tolerance — False convergence tolerance.

Default: *Tolerance* = 100 * ϵ , where ϵ is machine precision.

Output Keywords

Predicted — Named variable into which the one-dimensional array, containing the predicted values at the approximate solution, is stored.

Residual — Named variable into which the one-dimensional array, containing the residuals 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.

Df — Named variable into which the degrees of freedom is stored.

Sse — Named variable into which the residual sum of squares is stored.

Discussion

Function NONLINREGRESS 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 . Function 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.

Function NONLINREGRESS 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 non-critical point. In such instances, the user should either supply a Jacobian function, use the *Double* keyword, or do both.

Programming Notes

Nonlinear regression allows substantial flexibility over linear regression because the user can specify the functional form of the model. This added flexibility can cause unexpected convergence problems for users who are unaware of the limi-

tations 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 user-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 out-

side of the bounds (even though this introduces a discontinuity in the model function).

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 = NONLINREGRESS("fcn", n_parameters, x, y)
PRINT, "Estimated Coefficients:", theta_hat
Estimated Coefficients:
    0.380714    -0.0794534
```

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.

```
FUNCTION fcn, x, theta
    ; Define the function that defines the nonlinear regression problem .
RETURN, theta(0) * EXP(x(0) * theta(1))
END
FUNCTION jac, x, theta
    ; Define the Jacobian function.
fjac = theta
    ; The following assignment produces an array of the 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
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 = NONLINREGRESS("fcn", 2, x, y, $
        Theta_Guess = [60, -0.03], $
        Grad_Eps = 0.001, Jacobian = "jac")
        ; Call NONLINREGRESS.
PLOT, x, y, Psym = 4, $
        Title = 'Nonlinear Regression'
        ; Plot original data.
xtmp = 80 * FINDGEN(200)/199
OPLOT, xtmp, theta_hat(0) * $
        EXP(xtmp * theta_hat(1))
        ; Plot regression.
END

```

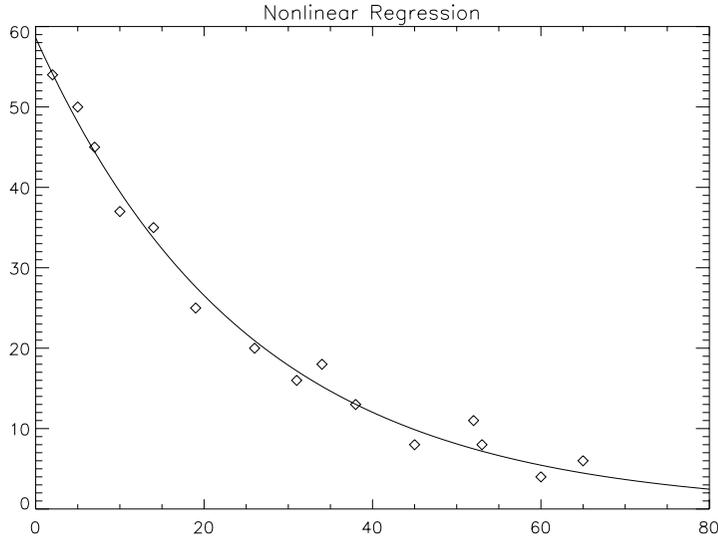


Figure 2-6 Plot of original data and the nonlinear regression fit.

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 the 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 a non-critical point.

HYPOTH_PARTIAL Function

Constructs an equivalent completely testable multivariate general linear hypothesis $H\beta U = G$ from a partially testable hypothesis $H_p\beta U = G_p$.

Usage

result = HYPOTH_PARTIAL(*info_v*, *hp*)

Input Parameters

info_v — One-dimensional array of type BYTE containing information about the regression fit. See function MULTIREGRESS.

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.

Returned Value

result — 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*).

Input Keywords

Double — If present and nonzero, double precision is used.

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.

Output Keywords

Rank_Hp — Named variable into which the rank of H_p is stored.

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.

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.

Discussion

Once a general linear model $y = X\beta + \varepsilon$ 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 function `HYPOTH_SCPH`, page 147) 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). Function `HYPOTH_PARTIAL` 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 function `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, 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 row space 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. 429-430). The method is closely related to a canonical correlation analysis discussed by Kennedy and Gentle (1980, pp. 561-565). The algorithm is as follows:

3. 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 columns of Q_1 . Set $\text{Rank}_{H_p} = n$.

4. 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 .

5. Form

$$A = Q_{11}^T Q_{21}$$

6. 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 $nh = s$.

7. Let W_1 be the first s columns of W . Set $H = (Q_1 W_1)^T$.
8. Assume R_{11} to be a n_{hp} by n_{hp} matrix related to R_1 as follows: If $n_{hp} < n_parameters$, R_{11} equals the first n_{hp} 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.

9. Partition

$$Z^T = (Z_1^T, Z_2^T)$$

so that Z_1 is the first n_1 rows of Z . Set

The degrees of freedom (nh) classify the hypothesis $H_p\beta U = G_p$ as nontestable ($nh = 0$), partially testable ($0 < nh < Rank_{Hp}$), or completely testable ($0 < nh = Rank_{Hp}$).

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

$$y_{ij} = \mu + \alpha_i + \varepsilon_{ij} \quad (i, j) = (1, 1) (2, 1) (2, 2)$$

The model is fitted using function MULTIREGRESS (page 77). The partially testable hypothesis

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 = REGRESSORS(z, n_class, n_cont)
size_x = SIZE(x)
nreg = size_x(2)
coefs = MULTIREGRESS(x, y, Predict_Info = info_v)
% 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 = 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

```

Warning Errors

STAT_HYP_NOT_CONSISTENT — The hypothesis is inconsistent within the computed tolerance.

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.

Usage

result = HYPOTH_SCPH(*info_v*, *h*)

Input Parameters

info_v — One-dimensional array of type BYTE containing information about the regression fit. See function 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.

Returned Value

result — Two-dimensional array, *scph*, containing the sums of squares and crossproducts attributable to the hypothesis.

Input Keywords

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

Output Keywords

Df_h — 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*.

Discussion

Function `HYPOTH_SCPH` computes the matrix of sums of squares and crossproducts for the general linear hypothesis $H\beta U = G$ for the multivariate general linear model $Y = X\beta + \varepsilon$.

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, function `HYPOTH_PARTIAL` (page 141) 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 using Householder transformations and column pivoting to yield 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

$$|t_{rr}| > |t_{11}| \varepsilon \cdot |t_{r+1, r+1}|$$

where $\varepsilon = 10.0 \cdot$ (machine epsilon)

Then, zero out all rows of T below r . Set the degrees of freedom for the hypothesis, Df_h , 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

$$H\beta U = G$$

$$A\beta = Z$$

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 the user to correct the hypothesis before any meaningful sums of squares from this function can be computed. However, in some cases, the user may know the hypothesis is consistent and completely testable, but the checks in `HYPOTH_SCPH` are too tight. For this reason, `HYPOTH_SCPH` continues with the calculations.

Function `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 function `MULTIREGRESS` and the results stored in the structure `info_v`. The sum of squares and crossproducts matrix, `scph`, is then computed by calling `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 = MULTIREGRESS(x, y, Predict_Info = p)
scph = 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
```

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.

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 .

Usage

result = HYPOTH_TEST(*info_v*, *dfh*, *scph*)

Input Parameters

info_v — One-dimensional array of type BYTE containing information about the regression fit. See function MULTIREGRESS.

dfh — Degrees of freedom for the sums of squares and crossproducts matrix.

scph — Two-dimensional array of size nu by nu containing S_H , the sums of squares and crossproducts attributable to the hypothesis.

Returned Value

result — The *p*-value corresponding to Wilks’ lambda test.

Input Keywords

Double — If present and nonzero, double precision is used.

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

Output Keywords

Wilk_Lambda — Named variable into which the one-dimensional array containing the Wilk's lambda 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.

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.

Discussion

Function 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

$$S_H = (H\hat{\beta}U - G)^T (C^T DC)^- (H\hat{\beta}U - G)$$

where C is a solution to $R^T C = H$ and where D is a diagonal matrix with diagonal elements

$$d_{ii} = \begin{cases} 1 & \text{if } r_{ii} > 0 \\ 0 & \text{otherwise} \end{cases}$$

See the section "Linear Dependence and the R Matrix" in the introduction of Chapter 2, *Regression* (page 56).

The error sum of squares and crossproducts matrix for the model $Y = X\beta + \varepsilon$ is

$$(Y - X\hat{\beta})^T (Y - X\hat{\beta})$$

which is input in MULTIREGRESS. The error sum of squares and crossproducts matrix for the hypothesis $H\beta U = G$ computed by 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.*,

$$p = \begin{cases} \text{NU} & \text{if NU} > 0 \\ \text{NDEP} & \text{otherwise} \end{cases}$$

Let q (stored in *dfh*) be the degrees of freedom for the hypothesis. Let v (input in *info_v*) be the degrees of freedom for error. Function HYPOTH_TEST 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

$$c = \max \lambda_i \quad \text{over all } i$$

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

$$U = tr(HE^{-1}) = \sum_{i=1}^p \lambda_i$$

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

$$F = \frac{b(v - p - 1)}{(b - 2)pq} U$$

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, *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, `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

$$V = \text{tr} \left[S_H (S_H + S_E)^{-1} \right] = \sum_{i=1}^p \theta_i$$

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 $\min(p, q) = 1$.

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 function `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, function `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 = MULTIREGRESS(x, y, Predict_Info = p)
scph = HYPOTH_SCPH(p, h, Dfh = dfh)
pvalue = 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 = MULTIREGRESS(x, y, Predict_Info = p)

scph = HYPOTH_SCPH(p, h, Dfh = dfh)

pvalue = 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
```

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 = (lambda)*(“scph”+“scpe”)*x failed to converge.

STAT_NO_STAT_3 — No statistics can be computed. Iterations for eigenvalues for the generalized eigenvalue problem “scph”*x = (lambda)*(“scph”+“u”*“scpe”*“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 #.

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.

Usage

result = NONLINOPT(*f*, *n_parameters*, *x*, *y*)

Input Parameters

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 *xi*. 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.

Returned Value

result — One-dimensional array of length *n_parameters* containing a solution,

$$\hat{\theta}$$

for the nonlinear regression coefficients.

Input Keywords

Double — If present and nonzero, double precision is used.

Theta_Guess — One-dimensional array with $n_parameters$ components containing an initial guess.

Default: $Theta_Guess(*) = 0$

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 $f(x_i)/q$ 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, NONLINREGRESS, the Jacobian values must be specified as the *negative* of the calculated derivatives.

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 .

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). Keywords *A_Matrix* and *B* must be used together.

Default: There are no default linear constraints.

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 $j = 1, n_parameter$.

Meq — Number of the *A_Matrix* constraints which are *equality* constraints; the remaining (`n_constraints - Meq`) constraints are *inequality* constraints.

Default: *Meq* = 0.

Frequencies — One-dimensional array of length `n_observations` containing the frequency for each observation.

Default: *Frequencies*(*) = 1

Weights — One-dimensional array of length `n_observations` containing the weight for each observation.

Default: *Weights*(*) = 1

Acc — The nonnegative tolerance on the first order conditions at the calculated solution.

Max_Sse_Evals — The maximum number of sse evaluations allowed.

Default: *Max_Sse_Eval* = 400

Output Keywords

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 sse fails to decrease although a decrease is predicted by the current gradient vector.

Stop_info	Reason for leaving routine
4	The calculation cannot begin because A_Matrix contains fewer than $n_constraints$ constraints or because the lower bound on a variable is greater than the upper bound.
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 (Max_Sse_Eval) is exceeded.
9	θ is determined by the equality constraints.

Num_Active — Named variable into which the final number of active constraints is stored.

Active_Const — Named variable into which a one-dimensional array of length Num_Active containing the indices of the final active constraints is stored.

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.

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.

Discussion

Function NONLINOPT is based on M.J.D. Powell's TOLMIN, which solves linearly constrained optimization problems, i.e., problems of the form $\min f(\theta)$, $\theta \in \mathfrak{R}$, subject to

$$A_1\theta = b_1$$

$$A_2\theta \leq b_2$$

$$\theta_l \leq \theta \leq \theta_u$$

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 provided by the user, 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 $\theta_i \leq (\theta_l)_i$. 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

$$(d^k)^T \nabla f(\theta^k + \alpha^k d^k) - \nabla f(\theta^k) > 0$$

holds. Let $\theta^k \leftarrow \theta^{k+1}$, and start another iteration.

The iteration repeats until the stopping criterion

$$\|\nabla f(\theta^k) - A^k \lambda^k\|_2 \leq \tau$$

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 NONLINOPT using the optional keyword *Jacobian*.

Example 1

In this example, a data set is fitted to the nonlinear model function

$$y_i = \sin(\theta_0 x_i) + \varepsilon_i$$

```

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 = NONLINOPT("fcn", n_parameters, x, y)
% 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. 641470] 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.

$$y_i = \theta_0 + (0.49 - \theta) e^{-\theta(x_i - 8)} + \varepsilon_i$$

Since the chlorine level will stabilize at about 30%, the initial guess for theta1 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.

```

FUNCTION fcn, x, theta
    res = theta(0) + (0.49 - theta(0)) * $
        exp(-theta(1) * (x(0) - 8.0))
    RETURN, res
END
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, 0.47, 0.48, 0.47, 0.46, 0.46, 0.45, $
     0.43, 0.45, 0.43, 0.43, 0.44, 0.43, 0.43, 0.46, 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 = NONLINOPT("fcn", n_parameters, x, y, $
                    Theta_Guess = theta_guess, Xlb =
                    xlb, $
                    Jacobian = "jacobian", Sse = sse)

% 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 =      0.390143      0.101631
PRINT, "Residual Sum of Squares =", sse
Residual Sum of Squares =      0.00500168

```

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” = #.

LNORMREGRESS Function

Fits a multiple linear regression model using criteria other than least squares. Namely, LNORMREGRESS allows the user to choose Least Absolute Value (L_1), Least L_p norm (L_p), or Least Maximum Value (Minimax or L_{∞}) method of multiple linear regression.

Usage

result = LNORMREGRESS(*x*, *y*)

Input Parameters

x — Two-dimensional array of size *n_rows* by *n_independent* containing the independent (explanatory) variables(s) where *n_rows* = N_ELEMENTS(*x*(*,0)) and *n_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.

Returned Value

result — One-dimensional array of length *n_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. LNORMREGRESS returns the L_p norm or least maximum value solution for the regression coefficients when appropriately specified in the input keyword list.

Input Keywords

Double — If present and nonzero, double precision is used.

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, 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.

P — The p in the L_p norm criterion (see the *Discussion* section for details). P must be greater than or equal to one. Keywords P and Llp must be used together.

Lmv — If present and nonzero, LNORMREGRESS fits a multiple linear regression model using the minimax criterion. Keywords Lav , Llp , and Lmv can not be used together.

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: $Eps = 100 * (\text{machine epsilon})$.

Weights — One-dimensional array of size n_rows containing the weights for the independent (explanatory) variable. Keyword Llp is required when using keyword *Weights*.

Frequencies — One-dimensional array of size n_rows containing the frequencies for the independent (explanatory) variable. Keyword Llp is required when using keyword *Frequencies*.

No_Intercept — If present and nonzero, the intercept term

$$(\hat{\beta}_0)$$

is omitted from the model and the Returned 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 $k = n_independent$.

Tolerance — Tolerance used in determining linear dependence. Keyword Llp is required when using keyword *Tolerance*.

Default: $Tolerance = 100 * (\text{machine epsilon})$.

Output Keywords

Rank — Named variable into which the rank of the fitted model is stored.

Iters — Named variable into which the number of iterations performed is stored.

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.

Sea — Named variable into which the sum of the absolute value of the errors is stored. Keyword *Lav* is required when using keyword *Sea*.

Resid_Max — Named variable into which the magnitude of the largest residual is stored. Keyword *Lmv* is required when using keyword *Resid_Max*.

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*.

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*.

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*.

Resid_Norm — Named variable into which the L_p norm of the residuals is stored. Keyword *Llp* is required when using keyword *Resid_Norm*.

Discussion

Least Absolute Value Criterion

Function LNORMREGRESS 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 *Lp* 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 the 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.

In each case, for the first iteration a least-squares solution for the regression coefficients is computed using routine MULTIREGRESS (page 77). 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 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

$$\delta_i^{(k+1)}$$

in the i -th residual from iteration k to iteration $k + 1$ is computed as follows:

$$\delta_i^{(k+1)} = \begin{cases} 0 & \text{if } e_i^{(k+1)} = e_i^{(k)} = 0 \\ \left| e_i^{(k+1)} - e_i^{(k)} \right| / \max\left(\left| e_i^{(k)} \right|, \left| e_i^{(k+1)} \right|, s \right) & \text{otherwise} \end{cases}$$

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

$$r_i^{(k)} = \left[(e_i^{(k)})^2 + c^2 \right]^{(p-2)/2}$$

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

$$\hat{\beta}^{(0)} = \hat{\beta}^{(k)} + \Delta cd$$

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 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)

$$\hat{\lambda}^2 = \left[\frac{\sqrt{DFE} (\tilde{e}_{(DFE-k+1)} - \tilde{e}_{(k)})}{2z_{0.975}} \right]^2$$

with

$$k = \frac{DFE + k}{2} - z_{0.975} \sqrt{\frac{DFE}{4}}$$

where $z_{0.975}$ is the 97.5 percentile of the standard normal distribution, and where

$$\tilde{\epsilon}_{(m)} (m = 1, 2, \dots, DFE)$$

are the ordered residuals where *Rank* zero residuals are excluded. Note that

$$DFE = \sum_{i=0}^{n-1} f_i - \text{irank}$$

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

$$m_r = \frac{\sum_{i=0}^{n-1} f_i |\sqrt{w_i} (y_i - \hat{y}_i)|^r}{\sum_{i=0}^{n-1} f_i}$$

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.

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 = 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.

```

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

```

```

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 = 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	2.500
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.

```

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 = 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
```

Correlation and Covariance

Contents of Chapter

Variances, Covariances, and Correlations

Variance-covariance or correlation matrix	COVARIANCES Function
Partial correlations and covariances	PARTIAL_COV Function
Pooled covariance matrix	POOLED_COV Function
Robust estimate of covariance matrix	ROBUST_COV Function

Introduction

This chapter is concerned with measures of correlation for bivariate data as follows:

- The usual multivariate measures of correlation and covariance for continuous random variables are produced by routine COVARIANCES.
- For data grouped by some auxiliary variable, routine POOLED_COV can be used to compute the pooled covariance matrix along with the means for each group.
- Partial correlations or covariances are computed by PARTIAL_COV.

- Function ROBUST_COV computes robust M-estimates of the mean and covariance matrix from a matrix of observations.

COVARIANCES Function

Computes the sample variance-covariance or correlation matrix.

Usage

result = COVARIANCES(*x*)

Input Parameters

x — Two-dimensional matrix containing the data. The data value for the *i*-th observation of the *j*-th variable should be in $x(i,j)$.

Returned Value

result — If no keywords are used, 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.

Input Keywords

Double — If present and nonzero, double precision is used.

Var_Covar or

Corrected_Sscp or

Correlation or

Stdev_Correlation — Exactly one of these options is used to specify the type of matrix to be computed.

Keyword	Type of Matrix
<i>Var_Covar</i>	variance-covariance matrix (default)
<i>Corrected_Sscp</i>	corrected sum-of-squares and crossproducts matrix
<i>Correlation</i>	correlation matrix

Keyword	Type of Matrix
<i>Stdev_Correlation</i>	correlation matrix, except for diagonal elements which are standard deviations

Weight — Array containing the vector of weights for the observation.

Default: all observations have equal weights of 1.

Frequencies — Array containing the vector of frequencies for the observation.

Default: all observations have a frequency of 1.

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:

Missing_Val	Action
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.

Output Keywords

Means — Named variable into which the array containing the means of the 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.

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 $nvar \times nvar$ 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.

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.

Discussion

Function COVARIANCES 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} :

$$r_{i+1} = \frac{f_{i+1}w_{i+1}}{\sum_{j=1}^{i+1} f_j w_j}$$

$$\bar{x}_{k, i+1} = \bar{x}_{ki} + (x_{k, i+1} - \bar{x}_{ki})r_{i+1}$$

$$c_{jk, i+1} = c_{jki} + f_{i+1}w_{i+1}(x_{j, i+1} - \bar{x}_{ji})(x_{k, i+1} - \bar{x}_{ki})(1 - r_{i+1})$$

Usage Notes

Function COVARIANCES 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 .

$$s_{jk} = \frac{\sum_{i=1}^n f_i w_i (x_{ji} - \bar{x}_j)(x_{ki} - \bar{x}_k)}{\left(\sum_{i=1}^n f_i\right) - 1}$$

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 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 = STATDATA(3)
x = x(0:49, *)
cov = COVARIANCES(x)
      ; Call 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
```

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.

PARTIAL_COV Function

Computes partial covariances or partial correlations from the covariance or correlation matrix.

Usage

```
result = PARTIAL_COV(n_independent, n_dependent, x)
```

Input Parameters

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.

n_dependent — Number of variables for which partial covariances/correlations are desired (the number of “dependent” 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.

Returned Value

result — Array of size $n_dependent$ by $n_dependent$ containing the partial covariances (the default) or partial correlations (set keyword *Corr*).

Input Keywords

Double — If present and nonzero, double precision is used.

Indices — An array containing values indicating the status of the variable as in the following table:

Indices(i)	Variable is...
-1	not used in analysis
0	dependent variable
1	independent variable

Default: The first $n_independent$ elements of *Indices* are equal to 1, and the last $n_dependent$ elements are equal to 0.

Cov — If present and nonzero, then partial covariances are calculated. (Default) Keywords *Cov* and *Corr* can not be used together.

Corr — If present and nonzero, then partial correlations are calculated. Keywords *Cov* and *Corr* can not be used together.

Input/Output Keywords

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.

Output Keywords

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

Function PARTIAL_COV 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, 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

$$\begin{pmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{pmatrix}$$

function 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.

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 = 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 = PARTIAL_COV(3, 6, x, Indices = indices, Df = df, $
Pvals = pvals, /Corr)
PRINT, 'Degrees Of Freedom: ', df
Degrees Of Freedom:          27
PM, pcov, Format = '(6F10.3)', Title = 'Partial Correlations'
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
PM, pvals, Format = '(6F10.4)', Title = 'P values'
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

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.

POOLED_COV Function

Compute a pooled variance-covariance from the observations.

Usage

result = POOLED_COV(*x*, *ngroups*)

Input Parameters

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.

ngroups — Number of groups in the data.

Returned Value

result — Two-dimensional array containing the matrix of covariances.

Input Keywords

Double — If present and nonzero, double precision is used.

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, ..., $n_variables - 1$,

Idx_Vars(0) = $n_variables$,

Idx_Vars(1) = -1, and

Idx_Vars(2) = -1

Output Keywords

Gcounts — Named variable into which the array of length n_groups containing the number of observations in each group 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.

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.

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 $S = U^T U$.

Nmissing — Named variable into which the number of rows of data containing missing values (NaN) for any of the variables used is stored.

Discussion

Function POOLED_COV 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. Function POOLED_COV should be used whenever the user suspects that 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

$$\bar{x}_{i\bullet} = \frac{T_i}{\sum_j w_i f_i}$$

$$S = \frac{1}{\sum_{ij} f_{ij} - g} \sum_{i,j} w_{ij} f_{ij} (x_{ij} - \bar{x}_{i\bullet})(x_{ij} - \bar{x}_{i\bullet})^T$$

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 = POOLED_COV(x, ngroups)
PM, cov, Format = "(2F10.3)", Title = "Pooled Covariance Matrix"
Pooled Covariance Matrix
      0.708      -1.575
     -1.575       3.883
```

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.

ROBUST_COV Function

Computes a robust estimate of a covariance matrix and mean vector.

Usage

```
result = ROBUST_COV(x, n_groups)
```

Input Parameters

x — Two-dimensional array of size nrows by (n_variables + 1) containing the data where n_rows = 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.

n_groups — Number of groups in the data.

Returned Value

result — Two-dimensional array containing the matrix of covariances.

Input Keywords

Double — If present and nonzero, double precision is used.

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

$Idx_Vars(2) = -1$

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.

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.

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.

Stahel — If present and nonzero, the Stahel's algorithm is used. Keywords *Stahel* and *Huber* can not be used together.

Huber — If present and nonzero, Huber's conjugate-gradient algorithm is used. Keywords *Stahel* and *Huber* can not be used together.

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

Itmax — Maximum number of iterations.

Default: *Itmax* = 30

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}

Output Keywords

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.

Group_Counts — Named variable into which the one-dimensional array of length *n_groups* containing the number of observations in each group is stored.

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.

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.

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 $S = U^T U$.

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.

Nmissing — Named variable into which the number of rows of data containing missing values (NaN) for any of the variables used is stored.

Discussion

Function ROBUST_COV 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:

$$u(r) = \begin{cases} \frac{a^2}{r^2} & r < a \\ 1 & a \leq r \leq b \\ \frac{b^2}{r^2} & r > b \end{cases}$$
$$w(r) = \min\left(1, \frac{c}{r}\right)$$

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”.

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 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.

Function ROBUST_COV 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

$$\hat{\mu}_{i,k+1} = \hat{\mu}_{i,k} + T_{ki}$$

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 $\varepsilon = \textit{Tolerance}$, 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, the user inputs initial estimates.

Function `ROBUST_COV` 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 the warning message is issued), the computed estimates are still optimal, but the scale of the estimated covariance ma-

trix may need to be multiplied by a constant in order for

$$\hat{\Sigma}$$

to have the correct multivariate normal covariance expectation.

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 = 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 function POOLED_COV. Function ROBUST_COV 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 POOLED_COV and ROBUST_COV. When outliers are present, the estimates of POOLED_COV are adversely affected, while the estimates produced by 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 = STATDATA(3)
```

```

p_cov = POOLED_COV(x, n_groups, Idx_Vars = idxv, $
                  Idx_Cols = idxc)
PM, p_cov, Title = "Pooled Covariance with No Outliers"
Pooled Covariance with No Outliers
      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
r_cov = ROBUST_COV(x, n_groups, Idx_Vars = idxv, $
                  Idx_Cols = idxc, Percentage = percentage)
PM, r_cov, Title = "Robust Covariance with No Outliers"
Robust Covariance with No Outliers
      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

```


Analysis of Variance

This chapter describes functions for analysis of variance models and for multiple comparison methods for means.

Contents of Chapter

Analyzes a one-way classification model	ANOVA1 Function
Analyzes a balanced factorial design with fixed effects	ANOVAFACT Function
Performs Student-Newman-Keuls multiple comparisons test.....	MULTICOMP Function
Nested random model	ANOVANESTED Function
Balanced fixed, random, or mixed model.....	ANOVABALANCED Function

Introduction

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.

Function ANOVA1 allows missing responses if confidence interval information is not requested. NaN (Not a Number) is the missing value code used by these functions. Use function 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 2: *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.

ANOVA1 Function

Analyzes a one-way classification model.

Usage

$result = ANOVA1(n, y)$

Input Parameters

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.

Returned Value

$result$ — The p -value for the F -statistic.

Input Keywords

Double — If present and nonzero, then double precision is used.

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

Output Keywords

Anova_Table — Named variable into which the analysis of variance table is stored.

The analysis of variance statistics are as follows:

Element	Analysis of Variance Statistics
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)

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.

Group_Counts — Named variable into which the array containing the number of nonmissing observations for the groups is stored.

*Tukey or
Dunn_Sidak or*

Bonferroni or

Scheffe or

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 $ngroups = N_ELEMENTS(n)$.

Column	Description
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

Function ANOVA1 computes the 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, the Tukey confidence intervals are calculated if the group sizes are equal; otherwise, the Tukey-Kramer confidence intervals are calculated.

Discussion

Function ANOVA1 performs an analysis of variance of responses from a one-way classification design. The model is

$$y_{ij} = \mu_i + \varepsilon_{ij} \quad i = 1, 2, \dots, k; \quad j = 1, 2, \dots, n_i$$

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 . Function ANOVA1 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).

Function ANOVA1 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-Sidák 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-Sidák method: The Dunn-Sidák 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 function 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 100 f 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

$$\sum^k c_i \mu_i$$

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 function 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-Sidák, 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:

$$\bar{y}_i - \bar{y}_j \pm t_{1-\frac{\alpha}{2};v} \sqrt{\frac{s^2}{n_i} + \frac{s^2}{n_j}}$$

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—three normal, two off-types, and one aberrant.

Normal	Off-Type	Aberrant
101	84	32
105	88	
94		

```
n = [3,2,1]
y = [101.0, 105.0, 94.0, 84.0, 88.0, 32.0]
PRINT,'p-value = ', ANOVA1(n, y)
p-value = 0.00276887
```

Example 2: Multiple Comparisons

Simultaneous confidence intervals are generated for the following measurements of cold-cranking power for five models of automobile batteries. 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

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.

```
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 = ANOVA1(n, y, Confidence = 99.0, $
    Anova_Table = anova_table, $
    Tukey = diff_means)
    ; Call 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

ANOVAFACT Function

Analyzes a balanced factorial design with fixed effects.

Usage

result = ANOVAFACT(*n_levels*, *y*)

Input Parameters

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.

Returned Value

result — The *p*-value for the overall *F*-test.

Input Keywords

Double — If present and nonzero, then double precision is used.

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

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.

Output 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:

Element	Analysis of Variance Statistics
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)

Test_Effects — Named variable into which an array of size $n_{ef} \times 4$ containing statistics relating to the sums of squares for the effects in the model is stored. Here,

$$n_{ef} = \binom{n}{1} + \binom{n}{2} + \dots + \binom{n}{\min(n, |Order|)}$$

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*

Column	Description
0	degrees of freedom
1	sum of squares
2	<i>F</i> -statistic
3	<i>p</i> -value

are as follows:

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.

Discussion

Function ANOVAFACT 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.

Function ANOVAFACT 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.

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

$$\sum_{i=0}^1 \alpha_i = 0; \sum_{i=0}^2 \beta_j = 0; \sum_{i=0}^1 \gamma_{ij} = 0$$

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 the following table:

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

$n = [3, 2, 10]$

$y = [73.0, 102.0, 118.0, 104.0, 81.0, \$$
 $107.0, 100.0, 87.0, 117.0, 111.0, \$$
 $90.0, 76.0, 90.0, 64.0, 86.0, \$$

```

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 = ANOVAFACT(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.

```

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 the statistics for effects.

PRINT, " * * Subgroup Means * *"
FOR i = 0, 11 DO PM, means_labels(i), $
    means(i), Format = '(a5,f15.2)'
    ; Print the means.

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 = ANOVAFACT(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 the following table:

	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	97.85	94.83	100.49	99.75	99.90	100.23	104.51

	A_0			A_1			A_2		
C_1	87.45	98.27	95.85	84.57	97.20	112.30	92.98	107.77	110.94
C_2	86.01	104.20	90.09	81.06	120.80	108.77	94.72	118.39	102.87

```

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 the analysis of variance table.

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, *)
    ; Print the statistics for effects.

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 = ANOVAFACT(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

```

MULTICOMP Function

Performs Student-Newman-Keuls multiple-comparisons test.

Usage

result = MULTICOMP(*means*, *df*, *std_error*)

Input Parameters

means — One-dimensional array containing the means.

df — Degrees of freedom associated with *std_error*.

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.

Returned Value

result — 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.

Input Keywords

Double — If present and nonzero, then double precision is used.

Alpha — Significance level of test. Must be in the interval [0.01, 0.10].

Default: *Alpha* = 0.01

Discussion

Function MULTICOMP 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 = MULTICOMP(means, df, std_error)
PM, equal_means, $
  Title = "Size of groups of means:"
Size of groups of means:
  3
  3
  3
  0
```

ANOVANESTED Function

Analyzes a completely nested random model with possibly unequal numbers in

the subgroups.

Usage

result = ANOVANESTED(*n_factors*, *eq_option*, *n_levels*, *y*)

Input Parameters

n_factors — Number of factors (number of subscripts) in the model, including error.

eq_option — Equal numbers option.

eq_option	Description
0	Unequal numbers in the subgroups
1	Equal numbers in the subgroups

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 following additional variables are referred to in the description of 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 $n_levels(0) * n_levels(1) * \dots *$ $n_levels(n_factors-1).$

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 ANOVANESTED:

Variable	Description
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.

Returned Value

result — The p -value for the F-statistic.

Input Keywords

Double — If present and nonzero, then double precision is used.

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 $Confidence = 100.0 - 2.0 * (100.0 - ONECL)$.

Default: $Confidence = 95.0$

Output Keywords

Anova_Table — Named variable into which the array of size 15 containing the analysis of variance table is stored. The analysis of variance statistics are as fol-

lows:

Element	Analysis of Variance Statistics
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)

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:

Column	Descriptions
1	Degrees of freedom
2	Sum of squares
3	Mean squares
4	F -statistic
5	<i>p</i> -value for F test
6	Variance component estimate
7	Percent of variance explained by variance component
8	Lower endpoint for a confidence interval on the variance component
9	Upper endpoint for a confidence interval on the variance component

If a test for the error variance equal to zero cannot be performed, $Var_Comp(n_factors, 4)$ and $Var_Comp(n_factors, 5)$ are set to NaN (not a number).

Ems — One-dimensional array of length $n_factors * ((n_factors + 1)/2)$ with expected mean square coefficients.

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 the description of argument n_levels for definitions of LNL and LNLNF.)
1	$1 + n_levels(0) + n_levels(0) * n_levels(1) + \dots + n_levels(0) * n_levels(1) * \dots * n_levels(n_factors - 2)$

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

Function ANOVANESTED 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 1

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 are given in the following table:

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

```

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
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 = 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

ANOVABALANCED Function

Analyzes a balanced complete experimental design for a fixed, random, or mixed model.

Usage

result = ANOVABALANCED(*n_levels*, *y*, *n_random*, *idx_rand_fct*, *n_fct_per_eff*, *idx_fct_per_eff*)

Input Parameters

n_levels — One-dimensional array containing the number of levels for each of the factors.

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.

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).

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 asso-

ciated with each effect in the model.

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 .

Returned Value

result — The p -value for the F -statistic.

Input Keywords

Double — If present and nonzero, then double precision is used.

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 α , α in the interval [50.0, 100.0], set *Confidence* = $100.0 - 2.0 * (100.0 - \alpha)$.

Default: *Confidence* = 95.0

Model — Model Option

Model	Description
0	Searle model
1	Scheffe model

For the 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.

Default: *Model* = 0

Output 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:

Element	Analysis of Variance Statistics
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)

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.

Column	Description
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 a confidence interval on the variance component
9	Upper endpoint for a 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

Function ANOVABALANCED 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

$$\sum_{i=1}^a \alpha_i = 0$$

the b_j 's are random effects identically and independently distributed

$$N(0, \sigma_B^2)$$

c_{ij} are interaction effects each distributed

$$N\left(0, \frac{a-1}{a} \sigma_{AB}^2\right)$$

and are subject to the restrictions

$$\sum_{i=1}^a c_{ij} = 0 \text{ for } j = 1, 2, \dots, b$$

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 $Model = 0$.

The computations for degrees of freedom and sums of squares are the same regardless of the option specified by *Model*. 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 the variance of the associated random effect divided by the 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 earli-

er two-way mixed model is computed for $Model = 1$ using the formula

$$\% \text{ variation(AB|Model} = 1) = \frac{\frac{a-1}{a} \sigma_{AB}^2}{\sigma_B^2 + \frac{a-1}{a} \sigma_{AB}^2 + \sigma^2}$$

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 the 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 are given in the following table:

	Block			
Treatment	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

```

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]
indrff = [2, 3]
nfef = [1, 1, 2]
indef = [1, 2, 1, 2]
p = ANOVABALANCED(n_levels, y, 2, indrff, nfef, indef, $
                  Anova_Table = at, Ems = ems, $
                  Y_Means = y_means, Var_Comp = var_comp)
print_results, p, at, ems, y_means, var_comp
% ANOVABALANCED: Note: STAT_ENDPNTS_NEGATIVE
    One or more endpoints are negative and are set to zero.
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 Er-
ror NaN
p-value for Er-
ror 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
2.75057

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 = POOLED_COV(x, n_groups, Idx_Vars = idxv, $
                Idx_Cols = idxc)
PM, p_cov, Title = "Pooled Covariance with Outliners"
Pooled Covariance 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
r_cov = ROBUST_COV(x, n_groups, Idx_Vars = idxv, $
                Idx_Cols = idxc, Percentage = percent-
age)
PM, r_cov, Title = "Robust Covariance with Outliners"
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

```

Categorical and Discrete Data Analysis

Contents of Chapter

Statistics in the Two-Way Contingency Table

Two-way contingency table analysis	CONTINGENCY Function
Exact probabilities in a table; total enumeration	EXACT_ENUM Function
Exact probabilities in a table	EXACT_NETWORK Function

Generalized Categorical Models

Generalized linear models	CAT_GLM Function
---------------------------------	----------------------------------

Introduction

Routine 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 EXACT_ENUM , but this routine uses the total enumeration algorithm and, thus, often uses orders of magnitude more

computer time than EXACT_NETWORK which computes the same probabilities by use of the network algorithm (but can still be quite expensive).

The routine 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 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.

CONTINGENCY Function

Performs a chi-squared analysis of a two-way contingency table.

Usage

result = CONTINGENCY(*table*)

Input Parameters

table — Two-dimensional array containing the observed counts in the contingency table.

Returned Value

result — Pearson chi-squared *p*-value for independence of rows and columns.

Input Keywords

Double — If present and nonzero, double precision is used.

Output Keywords

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.

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 .

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.

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:

Element	Chi-squared Statistics
0	exact mean
1	exact standard deviation
2	phi
3	contingency coefficient
4	Cramer's V

Table_Stats — Named variable into which a two-dimensional array of size 23 x 5 containing statistics associated with this table is stored. Each row corresponds to a statistic.

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)

Row	Statistic
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)
21	McNemar test of symmetry (square tables only)
22	McNemar one degree of freedom test of symmetry (square tables only)

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:

Column	Value
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

Function CONTINGENCY 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:

$$G^2 = -2 \sum x_{ij} \ln(x_{ij}/np_{ij})$$

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. Function CONTINGENCY 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:

$$\sum \sum a_{uv} b_{uv}$$

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 are 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 the a_{uv} variables 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 $n^2m / (m - 1)$, 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 ($p_{\bullet m}$). 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:

$$n/2 - \left(\sum x_{i\cdot}^2 \right) / (2n)$$

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:

$$q_j = x_{\cdot j} / 2 - \left(\sum x_{ij}^2 \right) / (2x_{i\cdot})$$

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:

$$U_{r|c} = \frac{\sum_{i,j} x_{ij} \log(x_{i\cdot} x_{\cdot j} / n x_{ij})}{\sum x_{i\cdot} \log(x_{i\cdot} / n)}$$

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

$$\hat{\beta} = \frac{\sum x_{\cdot j}(x_{1j}/x_{\cdot j} - x_{1\cdot}/n)(j - \bar{j})}{\sum x_{\cdot j}(j - \bar{j})^2}$$

where

$$\bar{j} = \sum x_{\cdot j} j / n$$

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

$$p_0 = \sum x_{ii} / n$$

denote the probability that the two judges agree, and let

$$p_c = \sum e_{ii} / n$$

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:

$$\sum \frac{(x_{ij} - x_{ji})^2}{(x_{ij} + x_{ji})}$$

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:

$$\frac{\left(\sum_{i < j} (x_{ij} - x_{ji}) \right)^2}{\sum (x_{ij} + x_{ji})}$$

The exact probability can be computed by the binomial distribution.

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", CONTINGENCY(table)
P-Value 0.00000
```

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.

```
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 = 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
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

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.

EXACT_ENUM Function

Computes exact probabilities in a two-way contingency table using the total enumeration method.

Usage

result = EXACT_ENUM(*table*)

Input Parameters

table — Two-dimensional array containing the observed counts in the contingency table.

Returned Value

result — 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”.

Input Keywords

Double — If present and nonzero, double precision is used.

Output Keywords

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.

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*.

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.

Discussion

Function EXACT_ENUM 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. The user should note that this definition can be considered “two-sided” in the cell counts.

Because 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 $f_{..}$ or a large value of r by c should not be analyzed using EXACT_ENUM. In such cases, try using EXACT_NETWORK.

Example

In this example, the exact conditional probability for the 2 by 2 contingency table

$$\begin{bmatrix} 8 & 12 \\ 8 & 2 \end{bmatrix}$$

is computed.

```
table = [[8, 8], [12, 2]]
p = EXACT_ENUM(table, P_Value = pv, Prob_Table = pt, $
              Error_Chk = ec)
PRINT, "p-value =", p
p-value =      0.0576712
```

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.

Usage

result = EXACT_NETWORK(*table*)

Input Parameters

table — Two-dimensional array containing the observed counts in the contingency table.

Returned Value

result — 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”.

Input Keywords

Double — If present and nonzero, double precision is used.

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.

No_Approx — If present and nonzero, the Fisher exact test is used and *Approx_Param* is ignored.

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.

Function EXACT_ENUM 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 restarted. 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 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, $nt = f_{..} + 1$, $mx = \max(n_rows, n_columns)$, $mn = \min(n_rows, n_columns)$, $t1 = \max(800 + 7mx, (5 + 2mx)(n_rows + n_columns + 1))$, and $t2 = \max(400 + mx, + 1, n_rows + n_columns + 1)$ where $n_rows = N_ELEMENTS(table(*,0))$ and $n_columns = N_ELEMENTS(table(0,*))$.

The following amount of integer workspace is allocated: $3mx + 2mn + t1$.

The following amount of real workspace is allocated: $nt + t2$.

The remainder of the 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: $Wk_Params(0) = 100$

$Wk_Params(1) = 3000$

$Wk_Params(2) = 10$

Output Keywords

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.

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*.

Discussion

Function EXACT_NETWORK 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

$$P_f = \frac{\prod_{i=1}^r f_{i\cdot}! \prod_{j=1}^c f_{\cdot j}!}{f_{\cdot\cdot}! \prod_{i=1}^r \prod_{j=1}^c f_{ij}!}$$

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. The user should note that this definition can be considered “two-sided” in the cell counts.

Example

The following example demonstrates and compares the various methods of computing the 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 function CONTINGENCY) can be different.

```
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 = CONTINGENCY(table)
p2 = EXACT_NETWORK(table)
p3 = EXACT_NETWORK(table, /No_Approx)
p4 = EXACT_ENUM(table)
print_results, p, p2, p3, p4
% CONTINGENCY: Warning: STAT_EXP_VALUES_TOO_SMALL
    Some expected values are less than 1. Some asymptotic
p-values may not be good.
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

```

Warning Errors

STAT_HASH_TABLE_ERROR_2 — The value “ldkey” = # is too small. “ldkey” is calculated as $Wk_Params(0) * \text{pow}(10, N_Attempts-1)$ 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.

CAT_GLM Function

Analyzes categorical data using logistic, Probit, Poisson, and other generalized linear models.

Usage

result = CAT_GLM(*n_class*, *n_continuous*, *model*, *x*)

Input Parameters

n_class — Number of classification variables.

n_continuous — Number of continuous variables.

model — Model used to analyze the data. The six models are as follows:

model	Relationship*	PDF of Response Variable
0	Exponential	Poisson
1	Logistic	Negative Binomial
2	Logistic	Logarithmic
3	Logistic	Binomial
4	Probit	Binomial
5	Log-log	Binomial

* Relationship between the parameter, θ or λ , and a linear model of the explanatory variables, $X\beta$.

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.

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*.

Returned Value

result — An integer value indicating the number of estimated coefficients in the model.

Input Keywords

Double — If present and nonzero, double precision is used.

Ifreq — Column number *Ifreq* in *x* containing the frequency of response for each observation.

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.

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 follows:

model	Parameter	Meaning of parameter (<i>i</i>)(<i>Ipar</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.

Default: When *model* \neq 2, each observation is assumed to have a parameter value of 1. When *model* = 2, this parameter is not referenced.

Eps — The convergence criterion. Convergence is assumed when the 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

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

No_Intercept — If present and nonzero, there is no intercept in the model. By default, the intercept is automatically included in the model.

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 *Indicies_Effects* must be used together.

Indicies_Effects — One-dimensional index array of length *Var_Effects*(0) + *Var_Effects*(1) + ... + *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 *Indicies_Effects* and *Var_Effects* must be used together.

Init_Est — One-dimensional array of length *n_coef_input* containing initial estimates of the parameters (*n_coef_input* can be completed by REGRES-SORS). By default, unweighted linear regression is used to obtain initial 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*

Output Keywords

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.

Class_Vals — Named variable into which an 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.

model	Statistic
0	Coefficient Estimate.
1	Estimated standard deviation of the estimated coefficient.
2	Asymptotic normal score for testing that the coefficient is zero.
3	The <i>p</i> -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.

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.

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 of length `n_coefficients - 1` otherwise.

Case_Analysis — Named variable into which a two-dimensional array of size `n_observations` by 5 containing the case analysis is stored.

Column	Statistic
0	Predicted mean for the observation if <code>model = 0</code> . 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.

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.

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.

Obs_Status(i)	Status of observation
0	Observation <i>i</i> is in the likelihood
1	Observation <i>i</i> cannot be in the likelihood because it contains at least one missing value in <i>x</i> .
2	Observation <i>i</i> is not in the likelihood. Its estimated parameter is infinite.

Remarks

- 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 REGRESSORS (Chapter 2, *Regression*).

4. 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.
5. 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

Function CAT_GLM uses iteratively reweighted 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 CAT_GLM are:

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	$f(y) = (1 - \theta)^y / (y \ln \theta)$	$\theta = \frac{\exp(\omega + \eta)}{1 + \exp(\omega + \eta)}$

$$3 \quad f(y) = \binom{N}{y} \theta^y (1-\theta)^{N-y} \quad \theta = \frac{\exp(\omega + \eta)}{1 + \exp(\omega + \eta)}$$

$$4 \quad f(y) = \binom{N}{y} \theta^y (1-\theta)^{N-y} \quad \theta = \Phi(\omega + \eta)$$

$$5 \quad f(y) = \binom{N}{y} \theta^y (1-\theta)^{N-y} \quad \theta = 1 - \exp(-\exp(\omega + \eta))$$

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 function 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

$$\Delta\beta = (\sum | \Psi''(\gamma_i) | x_i x_i^T)^{-1} \sum \Psi'(\gamma_i) x_i$$

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

$$l_i'(\hat{\gamma}_i)^T l''(\hat{\gamma})^{-1} l_i'(\hat{\gamma}_i)$$

This quantity is a one-step approximation to the change in the 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 function 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 1

The first example is from Prentice (1976) and involves the 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 given as follows:

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

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 $\theta = 0.5$.

```

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, "          Resid-
ual          ", $
          "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 = 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

Coefficient	Standard Error	Asymptotic Z-statistic	Asymptotic P-value
-60.76	5.21	-11.66	0.00
34.30	2.92	11.76	0.00

Covariate Means = 1.793

Case Analysis

Standardized		Residual		
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

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.

Nonparametric Statistics

Contents of Chapter

One sample tests - Nonparametric Statistics

Sign test	SIGNTEST Function
Wilcoxon rank sum test	WILCOXON Function
Noehter's test for cyclical trend	NCTRENDS Function
Cox and Stuarts' sign test for trends in location and dispersion	CSTRENDS Function
Tie statistics	TIE_STATS Function

Two or more samples

Kruskal-Wallis test	KW_TEST Function
Friedman's test	FRIEDMANS_TEST Function
Cochran's Q test	COCHRANQ Function
K-sample trends test	KTRENDS Function

Introduction

Much of what is considered nonparametric statistics is included in other chapters. Topics of possible interest in other chapters are: nonparametric measures of

location and scale (Chapter 1: *Basic Statistics*), nonparametric measures in a contingency table (Chapter 5: *Categorical and Discrete Data Analysis*), measures of correlation in a contingency table (Chapter 3: *Correlation and Covariance*), and tests of goodness of fit and randomness (Chapter 7: *Tests of Goodness of Fit*)

Missing Values

Most routines described in this chapter automatically handle missing values (NaN, “Not a Number”; see the introduction of this manual).

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 function WILCOXON (page 300), 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.

SIGNTEST Function

Performs a sign test.

Usage

result = SIGNTEST(*x*)

Input Parameters

x — One-dimensional array containing the input data.

Returned Value

result — 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.

Input Keywords

Double — If present and nonzero, double precision is used.

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

Output Keywords

N_Pos_Dev — Number of positive differences $x(j - 1) - \textit{Percentile}$, for $j = 1, 2, \dots, \text{N_ELEMENTS}(x)$.

N_Zero_Dev — Number of zero differences (ties) $x(j - 1) - \textit{Percentile}$, for $j = 1, 2, \dots, \text{N_ELEMENTS}(x)$.

Discussion

Function SIGNTEST 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, 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 $\text{N_ELEMENTS}(x)$ (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 $q = c$.

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 = ", 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 = SIGNTEST(x, Percentage = 0.75, $
      Percentile = 0, N_Pos_Dev = np, $
      N_Zero_Dev = nz)
PM, probability, Title = "Probability"
Probability
0.922543
PM, np, $
      Title = "Number of Positive Deviations"
Number of Positive Deviations
12
PM, nz, Title = "Number of Ties"
```

Number of Ties

0

WILCOXON Function

Performs a Wilcoxon rank sum test or a Wilcoxon signed rank test.

Usage

result = WILCOXON(*x1* [, *x2*])

Input Parameters

x1 — One-dimensional array containing the first sample.

x2 — (Optional) One-dimensional array containing the second sample.

Returned Value

result — 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.

Input 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.

Output Keywords

Stats — Named variable into which the one-dimensional array of length 10 containing the statistics below is stored.

If a Wilcoxon rank sum test is performed:

Row	Statistics
0	Wilcoxon W statistic (the sum of the ranks of the x observations) adjusted for ties in such a manner that W is as small as possible
1	$2 \times E(W) - W$, where $E(W)$ is the expected value of W
2	probability of obtaining a statistic less than or equal to $\min\{W, 2 \times E(W) - W\}$
3	W statistic adjusted for ties in such a manner that W is as large as possible
4	$2 \times E(W) - W$, where $E(W)$ is the expected value of W , adjusted for ties in such a manner that W 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 W is as large as possible
6	W 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
8	standard normal score associated with <i>Stats</i> (6)
9	two-sided p-value associated with <i>Stats</i> (6)

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 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.

Row	Statistics
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(6)</i> 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.

If Two Positional Arguments Are Supplied

Function WILCOXON 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. Function WILCOXON 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 following tests, 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.
2	$H_0 : Pr(x1 < x2) \leq 0.5$ $H_0 : E(x1) \geq E(x2)$	$H_1 : Pr(x1 < x2) > 0.5$ $H_1 : E(x1) < E(x2)$	Reject if <i>Stats</i> (6) is too small.
3	$H_0 : Pr(x1 < x2) \geq 0.5$ $H_0 : E(x1) \leq E(x2)$	$H_1 : Pr(x1 < x2) < 0.5$ $H_1 : E(x1) > E(x2)$	Reject if <i>Stats</i> (6) is too large.

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

Function WILCOXON 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 WILCOXON. A general reference for the methods used is Conover (1980).

Routine 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:

- $H_0 : M \leq 0 \quad H_1 : M > 0$
Reject if *stats*(0) [or *stats*(4)] is too large.
- $H_0 : M \geq 0 \quad H_1 : M < 0$
Reject if *stats*(1) [or *stats*(5)] is too large.
- $H_0 : M = 0 \quad H_1 : M \neq 0$
Reject if *stats*(2) [or *stats*(6)] is too small. Alternatively, if an asymptotic test is desired, reject if $2 * \textit{stats}$ (3) [or $2 * \textit{stats}$ (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 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.

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 = WILCOXON(x1, x2, Stats = stats)
% WILCOXON: Warning: AT_LEAST_ONE_TIE.
    At least one tie is detected between the
    samples.
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.

```
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 = WILCOXON(x1, x2, Stats = stats)
% WILCOXON: Warning: AT_LEAST_ONE_TIE.
  At least one tie is detected between the
  samples.

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; 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 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.

```

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 = WILCOXON(x, Fuzz = 0.0001, Stats = stats)
% WILCOXON: Warning: STAT_NOBS_LT_50
'n-observations' = 7. The number of observations is less than
  50, and exact tables should be referenced for probabilities.

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

```

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.

NCTRENDS Function

Performs the Noether test for cyclical trend.

Usage

result = NCTRENDS(*x*)

Input Parameters

x — One-dimensional array containing the data in chronological order.

Returned Value

result — 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).

Input 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.

Output Keywords

Nstat — Named variable into which the one-dimensional array of length 6 containing the statistics below is stored:

Statistics

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 <i>Nstat</i> (0).

Statistics

Nstat (2)	The number of nonmonotonic sequences where tied threesomes are counted as nonmonotonic
Nstat (3)	he 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 <i>Nstat(0)</i> sequences.
Nstat (5)	The number of tied sequences found in forming the <i>Nstat(2)</i> and <i>Nstat(3)</i> sequences. A sequence is called a tied sequence if the middle element is tied with either of the two other elements.

Nmissing — Named variable into which the number of missing values in *x* is stored.

Discussion

Routine 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 incremented.

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:

$$H_0 : q = \Pr(X_i > X_{i-1} > X_{i-2}) + \Pr(X_i < X_{i-1} < X_{i-2}) \leq 1/3 \quad H_1 : q > 1/3$$

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 distri-

bution.

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.

```
RANDOMOPT, set = 123457
x = RANDOM(1000, /Uniform)
pval = NCTRENDS(x, Nstat = nstat)
PM, pval
    0.697881
    0.697881
    0.697881
PM, nstat
    333
    107
    107
    107
    0
    0
```

CSTRENDS Function

Performs the Cox and Stuart sign test for trends in location and dispersion.

Usage

result = CSTRENDS(*x*)

Input Parameters

x — One-dimensional array containing the data in chronological order.

Returned Value

result — One-dimensional array of length 8 containing the probabilities.

The first four elements of *result* are computed from two groups of observa-

tions.

I result(I)

- 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).

Input 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: $Fuzz = 0.0$.

Output Keywords

Nstat — Named variable into which the one-dimensional array of length 8 containing the statistics below is stored:

I *Nstat*(I)

- 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).

Nmissing — Named variable into which the number of missing values in *x* is stored.

Discussion

Function CSTRENDS 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 $k = Dispersion(0)$ 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 = CSTRENDS(x, Nstat = nstat)

% CSTRENDS: Warning: STAT_AT_LEAST_ONE_TIE
           At least one tie is detected between the samples.

PM, nstat, Title = "          NSTAT"
           NSTAT
              0
             17
              1
             18
              0
             12
              0
             12

PM, pstat, Title = "          PSTAT"
           PSTAT
           0.999996
           7.24792e-05
           1.00000
           3.81470e-06
           1.00000
```

```

0.000244141
    1.00000
0.000244141
pstat = CSTRENDS(x, Nstat = nstat, Dispersion = [k, ids])
% CSTRENDS: Warning: STAT_AT_LEAST_ONE_TIE
    At least one tie is detected between the samples.
PM, nstat, Title = "          NSTAT"
    NSTAT
        4
        3
        2
        9
        4
        2
        0
        6
PM, pstat, Title = "          PSTAT"
    PSTAT
    0.253906
    0.910156
    0.746094
    0.500000
    0.343750
    0.890625
    0.343750
    0.890625

```

TIE_STATS Function

Computes tie statistics for a sample of observations.

Usage

result = TIE_STATS(*x*)

Input Parameters

x — One-dimensional array containing the observations. *x* must be ordered monotonically increasing with all missing values removed.

Returned Value

result — One-dimensional array of length 4 containing the tie statistics.

$$\text{result}(0) = \sum_{j=1}^{\tau} [t_j(t_j - 1)] / 2$$

$$\text{result}(1) = \sum_{j=1}^{\tau} [t_j(t_j - 1)(t_j + 1)] / 12$$

$$\text{result}(2) = \sum_{j=1}^{\tau} t_j(t_j - 1)(2t_j + 5)$$

$$\text{result}(3) = \sum_{j=1}^{\tau} t_j(t_j - 1)(t_j - 2)$$

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.

Input 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

Function TIE_STATS 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 = TIE_STATS(x, Fuzz = fuzz)
PRINT, tstat
      4.00000      2.50000      84.0000      6.00000
```

KW_TEST Function

Performs a Kruskal-Wallis test1 for identical population medians.

Usage

result = KW_TEST(*n*, *y*)

Input Parameters

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.

Returned Value

result — One-dimensional array of length 4 containing the Kruskal-Wallis statistics.

I *result*(I)

- 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

Input 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 function `KW_TEST` generalizes the Wilcoxon two-sample test computed by function `WILCOXON` (page 300) 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 function `KW_TEST` 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 = KW_TEST(n, y, Fuzz = fuzz)
% KTRENDS: Warning: <unknown error>
      Error code 30046.
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
```

***FRIEDMANS_TEST* Function**

Performs Friedman's test for a randomized complete block design.

Usage

result = FRIEDMANS_TEST(*y*)

Input Parameters

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.

Returned Value

results — The Chi-squared approximation of the asymptotic p -value for Friedman's two-sided test statistic.

Input Keywords

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: $Fuzz = 0.0$.

Alpha — Critical level for multiple comparisons. **Alpha** should be between 0 and 1 exclusive.

Default: $Alpha = 0.05$.

Output Keywords

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.

I **Stats(I)**

- 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.

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.

Discussion

Function FRIEDMANS_TEST 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 (*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 *Stats(3)*.

If the F distribution is used in place of the chi-squared distribution, then the usual oneway analysis of variance F -statistic computed on the ranks is used. This statistic, reported in *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. *Stats(4)* is the asymptotic probability of obtaining a larger F random variable. (If $A = B$, *Stats(0)* and *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 $\alpha = \textit{Alpha}$ 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 (*Stats(2)*) is used to test the same null hypothesis as the Friedman test but is sensitive to a mono-

tonic 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 the random variables within each block is equally likely. The alternative is that at least one of the treatments 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 = FRIEDMANS_TEST(y, Fuzz = fuzz, Diff = diff, $
                  Sum_Rank = sr, Stats = stat)

PM, stat, Title = "STATS"

STATS
      8.28000
      4.92857
     111.000
```

```

0.0405658
0.0185906
0.984954
PM, diff, Title = "DIFF"
DIFF
6.65638
PM, sr, Title = "Sum_Rank"
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.

COCHRANQ Function

Performs a Cochran Q test for related observations.

Usage

result = COCHRANQ(*x*)

Input Parameters

x — Two-dimensional array containing the matrix of dichotomized data.

Returned Value

result — The p -value for the Cochran Q statistic.

Input Keywords

Double — If present and nonzero, double precision is used.

Output Keywords

Q — Named variable into which the Cochran's Q statistic is stored.

Discussion

Function COCHRANQ 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 = COCHRANQ(x)
PRINT, "pq =", pq
pq = 0.000240266
```

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.

KTRENDS Function

Performs a k-sample trends test against ordered alternatives.

Usage

result = KTRENDS(*n*, *y*)

Input Parameters

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.

Returned Value

result — One-dimensional array of length 17 containing the test results.

I *result*(I)

- 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.

Input Keywords

Double — If present and nonzero, double precision is used.

Discussion

Function KTRENDS performs a *k*-sample trends test against ordered alterna-

tives. 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, \dots , 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 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 = 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

```


Goodness of Fit

Contents of Chapter

General Goodness-of-fit tests

Chi-squared goodness-of-fit test	CHISQTEST Function
Shapiro-Wilk W test for normality	NORMALITY Function
One-sample continuous data Kolmogorov-Smirnov	KOLMOGOROV1 Function
Two-sample continuous data Kolmogorov-Smirnov	KOLMOGOROV2 Function
Mardia's test for multivariate normality	MVAR_NORMALITY Function

Tests for Randomness

Runs test, Paris-serial test, d^2 test or triplets tests	RANDOMNESS_TEST Function
--	--------------------------

Introduction

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. The user supplies 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.

CHISQTEST Function

Performs a chi-squared goodness-of-fit test.

Usage

result = CHISQTEST(*f*, *n_categories*, *x*)

Input Parameters

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.

Returned Value

result — The *p*-value for the goodness-of-fit chi-squared statistic.

Input Keywords

Double — If present and nonzero, double precision is used.

N_Params_Estimated — Number of parameters estimated in computing the cumulative distribution function.

Equal_Cutpoints — If present and nonzero, equal probability cutpoints are used. Keywords *Equal_Cutpoints* and *Cutpoints* cannot be used together.

Cutpoints — Specifies the named variable containing user-defined cutpoints to be used by CHISQTEST. Keywords *Cutpoints* and *Equal_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.

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.

Output Keywords

Used_Cutpoints — Specifies the named variable into which the cutpoints to be used by CHISQTEST are stored.

Chi_Squared — Named variable into which the chi-squared test statistic is stored.

Df — Named variable into which the degrees of freedom for the chi-squared goodness-of-fit test are stored.

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.

Discussion

Function CHISQTEST 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 the user-defined function f . Because the user is 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, equiprobable intervals are computed by CHISQTEST, but intervals that are not equiprobable 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 CHISQTEST cannot determine the discrete elements in discrete distributions.

By default, the lower and upper endpoints of the first and last intervals are $-\infty$ and $+\infty$. 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 by the user, the tally is made in the interval to which x_i belongs using the endpoints specified by the user.
- If the cutpoints are determined by 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

[·]

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

The user 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 PV-WAVE function that calls the CDF, and then pass the name of this user-defined function for f .

Example

This example illustrates the use of CHISQTEST on a randomly generated sample from the normal distribution. One-thousand randomly generated observations are tallied into 10 equiprobable intervals. In this example, the null hypothesis is not rejected.

```
.RUN
    ; Define the hypothesized, cumulative distribution function.

- FUNCTION user_cdf, k
- RETURN, NORMALCDF(k)
- END

RANDOMOPT, Set = 123457
x = RANDOM(1000, /Normal)
    ; Generate normal deviates.

p_value = CHISQTEST("user_cdf", 10, x)
    ; Perform chi-squared test.

PM, p_value
```

```
    ; Output the results.  
0.154603
```

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.

NORMALITY Function

Performs a test for normality.

Usage

result = NORMALITY(*x*)

Input Parameters

x — One-dimensional array containing the observations.

Returned Value

result — 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.

Input Keywords

Double — If present and nonzero, double precision is used.

N_cat — An integer specifying number of cells into which the observations are to be tallied. Keywords *N_cat*, *Df*, and *Chisq* must be used together and indicate that the chi-squared goodness-of-fit test is to be performed.

Output Keywords

Chisq — Specifies a variable into which the chi-square statistic is stored. Keywords *N_cat*, *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 *N_cat*, *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

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.

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 *N_cat* finds the number of intervals into which the observations are to be divided. The intervals are equiprobable 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 function 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).

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. For this example, the data is stored in ASCII file *data.dat* and read using procedure RMF. The file *data.dat* contains the following data:

```
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
```

```
OPENR, unit, 'data.dat', /Get_Lun
RMF, unit, x, 50, 1
CLOSE, unit
p = 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.

```
OPENR, unit, 'data.dat', /Get_Lun
RMF, unit, x, 50, 1
CLOSE, unit
p = NORMALITY(x, Shapiro_Wilk = sw)
PRINT, "p-Value = ", p
p-Value =          0.230858
PRINT, "Shapiro Wilk W Statistic = ", sw
Shapiro Wilk W Statistic =          0.964217
```

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.

KOLMOGOROV1 Function

Performs a Kolmogorov-Smirnov one-sample test for continuous distributions.

Usage

result = KOLMOGOROV1(*f*, *x*)

Input Parameters

f — Scalar string specifying a user-supplied function to compute the cumulative distribution function (CDF) at a given value. Parameter *f* accepts the following parameter and returns the computed function value at this point:

y — Point at which the function is to be evaluated.

x — One-dimensional array containing the observations.

Returned Value

result — One-dimensional array of length 3 containing Z , p_1 , and p_2 .

Input Keywords

Double — If present and nonzero, double precision is used.

Output Keywords

Differences — Named variable into which an array containing D_n , D_n^+ , D_n^- is stored.

Nmissing — Named variable into which the number of missing values is stored.

Discussion

The function KOLMOGOROV1 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 user-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 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).

```
FUNCTION l_cdf, x
  mean = 0.5
  std = 0.2886751
  z = (x - mean)/std
  val = NORMALCDF(z)
  RETURN, val
END

RANDOMOPT, set = 123457
x = RANDOM(100, /Uniform)
stats = KOLMOGOROV1("l_cdf", x, Differences = d, $
                  Nmissing = nm)

PRINT, "D =", d(0)
D = 0.147083
PRINT, "D+ =", d(1)
```

```
D+ =      0.0809559
PRINT, "D- =", d(2)
D- =      0.147083
PRINT, "Z  =", stats(0)
Z   =      1.47083
PRINT, "Prob greater D one sided =", stats(1)
Prob greater D one sided =      0.0132111
```

KOLMOGOROV2 Function

Performs a Kolmogorov-Smirnov two-sample test.

Usage

result = KOLMORGOROV2(*x*, *y*)

Input Parameters

x — One-dimensional array containing the observations from sample one.

y — One-dimensional array containing the observations from sample two.

Returned Value

result — One-dimensional array of length 3 containing Z , p_1 , and p_2 .

Input Keywords

Double — If present and nonzero, double precision is used.

Output Keywords

Differences — Named variable into which a one-dimensional array containing D_n , D_n^+ , D_n^- is stored.

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

Function KOLMOGOROV2 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 $n_{\text{observations}_x} = N_{\text{ELEMENTS}}(x)$ and $n_{\text{observations}_y} = N_{\text{ELEMENTS}}(y)$, then the exact p -values are computed for the two-sided test when $n_{\text{observations}_x} * n_{\text{observations}_y}$ 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 $n = n_{\text{observations}_x} - N_{\text{missing}_x}$ and $m = n_{\text{observations}_y} - N_{\text{missing}_y}$, and let the corresponding population distribution functions be denoted by $F(x)$ and $G(y)$, respectively. Then, the hypotheses tested by KOLMOGOROV2 are as follows:

- $H_0: F(x) = G(x)$ $H_1: F(x) \neq G(x)$
- $H_0: F(x) \leq G(x)$ $H_1: F(x) > G(x)$
- $H_0: F(x) \geq G(x)$ $H_1: F(x) < G(x)$

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 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.

```
RANDOMOPT, set = 123457
x = RANDOM(100, /Uniform)
y = RANDOM(60, /Uniform)
stats = KOLMOGOROV2(x, y, Differences = d, Nmissingx = nmx, $
                Nmissingy = nmy)

PRINT, "D =", d(0)
D =      0.180000
PRINT, "D+ =", d(1)
D+ =     0.180000
PRINT, "D- =", d(2)
D- =     0.0100001
PRINT, "Z =", stats(0)
Z =      1.10227
PRINT, "Prob greater D one sided =", stats(1)
Prob greater D one sided =      0.0720105
PRINT, "Prob greater D two sided =", stats(2)
Prob greater D two sided =      0.144021
PRINT, "Missing X =", nmx
Missing X =      0
PRINT, "Missing Y =", nmy
Missing Y =      0
```

MVAR_NORMALITY Function

Computes Mardia's multivariate measures of skewness and kurtosis and tests

for multivariate normality.

Usage

result = MVAR_NORMALITY(*x*)

Input Parameters

x — Two-dimensional array containing the 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*).

Returned Value

result — One-dimensional array of size 13 containing output statistics

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 p -value corresponding to *result*(8)
- 10 Mardia and Foster's standard normal score for kurtosis
- 11 Mardia's S_W statistic based upon *result*(4) and *result*(10)
- 12 p -value for *result*(11)

Input 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.

Weights — One-dimensional array containing the weights. *Weights* must be non-negative. Default assumes all *Weights* equal one.

Output Keywords

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.

Nmissing — Named variable into which the number of rows of data in x containing any missing values (NaN) is stored.

Means — Named variable into which a one-dimensional array of length $N_ELEMENTS(x(0,*))$ containing the sample means 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.

Discussion

Function `MVAR_NORMALITY` computes Mardia's (1970) measures $b_{1,p}$ and $b_{2,p}$ of multivariate skewness and kurtosis, respectively, for $p = N_ELEMENTS(x(0,*))$. 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 nor-

mal scores obtained from $b_{1,p}$ and $b_{2,p}$ are computed. On the order of np^3 , operations are re-quired 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 the univariate measures when $p = \text{N_ELEMENTS}(x(0,*)) = 1$. 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 RANDOM (Chapter 12, *Random Number Generation*). The skewness and kurtosis statistics are then computed for these observations.

```
m = 150
n = 5
RANDOMOPT, set = 123457
x = FLTARR(n, m)
x(*) = RANDOM(m*n, /Normal)
x = TRANSPOSE(x)
stats = MVAR_NORMALITY(x, Sum_Weights = sw, Sum_Freq = sf, $
                    Means = means, R_Matrix = r_mat)
PRINT, "Sum of Frequencies =", sf, Format = "(A25, I4)"
      Sum of Frequencies = 150
PRINT, "Sum of the weights =", sw, Format = "(A25, F8.3)"
      Sum of the weights = 150.000
FOR i = 0, 12 DO $
      PM, i, stats(i), Format = "(I5, F10.2)"
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

***RANDOMNESS_TEST* Function**

Performs a test for randomness.

Usage

result = RANDOMNESS_TEST(*x*, *n_run*)

Input Parameters

x — One-dimensional array containing the data.

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.

Returned Value

result — The probability of a larger chi-squared statistic for testing the null hypothesis of a uniform distribution.

Input Keywords

Double — If present and nonzero, double precision is used.

Pairs_Lag — The lag to be used in computing the pairs statistic. Keywords *Pairs_Lag* and *Pairs_Counts* must be used together.

Output Keywords

Exactly one of these options 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

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.

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.

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.

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

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.

If *Runs_Counts* is specified:

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. This keyword is optional if *Runs_Counts* is used.

If *Pairs_Counts*, *Dsquare_Counts*, or *Dcube_Counts* is specified:

Expect — Named variable into which the expected number of counts for each cell is stored. 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.

Discussion

Runs Up Test

Function RANDOMNESS_TEST 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, the user 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.

d^2 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

$$\Pr(D^2 \leq d^2) = \frac{1}{3} + (\pi - 2)d^2 + 4\sqrt{d^2 - 1} + 8\frac{(d^2 - 1)^{\frac{3}{2}}}{3} - \frac{d^4}{2} - 4d^2 \arctan\left(\frac{\sqrt{1 - \frac{1}{d^2}}}{\frac{1}{d}}\right)$$

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

$$\chi^2 = \sum_{i,j,k=0}^{k-1} \frac{(o_{ijk} - e)^2}{e}$$

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.

Example 1

The following example illustrates the use of the runs test on 10^4 pseudorandom 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.

```
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)
RANDOMOPT, set = 123457
x = RANDOM(nran, /Uniform)
p = 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.

```

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
RANDOMOPT, set = 123467
x = RANDOM(nran, /Uniform)
p = 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 RAN-
DOM are input to *Dsquare_Counts* in one call. In the example, the null hy-
pothesis of a uniform distribution is not rejected.

```

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)
RANDOMOPT, set = 123457
x = RANDOM(nran, /Uniform)
p = 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 RANDOM are input to *Dcube_Counts*, and tabulated in 27 equally sized cubes. In the example, the null hypothesis is not rejected.

```

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)
RANDOMOPT, set = 123457
x = RANDOM(nran, /Uniform)
p = 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

```

Time Series and Forecasting

Contents of Chapter

ARMA Models

Computes least-squares or method-of-moments estimates of parameters and optionally computes forecasts and their associated probability limits	ARMA Function
Performs differencing on a time series	DIFFERENCE Function
Perform a Box-Cox transformation	BOXCOXTRANS Function
Sample autocorrelation function	AUTOCORRELATION Function
Sample partial autocorrelation function	PARTIAL_AC Function
Lack-of-fit test based on the correlation function	LACK_OF_FIT Function
Compute estimates of the parameters of a GARCH(p,q) model	GARCH Function
Performs Kalman filtering and evaluates the likelihood function for the statespace model	KALMAN Procedure

Introduction

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 NaN (see the routine MACHINE), 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 AUTOCORRELATION (page 391), and PARTIAL_AC (page 395) may be used to diagnose the presence of nonstationarity in the data, as well as to indicate the type of transformation¹ require to induce stationarity. The family of power transformations provided by routine BOXCOXTRANS (page 387) coupled with the ability to difference the transformed data using routine DIFFERENCE (page 382) 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 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 ARMA processes defined by

$$\phi(B) (W_t - \mu) = \theta(B)A_p, \quad t \in Z$$

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_p 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 ARMA (p, q) model.

An equivalent version of the ARMA (p, q) model is given by

$$\phi(B) W_t = \theta_0 + \theta(B) A_p, \quad t \in Z$$

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 DIFFERENCE (page 382) induces stationarity, and the model is called ARIMA (AutoRegressive Integrated Moving Average). Parameter estimation is performed on the stationary time series $W_p = \nabla^d Z_t$, where $\nabla^d = (I - B)^d$ 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 function ARMA (page 366) 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 by the user 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.

ARMA Function

Computes method-of-moments or least-squares estimates of parameters for a nonseasonal ARMA model.

Usage

result = ARMA(*z*, *p*, *q*)

Input Parameters

z — One-dimensional array containing the observations.

p — Number of autoregressive parameters.

q — Number of moving average parameters.

Returned Value

result — 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.

Input Keywords

Double — If present and nonzero, double precision is used.

No_Constant — If present and nonzero, the time series is not centered about its mean. Keywords *No_Constant* and *Constant* cannot be used together.

Constant — If present and nonzero, the time series is centered about its mean. Keywords *No_Constant* and *Constant* cannot be used together.

Ar_Lags — One-dimensional array of length *p* containing the order of the non-zero autoregressive parameters. The elements of *Ar_Lags* must be greater than or equal to 1.

Default: *Ar_Lags* = [1, 2, ..., *p*]

Ma_Lags — One-dimensional array of length *q* containing the order of the non-zero moving average parameters. The elements of *Ma_Lags* must be greater than or equal to 1.

Default: *Ma_Lags* = [1, 2, ..., *q*]

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)

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: $Lgth_Backcast = 10$

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: $Tol_Backcast = 0.01 \times \text{standard deviation of } l$

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.

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

Itmax — Maximum number of iterations allowed in the nonlinear equation solver used in both the method-of-moments and least-squares algorithms.

Default: $Itmax = 200$

Mean_Est — Initial estimate of the mean of the time series z .

$$\text{Default: } Mean_Est = \sum_{t=1}^n z_t/n$$

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:

N_Predict — Maximum lead time for forecasts. Keyword *N_Predict* must be greater than zero. Keywords *Forecast* and *N_Predict* must be used together.

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

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

Output Keywords

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.

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*.

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 *Auto-*

$cov(0)$ contains the variance of the series z . Keyword *Autocov(k)* contains the autocovariance of lag k , where $k = 1, \dots, p + q + 1$.

Ss_Residual — Named variable into which the sum of squares of the random error is stored.

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.

Discussion

Function ARMA computes estimates of parameters for a nonseasonal ARMA model given a sample of observations, $\{Z_t\}$, for $t = 1, 2, \dots, n$, where $n = N_ELEMENTS(z)$. The user may choose either method of moments or least squares. The default is method of moments.

The user chooses the method-of-moments algorithm with the keyword *Moments*. The least-squares algorithm is used if *Lsq* is specified. If the user wishes to use the least-squares algorithm, the preliminary estimates are the method-of-moments estimates by default; otherwise, the user can input initial estimates by specifying keywords *Init_Est_Ar* and *Init_Est_Ma*. The following table 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 Constant (or No_Constant) Ar_Lags Ma_Lags Lgth_Backcast Tol_Backcast Tol_Convergence Init_Est_Ar Init_Est_Ma Residual Param_Est_Cov Ss_Residual	Err_Rel Itmax Mean_Estimate Autocov Forecast N_Predict Confidence Backward_Origin

Method-of-moments Estimation

Suppose the time series $\{Z_t\}$ is generated by an 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:

$$\hat{\mu} = \begin{cases} \mu & \text{for } \mu \text{ known} \\ \frac{1}{n} \sum_{t=1}^n Z_t & \text{for } \mu \text{ unknown} \end{cases}$$

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

$$\begin{aligned}\hat{\phi} &= (\hat{\phi}_1, \dots, \hat{\phi}_p)^T \\ \hat{\Sigma}_{ij} &= \hat{\sigma}(|q+i-j|), \quad i, j = 1, \dots, p \\ \hat{\sigma}_i &= \hat{\sigma}(q+i), \quad i, j = 1, \dots, p\end{aligned}$$

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

$$f_j = \sum_{i=0}^{q-j} \tau_i \tau_{i+j} - \hat{\sigma}'(j) \text{ for } j = 0, 1, \dots, q$$

$$= \begin{bmatrix} \tau_0 & \tau_1 & \dots & \dots & \tau_q \\ \tau_1 & \tau_2 & \dots & \tau_q & 0 \\ \dots & \dots & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots \\ \tau_q & 0 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} \tau_0 & \dots & \dots & \tau_q \\ 0 & \tau_0 & \dots & \tau_{q-1} \\ 0 & 0 & \dots & \dots \\ \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \tau_0 \end{bmatrix}$$

Then, the value of τ at the $(i + 1)$ -th iteration is determined by the following:

$$\tau^{i+1} = \tau^i - (T^i)^{-1} f^i$$

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:

$$\hat{\sigma}_A^2 = \begin{cases} \hat{\sigma}(0) - \sum_{i=1}^p \hat{\phi}_i \hat{\sigma}(i) & \text{for } q = 0 \\ \tau_0^2 & \text{for } q > 0 \end{cases}$$

See Box and Jenkins (1976, pp. 498–500) for a description of a function that performs similar computations.

Least-squares Estimation

Suppose the time series $\{Z_t\}$ is generated by a nonseasonal ARMA model of the form

$$\phi(B)(Z_t - \mu) = \theta(B)A_t \quad \text{for } t \in \{0, \pm 1, \pm 2, \dots\}$$

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 ARMA model is of order (p', q') , where

$$p' = l_\phi(p) \text{ and } q' = l_\theta(q).$$

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

$$S_T(\mu, \phi, \theta) = \sum^n [A_t]^2$$

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

$$S_\infty(\mu, \phi, \theta) / (2\sigma_A^2) \text{ dominates } l(\mu, \phi, \theta, \sigma_A^2).$$

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 function DIFFERENCE on page 382, together with ARMA.

Forecasting Option

The Box-Jenkins forecasts and their associated confidence intervals for a non-seasonal ARMA model are computed given a sample of $n = \text{N_ELEMENTS}(z)$ $\{Z_t\}$ for $t = 1, 2, \dots, n$.

Suppose the time series $\{Z_t\}$ is generated by a nonseasonal ARMA model of the form

$$\phi(B) Z_t = \theta_0 + \theta(B) A_t$$

$$\text{for } t \in \{0, \pm 1, \pm 2, \dots\},$$

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 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:

$$l_\phi(i) = i, \quad 1 \leq i \leq p$$

$$l_\theta(j) = j, \quad 1 \leq j \leq q$$

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

$$E \left[Z_{t+l} - \hat{Z}_t(l) \right]^2 .$$

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).

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 8-1*. The method-of-moments estimates

$$\hat{\theta}_0, \hat{\phi}_1, \hat{\phi}_2 \text{ and } \hat{\theta}_1$$

for the 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 identical normally distributed with mean zero and variance σ_A^2 .

```
temp = 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 = ARMA(z, p, q)
      ; Perform time-series analysis.

PRINT, "AR estimates:", parameters(1), parameters(2)
AR estimates: 1.24426 -0.575149

PRINT, "MA estimate :", parameters(3)
MA estimate : -0.124094
```

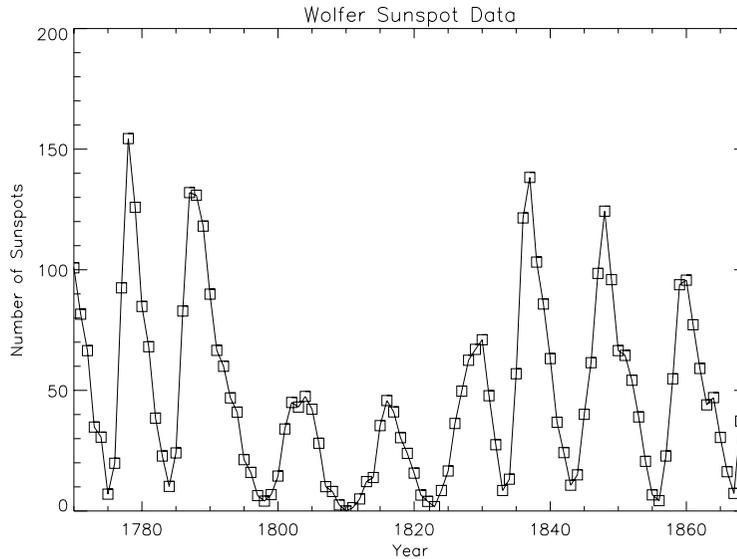


Figure 8-1 Plot of Wolfer Sunspot Data.

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 = STATDATA(2)
      ; Get the Wolfer Sunspot Data.

z = TEMP(21:120, 1)
      ; Use only 100 observations, 1770-1869.

parameters = 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)

AR estimates: 1.39257 -0.732948

PRINT, "MA estimate :", parameters(3)

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. Function ARMA computes forecasts and 95-percent confidence limits for the forecasts for an ARMA(2, 1) model fit using function 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:

```
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 = STATDATA(2)
      ; Get the Wolfer Sunspot Data.

z = TEMP(21:120, 1)
      ; Use only 100 observations, 1770-1869.

parameters = 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

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

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

```

The plot of the forecasts and the confidence limits from year 1869 are shown in [Figure 8-2](#).

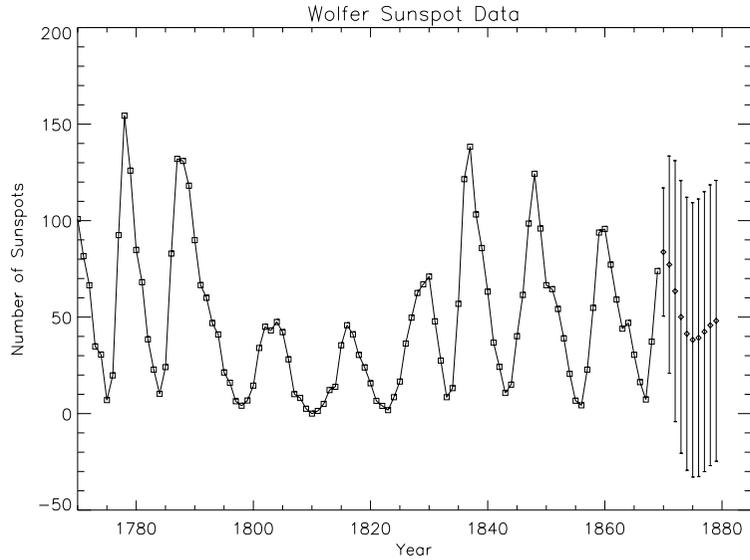


Figure 8-2 Plot of sunspot data with predicted values and confidence bands.

DIFFERENCE Function

Differences a seasonal or nonseasonal time series.

Usage

result = DIFFERENCE(*z*, *periods*)

Input Parameters

z — One-dimensional array containing the time series.

periods — One-dimensional array containing the periods at which *z* is to be differenced.

Returned Value

result — One-dimensional array of length N_ELEMENTS (*z*) containing the differenced series.

Input Keywords

Double — If present and nonzero, double precision is used.

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

Exclude_First or

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*

Output Keywords

Num_Lost — Named variable into which the number of observations “lost” because of differencing the time series *z* is stored.

Discussion

Function DIFFERENCE 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 the function 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 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 differenc-

ing of a series enables model identification and parameter estimation in the class of homogeneous stationary ARMA.

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 8-3*. 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. Function DIFFERENCE 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(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 = DIFFERENCE(z, periods)
      ; Call 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      112.0      NaN
1      118.0      NaN
2      132.0      NaN
3      129.0      NaN
4      121.0      NaN
5      135.0      NaN
6      148.0      NaN
7      148.0      NaN
```

8	136.0	NaN
9	119.0	NaN
10	104.0	NaN
11	118.0	NaN
12	115.0	NaN
13	126.0	5.0
14	141.0	1.0
15	135.0	-3.0
16	125.0	-2.0
17	149.0	10.0
18	170.0	8.0
19	170.0	0.0
20	158.0	0.0
21	133.0	-8.0
22	114.0	-4.0
23	140.0	12.0

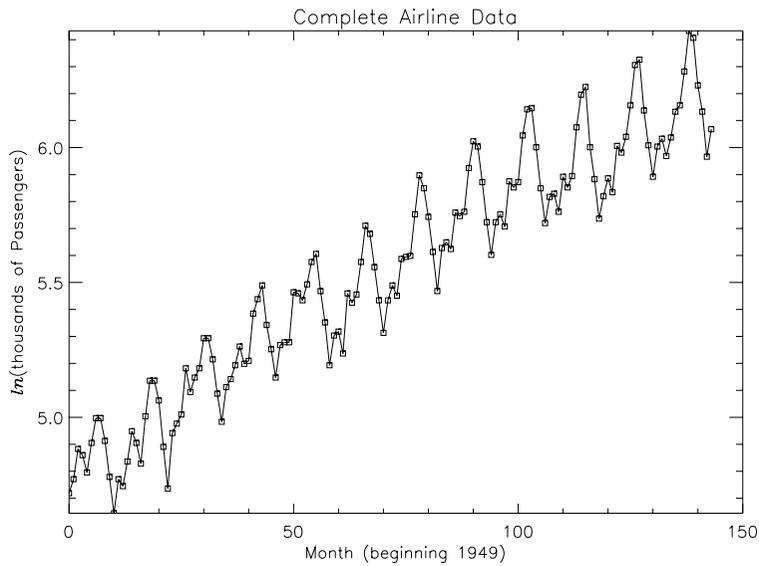


Figure 8-3 Plot of the complete data set for airline passengers.

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 = STATDATA(4)
      ; Get the data set.

z = ztemp(0:23)
periods = [1, 12]

diff = DIFFERENCE(z, periods, $
      /Exclude_First, Num_Lost = num_lost)
      ; Call DIFFERENCE.

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)      DIFFERENCE(i)"
      ; Output the results.

      i      z(i)      DIFFERENCE(i)
0      112.0      5.0
1      118.0      1.0
2      132.0     -3.0
3      129.0     -2.0
4      121.0     10.0
5      135.0      8.0
6      148.0      0.0
7      148.0      0.0
8      136.0     -8.0
9      119.0     -4.0
10     104.0     12.0

```

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.

BOXCOXTRANS Function

Performs a forward or an inverse Box-Cox (power) transformation.

Usage

result = BOXCOXTRANS(*z*, *power*)

Input Parameters

z — One-dimensional array containing the observations.

power — Exponent parameter in the Box-Cox (power) transformation.

Returned Value

result — One-dimensional array containing the transformed data.

Input Keywords

Double — If present and nonzero, double precision is used.

S — Shift parameter in the Box-Cox (power) transformation. Parameter shift must satisfy the relation $\min(z(i)) + S > 0$.

Default: $S = 0.0$.

Inverse — If present and nonzero, the inverse transform is performed.

Discussion

Function BOXCOXTRANS performs a forward or an inverse Box-Cox (power) transformation of $n = \text{N_ELEMENTS}(z)$ observations $\{Z_t\}$ for $t = 0, 1, \dots, n-1$.

The forward transformation is useful in the analysis of linear models or models with nonnormal errors or nonconstant 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 BOXCOX-TRANS 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 BOXCOXTRANS for a forward transformation of the original data using parameters λ and ξ .

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 = 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]
    ; Transform Data using Box Cox Transform
x = 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 = BOXCOTRANS(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

```

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.

AUTOCORRELATION Function

Computes the sample autocorrelation function of a stationary time series.

Usage

result = AUTOCORRELATION(*x*, *lagmax*)

Input Parameters

x — One-dimensional array containing the time series. `N_ELEMENTS(x)` must be greater than or equal to 2.

lagmax — Scalar integer containing the maximum lag of autocovariance, autocorrelations, and standard errors of autocorrelations to be computed. *lagmax* must be greater than or equal to 1 and less than `N_ELEMENTS(x)`.

Returned Value

result — One-dimensional array of length *lagmax* + 1 containing the autocorrelations 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, ..., *lagmax*.

Input Keywords

Double — If present and nonzero, double precision is used.

Xmean_In — The estimate of the mean of the time series *x*.

Se_Option — Method of computation for standard errors of the autocorrelations. Keywords *Se_Option* and *Seac* must be used together.

Se_Option	Action
1	Compute the standard errors of autocorrelation using Barlett's formula.
2	Compute the standard errors of autocorrelation using Moran's formula.

Output Keywords

Acv — Named variable into which an array of length *lagmax* + 1 containing the variance and autocovariances 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 au-

tocovariance of lag k where $k = 1, \dots, lagmax$.

Seac — Named variable into which an array of length $lagmax$ containing the standard errors of the autocorrelations of the time series x is stored. Keywords **Seac** and **Se_Option** must be used together.

Xmean_Out — Named variable into which the estimate of the mean of the time series x is stored.

Discussion

Function AUTOCORRELATION estimates the autocorrelation function of a stationary time series given a sample of $n = N_ELEMENTS(x)$ observations $\{X_t\}$ for $t = 1, 2, \dots, n$.

Let

$$\hat{\mu} = x_mean$$

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 $K = lagmax$. Note that

$$\hat{\sigma}(0)$$

is an estimate of the sample variance. The autocorrelation function $\rho(k)$ is esti-

mated by

$$\frac{\hat{\sigma}(k)}{\hat{\sigma}(0)}, \quad k = 0, 1,$$

Note that

by definition.

The standard errors of the sample autocorrelations 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

where

assumes μ is unknown. For computational purposes, the autocorrelations $r(k)$ are replaced by their estimates

for $|k| \leq K$, and the limits of summation are bounded because of the assumption that $r(k) = 0$ 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, identi-

cally 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. Function AUTOCORRELATION computes the estimated autocovariances, estimated autocorrelations, and estimated standard errors of the autocorrelations.

```

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 = STATDATA(2)
x = data(21:120,1)
result = 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

PARTIAL_AC Function

Computes the sample partial autocorrelation function of a stationary time series.

Usage

result = PARTIAL_AC(*cf*)

Input Parameters

cf — One-dimensional array containing the autocorrelations of the time series *x*.

Returned Value

result — One-dimensional array containing the partial autocorrelations of the time series *x*.

Input Keywords

Double — If present and nonzero, double precision is used.

Discussion

Function PARTIAL_AC estimates the partial autocorrelations of a stationary time series given the $K = (N_ELEMENTS(cf) - 1)$ sample autocorrelations

$$\hat{\rho}(k)$$

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 nonstationarity 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

$$\text{var}\{\hat{\phi}_{kk}\} \simeq \frac{1}{n} \quad k \geq p+1$$

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 `PARTIAL_AC` is used to compute the estimated partial autocorrelations.

```
data = STATDATA(2)
x = data(21:120,1)
result = AUTOCORRELATION(x, 20)
partial = 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

```

LACK_OF_FIT Function

Performs lack-of-fit test for a univariate time series or transfer function given the appropriate correlation function.

Usage

result = LACK_OF_FIT(*nobs*, *cf*, *npfree*)

Input Parameters

nobs — Number of observations of the stationary time series.

cf — One-dimensional array containing the correlation function.

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 $\text{lagmax} = (\text{N_ELEMENTS}(cf) - 1)$. Woodfield (1990) recommends $\text{npfree} = p + q$.

Returned Value

result — 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.

Input 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 LACK_OF_FIT may be used to diagnose lack of fit in both ARMA and transfer function models. Typical arguments for these situations are

Model	LAGMIN	LAGMAX	NPFREE
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>

Function LACK_OF_FIT 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 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 ARMA(2,1) with nonzero mean is fitted using routine ARMA (page 366). The autocorrelations of the residuals are estimated using routine AUTOCORRELATION (page 391). A portmanteau lack of fit test is computed using 10 lags with LACK_OF_FIT.

The warning message from ARMA in the output can be ignored. (See the example for routine 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 = STATDATA(2)
x = data(21:120,1)
    ; Get residuals for ARMA(2, 1) for autocorrelation/lack of fit
params = ARMA(x, p, q, /Lsq, Tol_Convergence = tc, Residual = r)
% ARMA: Warning: STAT_LEAST_SQUARES_FAILED
Least squares estimation of the parameters has failed to converge. Increase "LGTH_BACKCAST" and/or "TOL_BACKCAST" and/or "TOL_CONVERGENCE". The estimates of the parameters at the last iteration may be used as new starting values.
    ; Get autocorrelations from residuals for lack of fit test
    ; NOTE: number of observations is equal to number of residuals
corrs = AUTOCORRELATION(r, lagmax)
    ; Get lack of fit test statistic and p-value
    ; NOTE: number of observations is equal to original number of data
result = 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)"
Lack of Fit Statistic (Q) = 14.572
PRINT, "P-value (PVALUE) =", result(1), Format = "(A28, F8.4)"
    P-value (PVALUE) = 0.9761

```

GARCH Function

Compute estimates of the parameters of a GARCH(p,q) model.

Usage

result = GARCH(p , q , y , $xguess$)

Input Parameters

p — Number of autoregressive (AR) parameters

q — Number of moving average (MA) parameters

y — One-dimensional array containing the observed time series data.

$xguess$ — One-dimensional array of length $p + q + 1$ containing the initial values for the parameter array x .

Returned Value

result — One-dimensional array of length $p + q + 1$ containing the estimated values of sigma squared, the AR parameters, and the MA parameters.

Input Keywords

Double — If present and nonzero, double precision is used.

Max_Sigma — Value of the upperbound on the first element (sigma) of the array of returned estimated coefficients.

Default: $Max_Sigma = 10$.

Output Keywords

Log_Likelihood — Named variable into which the value of Log-likelihood function evaluated at the estimated parameter array x is stored.

Aic — Named variable into which the value of Akaike Information Criterion evaluated at the estimated parameter array x is stored.

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 (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 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 nonnegativity 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, GARCH(1,1) or 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 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 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 GARCH into nonlinear fashion.

The maximal likelihood method is used in estimating the parameters in 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 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 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 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 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.

```
FUNCTION SGARCH, p, q, m, x
  z = FLTARR(m + 1000)
  y0 = FLTARR(m + 1000)
  sigma = FLTARR(m + 1000)
  z = 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

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 = GARCH(p, q, y, xguess, Log_Likelihood = a, Aic = aic)
PRINT, "Sigma estimate is", result(0)
Sigma estimate is 1.27742
PRINT, "AR(1) estimate is", result(1)
AR(1) estimate is 0.230132
PRINT, "AR(2) estimate is", result(2)
AR(2) estimate is 0.375924

```

```
PRINT, "MA(1) estimate is", result(3)
MA(1) estimate is      0.312843
PRINT, "Log-likelihood function value is", a
Log-likelihood function value is      -2707.53
PRINT, "Akaike Information Criterion value is", aic
Akaike Information Criterion value is      5423.06
```

KALMAN Procedure

Performs Kalman filtering and evaluates the likelihood function for the state-space model.

Usage

`KALMAN, b, covb, n, ss, alndet`

Input/Output Parameters

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 `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 `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 `KALMAN`. In the usual case when the variance-covariance matrix is nonsingular, n equals the sum of the `N_ELEMENTS(Y)` from the invocations to `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 KALMAN. The estimate of σ^2 is given by

$$\frac{ss}{n}.$$

alnet — 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, KALMAN avoids the explicit computation of P . *alndet* must be initialized to zero before the first call to KALMAN. In the usual case when P is nonsingular, *alndet* is the natural log of the determinant of P .

Input Keywords

Y — One dimensional array containing the observations. Keywords *Y*, *Z* and *R* indicate an update step and must be used together

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

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

Tolerance — Tolerance used in determining linear dependence.

Default: Tolerance = 100*eps where eps is machine precision.

Output Keywords

V — One dimensional array of length N_ELEMENTS(*Y*) containing the one-step-ahead prediction error.

Covv — Two dimensional array of size N_ELEMENTS(Y) by N_ELEMENTS(Y) containing a matrix such that $Covv * \sigma^2$ is the variance-covariance matrix of v .

Discussion

Routine 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 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

$$\sigma^2 C_{k|j} = E(\hat{\beta}_{k|j} - b_k)(\hat{\beta}_{k|j} - b_k)^T$$

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 KALMAN computes the filtered estimate

$$\hat{\beta}_{k|k}$$

along with $C_{k|k}$. These quantities are given by the *update equations*:

$$\begin{aligned}\hat{\beta}_{k|k} &= \hat{\beta}_{k|k-1} + C_{k|k-1} Z_k^T H_k^{-1} v_k \\ C_{k|k} &= C_{k|k-1} - C_{k|k-1} Z_k^T H_k^{-1} Z_k C_{k|k-1}\end{aligned}$$

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 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 KALMAN computing the one-step-ahead estimate

$$\hat{\beta}_{k+1|k}$$

along with $C_{k+1|k}$ given by the *prediction equations*:

$$\begin{aligned}\hat{\beta}_{k+1|k} &= T_{k+1} \hat{\beta}_{k|k} \\ C_{k+1|k} &= T_{k+1} C_{k|k} T_{k+1}^T + Q_{k+1}\end{aligned}$$

If both the filtered estimates and one-step-ahead estimates are needed by the user at each time point, 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 , the prediction equations are skipped. Without keywords Y , Z , and R , the 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 , KALMAN is invoked with keywords Y , Z , and R to compute

$$\hat{\beta}_{j+1|j}$$

Subsequent invocations of KALMAN without keywords 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

$$N = \sum_{k=1}^m n_k \quad \text{and} \quad SS = \sum_{k=1}^m v_k^T H_k^{-1} v_k$$

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, KALMAN can be used in conjunction with an optimization program (see rou-

tine FMINV, PV-WAVE: IMSL Mathematics Reference, Chapter 8, “Optimization”) 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

$$L(\theta, \sigma^2; y_1, y_2, \dots, y_m) = -\frac{1}{2} N \ln \sigma^2 - \frac{1}{2} \sum_{k=1}^m \ln[\det(H_k)] - \frac{1}{2} \sigma^{-2} \sum_{k=1}^m v_k^T H_k^{-1} v_k$$

(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 nonsingular. If H_k is singular, a modification for singular distributions described by Rao (1973, pages 527–528) is used. The necessary 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 1

Routine 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

$$\begin{aligned} y_k &= b_k + e_k \\ b_{k+1} &= b_k + w_{k+1} \quad k = 1, 2, 3, 4 \end{aligned}$$

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 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 KALMAN.

Note that this example is in the form of a WAVE procedure, with the output following the procedure.

```

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
    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
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

Multivariate Analysis

Contents of Chapter

Performs a K -means (centroid) cluster analysis	K_MEANS Function
Computes principal components	PRINC_COMP Function
Extracts factor-loading estimates	FACTOR_ANALYSIS Function
Perform discriminant function analysis	DISCR_ANALYSIS Procedure

Introduction

Cluster Analysis

Function `K_MEANS` 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. Function `K_MEANS` is one implementation of the basic algorithm.

The usual course of events in K -means cluster analysis is to use `K_MEANS` to obtain the optimal clustering. The clustering is then evaluated by functions described in [Chapter 1, *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 function `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 `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, `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 quite different in assumptions, often serve the same ends. 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.

K_MEANS Function

Performs a K -means (centroid) cluster analysis.

Usage

result = K_MEANS(*x*, *seeds*)

Input Parameters

x — Two-dimensional array containing the observations to be clustered. The data value for the i -th observation of the j -th variable should be in $x(i, j)$.

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).

Returned Value

result — The cluster membership for each observation is returned.

Input Keywords

Double — If present and nonzero, double precision is used.

Weights — One-dimensional array containing the weight of each observation of matrix *x*.

Default: *Weights*(*) = 1

Frequencies — One-dimensional array containing the frequency of each observation of matrix x .

Default: $Frequencies(*) = 1$

Itmax — Maximum number of iterations.

Default: $Itmax = 30$

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: $Vars_Columns(*) = 0, 1, 2, \dots, N_ELEMENTS(x(0, *)) - 1$

Output Keywords

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.

Counts_Cluster — Named variable into which an array containing the number of observations in each cluster is stored.

Discussion

Function K_MEANS 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 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 nonmissing 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 nonmissing 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 function 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 = 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 = 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

```

Warning Errors

STAT_NO_CONVERGENCE — Convergence did not occur.

PRINC_COMP Function

Computes principal components.

Usage

result = PRINC_COMP(*covariances*)

Input Parameters

covariances — Two-dimensional square matrix containing the covariance or correlation matrix.

Returned Value

result — One-dimensional array containing the eigenvalues of *covariances* ordered from largest to smallest.

Input Keywords

Double — If present and nonzero, double precision is used.

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.

Output Keywords

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.

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.

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.

Df — Named variable into which the number of degrees of freedom in *covariances* is stored. Keywords *Df* and *Stdev* must be used together.

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

Function PRINC_COMP 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.

Example 1

In this example, principal components are computed for a nine-variable covariance matrix. This example opens a file, *cov.dat*, and reads in the covariance matrix using the RMF procedure. The file *cov.dat* contains the following data:

```
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
```

```
OPENR, unit, 'cov.dat', /Get_Lun
```

```
RMF, unit, covariances, 9, 9
```

```
CLOSE, unit
```

```
values = 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.

```
OPENR, unit, 'cov.dat', /Get_Lun
RMF, unit, covariances, 9, 9
CLOSE, unit

values = PRINC_COMP(covariances, $
  /Corr_Matrix, $
  Eigenvectors = ev, $
  Stdev = stdev, $
  Df = 100, $
  Cum_Percent = cp, $
  Comp_Resp_Corr = 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

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 *Eigenvectors*(#) = #. One or more eigenvalues much less than zero are computed. The matrix *Covariances* is not nonnegative definite. In order to continue computations of *Eigenvectors* and *Correlations*, these eigenvalues are treated as zero.

STAT_FAILED_TO_CONVERGE — Iteration for the eigenvalue failed to converge in 100 iterations before deflating.

FACTOR_ANALYSIS Function

Extracts initial factor-loading estimates in factor analysis.

Usage

result = FACTOR_ANALYSIS(*covariances*, *n_factors*)

Input Parameters

covariances — Two-dimensional array containing the variance-covariance or correlation matrix.

n_factors — Number of factors in the model.

Returned Value

result — A two-dimensional array containing the matrix of factor loadings.

Input Keywords

Double — If present and nonzero, double precision is used.

NOTE Keywords *Max_Likelihood*, *Princ_Comp*, *Princ_Factor*, *Unwgt_Lsq*, *Gen_Lsq*, *Image*, and *Alpha* cannot be used together.

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.

Princ_Comp — If present and nonzero, the principal component (principal component model) is used to obtain the estimates.

Princ_Factor — If present and nonzero, the principal factor (common factor model) is used to obtain the estimates.

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.

Gen_Lsq — If present and nonzero, the generalized least-squares (common factor model) method is used to obtain the estimates.

Image — If present and nonzero, the image-factor analysis (common factor model) method is used to obtain the estimates.

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.

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*

Itmax — Maximum number of iterations in the iterative procedure.

Default: *Itmax* = 60

Max_Steps — Maximum number of step halvings allowed during any one iteration.

Default: *Max_Steps* = 10

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

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

Output Keywords

Unique_Var_Out — One-dimensional array of length $N_ELEMENTS(covariances(0, *))$ containing the estimated unique variances.

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.

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.

Tucker_Coef — Named variable into which the Tucker reliability coefficient is stored.

Iters — Named variable into which the number of iterations is stored.

F_Min — Named variable into which the value of the function minimum is stored.

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.

Discussion

Function S computes unrotated factor loadings in exploratory factor-analysis models. Models available in 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 $\hat{\Gamma}$ and the diagonal matrix $\hat{\Delta}$ 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 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 FACTOR_ANALYSIS using the keyword *Unique_Var_In*. In practice, the estimates of these parameters are produced by 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, the user 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, the user can either initialize the unique error variances or allow FACTOR_ANALYSIS to compute initial estimates. Unlike the principal factor method, 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:

$$\rho = \frac{mM_o - mM_k}{mM_o - 1}$$

$$m = d - \frac{2p + 5}{6} - \frac{2k}{6}$$

$$M_o = \frac{-\ln(|S|)}{p(p-1)/2}$$

$$M_k = \frac{\phi}{((p-k)^2 - p - k)/2}$$

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 $d = \text{Df}$.

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. Function `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.)

Example 1

In this example, factor analysis is performed for a nine-variable matrix using the default method of unweighted least squares. This example opens a file, *cov.dat*, and reads in the covariance matrix using procedure RMF. The file *cov.dat* contains the following data:

```
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
```

```
OPENR, unit, 'cov.dat', /Get_Lun
RMF, unit, covariances, 9, 9
CLOSE, unit

n_factors = 3

a = FACTOR_ANALYSIS(cov, 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.

```
OPENR, unit, 'cov.dat', /Get_Lun
RMF, unit, covariances, 9, 9
```

```

CLOSE, unit
n_factors = 3
a = FACTOR_ANALYSIS(cov, 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

```

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.

DISCR_ANALYSIS Procedure

Performs a linear or a quadratic discriminant function analysis among several known groups.

Usage

DISCR_ANALYSIS, *x*, *n_groups*

Input Parameters

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*.

n_groups — Number of groups in the data.

Input Keywords

Double — If present and nonzero, double precision is used.

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, ..., *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.

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	leaving-out-one
5	quadratic	pooled, group	leaving-out-one
6	linear	pooled	leaving-out-one

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_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.

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.

Output Keywords

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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

Function 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. The linear discriminant function coefficients are also computed.

Covariance matrices are defined as follows: Let N_i denote the sum of the frequencies of the 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

$$C^{-1} = \frac{1 - 2p^2 + 3p - 1}{6(p+1)(k-1)} \left(\frac{k}{\sum_{i=1}^k n_i} - \frac{1}{\sum_j n_j} \right)$$

where p is the number of variables.

The estimated posterior probability of each observation x belonging to group i 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

$$\hat{q}_i(x) = \frac{\exp(-0.5D_i^2(x))}{\sum_{j=1}^k \exp(-0.5D_j^2(x))}$$

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 linear discrimination with equal prior probabilities on Fisher's (1936) iris data.

```
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 = STATDATA(3)
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
```

```
nrmisss = 0
```

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 #.

Survival Analysis

Contents of Chapter

Analyzes survival data using a generalized linear model and estimates using various parametric modes..... [SURVIVAL_GLM Function](#)

Introduction

The routine described in this chapter have 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. Routine SURVIVAL_GLM (page 450) fits any of several generalized linear models, and computes estimates of survival probabilities based on the same models.

SURVIVAL_GLM Function

Analyzes censored survival data using a generalized linear model and estimates survival probabilities and hazard rates for the various parametric models.

Usage

```
result = SURVIVAL_GLM(n_class, n_continuous, model, x)
```

Input Parameters

n_class — Number of classification variables.

n_continuous — Number of continuous variables.

model — Specifies the model used to analyze the data.

model	PDF of the Response Variable
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.

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 the optional parameters.

Returned Value

result — An integer value indicating $n_coefficients$, where $n_coefficients$ is the number of estimated coefficients in the model.

Input Keywords

Double — If present and nonzero, double precision is used.

Icen — The column in x containing the censoring code for each observation.

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 $x(i, Irt)$.

Ult — The column number of x containing the upper endpoint of the failure interval for interval- and left-censored observations.

Irt — The column number of x containing the lower endpoint of the failure interval for interval- and right-censored observations.

Ifreq — The column number of x containing the frequency of response for each observation.

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.

Eps — The convergence criterion. Convergence is assumed when the 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: $Eps = 0.001$

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: $Itmax = 30$

No_Intercept — If present and nonzero, there is no intercept in the model. By default, the intercept is automatically included in the model.

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 output 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; $Lp_Max = 0$

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.

Indicies_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 *Indicies_Effects* and *Var_Effects* must be used together.

Init_Est — One-dimensional array containing the initial estimates of the parameters (which requires that the user know the number of coefficients in the model prior to the use of 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, unweighted linear regression is used to obtain initial estimates.

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: $Max_Class = n_observations * n_class$

Estimation Input Keywords

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_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_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_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.

Output Keywords

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 $N_Class_Vals(i)$.

Class_Vals — Named variable into which an 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:

Column	Statistic
0	Coefficeient 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:

model	
0	No nuisance parameter
1	Coefficient of the quadratic term, term in time, θ
2-9	Scale parameter, σ
10	Scale parameter, θ

Criterion — Named variable into which the optimized criterion is stored. The criterion to be maximized is a constant plus the log-likelihood.

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.

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.

Case_Analysis — Named variable into which a two-dimensional array of size $n_observations$ by 5 containing the case analysis below is stored:

Column	Statistic
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.

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.

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.

Obs_Status (i)	Status of Observation
0	Observation <i>i</i> is in the likelihood
1	Observation <i>i</i> cannot be in the likelihood because it contains at least one missing value in <i>x</i> .
2	Observation <i>i</i> is not in the likelihood. Its estimated parameter is infinite.

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.

Column	Statistic
0	Method of iteration Q-N Step = 0 N-R Step = 1
1	Iteration number
2	Step size

Column	Statistic
3	Maximum scaled coefficient update
4	Log-likelihood

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*.

Estimation Output Keywords

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_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*.

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 function 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.

- 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

Function SURVIVAL_GLM 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 SURVIVAL_GLM. Thus, while it is not advisable to do so, standard multiple linear regression can be performed by routine 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 SURVIVAL_GLM via function 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 I_{fix} 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 SURVIVAL_GLM are:

model	Name	$S(t)$
0	Exponential	$\exp[-t \exp(w_i + \eta)]$
1	Linear hazard	$\exp\left[-\left(t + \frac{\theta t^2}{2}\right) \exp(w_i + \eta)\right]$

model	Name	$S(t)$
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	$\{1 + \exp\left(\frac{\ln(t) - \eta - w_i}{\sigma}\right)\}^{-1}$
5	Logistic	$\{1 + \exp\left(\frac{t - \eta - w_i}{\sigma}\right)\}^{-1}$
6	Log least extreme value	$\exp\{-\exp\left(\frac{\ln(t) - \eta - w_i}{\sigma}\right)\}$
7	Least extreme value	$\exp\{-\exp\left(\frac{t - \eta - w_i}{\sigma}\right)\}$
8	Log extreme value	$1 - \exp\{-\exp\left(\frac{\ln(t) - \eta - w_i}{\sigma}\right)\}$
9	Extreme value	$1 - \exp\{-\exp\left(\frac{t - \eta - w_i}{\sigma}\right)\}$

model	Name	$S(t)$
10	Weibull	$\exp\left\{-\left[\frac{t}{\exp(w_i + \eta)}\right]^\theta\right\}$

Note that the log-least-extreme-value model is a reparameterization 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, etc., then a multiplicative factor of the exponential parameter $\lambda = \exp(x\beta)$ may be known apriori. 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 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 by the user (see keyword *Init_Est*), the initial estimates 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

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

$$\hat{H}_{\kappa_j \kappa_l} = \sum_i l'_{i\alpha_j \alpha_l}$$

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 a_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), then the methods of Clarkson and Jenrich (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 SURVIVAL_GLM, such estimates can be “computed.”

In all models fit by 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, $I'_{\eta,i}$ 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

$$(I'_i)^T H^{-1} I'_i$$

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 $I'_{\eta,i}$.
- 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).

Function *SURVIVAL_GLM* computes estimates of survival probabilities and hazard rates for the parametric survival/reliability models when using the *Est_** keywords.

Let $\eta = x^T\beta$ be the linear parameterization, where x is the design vector corresponding to a row of x (*SURVIVAL_GLM* generates the design vector using

function REGRESSORS), 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 (input in column *Fix* of 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 function SURVIVAL_GLM when using the keywords *Est_** as follows:

1. The input arguments are checked for consistency and validity.
2. For each row of x , the explanatory variables are generated from the classification and variables and the covariates using function 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 function REGRESSORS (Chapter 2, *Regression*) using keyword *Dummy_Method*.

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.

```
PRO print_results, cs
```

```

PRINT, "                                Coefficient Statistics"
PRINT, "      Coefficient      s.e              z
p"
PM, cs, Format = "(4F14.4)"

END

x = TRANSPOSE([ $
               [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] , $
               [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] , $
               [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] , $
               [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]])

n_class = 2
n_continuous = 3
model = 0
icen = 6
irt = 5
lp_max = 40
n_coef = 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 is the same as Example 1, but more optional arguments are demonstrated.

```

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

x =  TRANSPOSE([ $
      [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] , $
      [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] , $
      [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] , $

```


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

$$\chi^2 = g^T \hat{\Sigma}^{-1} g$$

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).

```

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

x = TRANSPOSE([ $
               [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] , $
               [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] , $
               [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] , $

```

```

[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]])

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 = SURVIVAL_GLM(n_class, n_continuous, model, x, $
                    Icen = icen, Irt = irt, Itmax = it-
                    max, $
                    $
                    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 the function 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.

```

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

x =  TRANSPOSE([ $
              [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] , $
[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] , $
[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] , $
[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.]]

```

```

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 = 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

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.

Probability Distribution Functions and Inverses

Contents of Chapter

Normal (Gaussian) distribution function	NORMALCDF Function
Bivariate normal distribution	BINORMALCDF Function
Chi-squared distribution function	CHISQCDF Function
F distribution function	FCDF Function
Student's t distribution function	TCDF Function
Gamma distribution function	GAMMACDF Function
Beta distribution function	BETACDF Function
Binomial distribution function	BINOMIALCDF Function
Binomial probability function	BINOMIALPDF Function
Hypergeometric distribution function	HYPERGEOCDF Function
Poisson distribution function	POISSONCDF Function

NORMALCDF Function

Evaluates the standard normal (Gaussian) distribution function. Using a keyword, the inverse of the standard normal (Gaussian) distribution can be evaluated.

Usage

result = NORMALCDF(*x*)

Input Parameters

x — Expression for which the normal distribution function is to be evaluated.

Returned Value

result — The probability that a normal random variable takes a value less than or equal to *x*.

Input 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

Function NORMALCDF 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 NORMALCDF is the distribution function) has mean of zero and variance of 1. The probability that a normal ran-

dom variable with mean μ and variance σ^2 is less than y is given by NORMALCDF evaluated at $(y - \mu) / \sigma$.

The function $\Phi(x)$ is evaluated by use of the complementary error function, ERFC. The relationship follows below.

$$\Phi(x) = \text{ERFC}((-x/\sqrt{2.0})/2.)$$

If the keyword *Inverse* is specified, the NORMALCDF function evaluates the inverse of the distribution function, Φ , of a standard normal (Gaussian) random variable; that is,

$$\text{NORMALCDF}(x, \text{Inverse}) = \Phi^{-1}(x) \quad \text{where}$$

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-t^2/2} dt$$

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 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 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 = 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 = NORMALCDF(x2) - 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
```

BINORMALCDF Function

Evaluates the bivariate normal distribution function.

Usage

result = BINORMALCDF(*x*, *y*, *rho*)

Input Parameters

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.

rho — Correlation coefficient.

Returned Value

result — 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*.

Input Keywords

Double — If present and nonzero, double precision is used.

Discussion

Function BINORMALCDF evaluates the distribution function *F* of a bivariate normal distribution with means of zero, variances of 1, and correlation of *rho*; that is, $\rho = rho$ 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 BINORMALCDF would be as follows:

$$X = \frac{(u_0 - \mu_U)}{\sigma_U}, Y = \frac{(v_0 - \mu_V)}{\sigma_V}, \text{ and } \rho = \frac{\sigma_{UV}}{(\sigma_U \sigma_V)}$$

The 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 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 = BINORMALCDF(x, y, rho)
; Call 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
```

CHISQCDF Function

Evaluates the chi-squared distribution or noncentral chi-squared distribution. Using a keyword the inverse of these distributions can be computed.

Usage

result = CHISQCDF(*chisq*, *df* [, *delta*])

Input Parameters

chisq — Expression for which the chi-squared distribution function is to be evaluated. If keyword *Inverse* is specified, the probability for which the inverse of the noncentral, chi-squared distribution function is to be evaluated, the parameter *chisq* must be in the open interval (0.0, 1.0).

df — Number of degrees of freedom of the chi-squared distribution. Argument *df* must be greater than or equal to 0.5.

delta — (Optional) The noncentrality parameter. *delta* must be nonnegative, and *delta* + *df* must be less than or equal to 200,000.

Returned Value

result — The probability that a chi-squared random variable takes a value less than or equal to *chisq*.

Input 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

Function CHISQCDF 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 $v > 65$, CHISQCDF uses the Wilson-Hilferty approximation (Abramowitz and Stegun 1964, Equation 26.4.17) to the normal distribution, and NORMALCDF function is used to evaluate the normal distribution function.

For $v \leq 65$, CHISQCDF uses series expansions to evaluate the distribution function. If $x < \max(v/2, 26)$, 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 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 $v < 40$, 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 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

Function CHISQCDF evaluates the distribution function of a noncentral chi-squared random variable with df degrees of freedom and noncentrality parameter $delta$, that is, with $\nu = df$, $\lambda = delta$, and $x = chisq$,

$$non_central_chi_sq(x) = \sum_{i=0}^{\infty} \frac{e^{-\lambda/2} (\lambda/2)^i}{i!} \int_0^x \frac{t^{(\nu+2i)/2-1} e^{-t/2}}{2^{\nu+2i}/2\Gamma(\frac{\nu+2i}{2})} dt$$

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 noncentral 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 noncentral chi-squared distribution with n degrees of freedom and noncentrality parameter equal to

$$\sum_{i=1}^n \mu_i^2$$

With a noncentrality parameter of zero, the noncentral chi-squared distribution is the same as the chi-squared distribution.

Function CHISQCDF 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, CHISQCDF evaluates the inverse distribution function of a noncentral chi-squared random variable with df degrees of freedom and noncentrality parameter $delta$; that is, with $P = chisq$, $\nu = df$, and $\lambda = delta$, it determines $c_0 (= CHISQCDF(chisq, df, delta))$, such that

$$P = \sum_{i=0}^{\infty} \frac{e^{-\lambda/2} (\lambda/2)^i}{i!} \int_0^{c_0} \frac{x^{(v+2i)/2-1} e^{-x/2}}{2^{(v+2i)/2} \Gamma(\frac{v+2i}{2})} dx$$

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 = 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 - 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
```

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 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 CHISQCDF is returned.

Alert Errors

STAT_NORMAL_UNDERFLOW — Using the normal distribution for large degrees of freedom, underflow would have occurred.

FCDF Function

Evaluates the F distribution function. Using a keyword, the inverse of the F distribution function can be evaluated.

Usage

result = FCDF(*f*, *dfnum*, *dfden*)

Input Parameters

f — Expression for which the F distribution function is to be evaluated.

dfnum — Numerator degrees of freedom. Parameter *dfnum* must be positive.

dfden — Denominator degrees of freedom. Parameter *dfden* must be positive.

Returned Value

result — The probability that an F random variable takes a value less than or equal to the input point *f*.

Input 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

Function FCDF 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 ν_1 and ν_2 degrees of freedom and $Y = (\nu_1 X) / (\nu_2 + \nu_1 X)$, then Y is a beta variate with parameters $p = \nu_1 / 2$ and $q = \nu_2 / 2$. The FCDF function also uses a relationship between F random variables that can be expressed as follows: $F_F(f, \nu_1, \nu_2) = 1 - F_F(1/f, \nu_2, \nu_1)$, where F_F is the distribution function for an F random variable.

If the keyword *Inverse* is specified, the 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 - 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
```

Fatal Errors

STAT_F_INVERSE_OVERFLOW — Function 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.

TCDF Function

Evaluates the Student's t distribution or noncentral Student's t distribution. Using a keyword the inverse of these distributions can be computed.

Usage

result = TCDF(*chisq*, *df* [, *delta*])

Input Parameters

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).

df — Degrees of freedom. Argument *df* must be greater than or equal to 1.0.

delta — (Optional) The noncentrality parameter.

Returned Value

result — The probability that a Student's t random variable takes a value less than or equal to the input *t*.

Input 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

Function TCDF 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 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

Function TCDF evaluates the distribution function F of a noncentral t random variable with df degrees of freedom and noncentrality parameter $delta$; 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 noncentral 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 noncentral t distribution with degrees of freedom and noncentrality parameter δ .

The distribution function of the noncentral 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 `TCDF` evaluates the inverse distribution function of a noncentral t random variable with df degrees of freedom and noncentrality parameter δ ; that is, with $P = t$, $v = df$, and $\delta = \delta$, it determines $t_0 (= \text{TCDF}(t, df, \delta))$, 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 * TCDF(-2.447, 6)
PM, 'Pr(|t(6)| > 2.447) = ', p, $
  Format = '(a21, f7.4)'
Pr(|t(6)| > 2.447) = 0.0500
```

Informational Errors

`STAT_OVERFLOW` — Function `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.

GAMMACDF Function

Evaluates the gamma distribution function.

Usage

result = GAMMACDF(*x*, *a*)

Input Parameters

x — Argument for which the gamma distribution function is to be evaluated.

a — Shape parameter of the gamma distribution. This parameter must be positive.

Returned Value

result — The probability that a gamma random variable takes a value less than or equal to *x*.

Input Keywords

Double — If present and nonzero, double precision is used.

Discussion

Function GAMMACDF 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 distri-

bution 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 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, 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 = 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 = 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
```

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.

BETACDF Function

Evaluates the beta probability distribution function.

Usage

result = BETACDF(*x*, *pin*, *qin*)

Input Parameters

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).

pin — First beta distribution parameter. Parameter *pin* must be positive.

qin — Second beta distribution parameter. Parameter *qin* must be positive.

Returned Value

result — The probability that a beta random variable takes on a value less than or equal to *x*.

Input 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

Function BETACDF 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 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 = BETACDF(.6, 12, 12)
; Call BETACDF to compute the first probability and output the 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 - BETACDF(.5, 12, 12)
; Call 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
```

BINOMIALCDF Function

Evaluates the binomial distribution function.

Usage

result = BINOMIALCDF(*k*, *n*, *p*)

Input Parameters

k — Argument for which the binomial distribution function is to be evaluated.

n — Number of Bernoulli trials.

p — Probability of success on each trial.

Returned Value

result — The probability that *k* or fewer successes occur in *n* independent Bernoulli trials, each of which has a probability *p* of success.

Input Keywords

Double — If present and nonzero, double precision is used.

Discussion

Function BINOMIALCDF 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 $p = 0$, BINOMIALCDF is set to 1; for the case $p = 1$, 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 = BINOMIALCDF(3, 5, .95)
PM, 'Pr(x < 3) = ', p, $
    Format = '(a12, f7.4) '
Pr(x < 3) = 0.0226
```

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 .

BINOMIALPDF Function

Evaluates the binomial probability function.

Usage

result = BINOMIALPDF (k , n , p)

Input Parameters

k — Argument for which the binomial probability function is to be evaluated.

n — Number of Bernoulli trials.

p — Probability of success on each trial.

Returned Value

result — The probability that a binomial random variable takes a value equal to k .

Discussion

The function BINOMIALPDF 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

$$\Pr(X = j) = \frac{(n+1-j)p}{j(1-p)} \Pr(X = j-1)$$

To avoid the possibility of underflow, the probabilities are computed forward from 0, if k is not greater than n times p , 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 $p = 0$, BINOMIALPDF returns 0 if k is greater than 0 and to 1 otherwise; and for the case $p = 1$, 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, BINOMIALPDF(3, 5, .95)
0.0214344
```

HYPERGEOCDF Function

Evaluates the hypergeometric distribution function.

Usage

result = HYPERGEOCDF(*k*, *n*, *m*, *l*)

Input Parameters

k — Parameter for which the hypergeometric distribution function is to be evaluated.

n — Sample size. Argument *n* must be greater than or equal to *k*.

m — Number of defectives in the lot.

l — Lot size. Parameter *l* must be greater than or equal to *n* and *m*.

Returned Value

result — 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.

Input Keywords

Double — If present and nonzero, double precision is used.

Discussion

Function HYPERGEOCDF 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 $\min(n, m)$, 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, 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 $(m + 1)(n + l) / (l + 2)$.

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 = HYPERGEOCDF(7, 100, 70, 1000)
PM, 'Pr(x <= 7) = ', p, $
    Format = '(a13,f7.4)'
Pr(x <= 7) = 0.5995
```

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 .

POISSONCDF Function

Evaluates the Poisson distribution function.

Usage

result = POISSONCDF(*k*, *theta*)

Input Parameters

k — Parameter for which the Poisson distribution function is to be evaluated.

theta — Mean of the Poisson distribution. Parameter *theta* must be positive.

Returned Value

result — The probability that a Poisson random variable takes a value less than or equal to *k*.

Input Keywords

Double — If present and nonzero, double precision is used.

Discussion

Function POISSONCDF 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 POISSONCDF function uses the recursive relationship

$$f(x + 1) = f(x)(\theta/(x + 1)) \quad , \quad \text{for } x = 0, 1, 2, \dots, k - 1$$

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 = POISSONCDF(7, 10)
PM, 'Pr(x <= 7) = ', p, $
    Format = '(a13,f7.4) '
Pr(x <= 7) = 0.2202
```

Informational Errors

STAT_LESS_THAN_ZERO — Input parameter, k , is less than zero.

Random Number Generation

Contents of Chapter

Random Numbers

- Retrieves uniform (0, 1) multiplicative, congruential pseudorandom-number generator [RANDOMOPT Procedure](#)
- Sets or retrieves the current table used in either the shuffled or GFSR random number generator. [RANDOM_TABLE Procedure](#)
- Generates pseudorandom numbers..... [RANDOM Function](#)
- Generates pseudorandom numbers from a nonhomo-geneous Poisson proces [RANDOM_NPP Function](#)
- Generates pseudorandom order statistics from a uniform (0, 1) distribution, or optionally from a standard normal distribution [RANDOM_ORDER Function](#)
- Generates a pseudorandom two-way table..... [RAND_TABLE_2WAY Function](#)
- Generates a pseudorandom orthogonal matrix or a correlation matrix..... [RAND_ORTH_MAT Function](#)

Generates a simple pseudorandom sample from a finite population.....[RANDOM_SAMPLE Function](#)

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

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

Generates pseudorandom numbers from a general continuous distribution.....[RAND_GEN_CONT Function](#)

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

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

Stochastic Processes

Generate pseudorandom ARMA process numbers[RANDOM_ARMA Function](#)

Low-discrepancy sequences

Initializes the structure used for computing a shuffled Faure sequence[FAURE_INIT Function](#)

Generates a shuffled Faure sequence[FAURE_NEXT_PT Function](#)

Introduction

Overview of Random Number Generation

The *Random Numbers* section describes functions for the generation of random numbers that are useful for applications in Monte Carlo or simulation studies. Before using any of the random number generators, the generator must be ini-

tialized by selecting a *seed* or starting value. The user can do this by calling the function RANDOMOPT. If the user does 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 discussions, the phrases “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 completely deterministic and repeatable, they simulate the realizations of independent and identically distributed random variables.

Basic Uniform Generator

The default action of the RANDOM function is the generation of uniform (0,1) numbers. This function is portable in the sense 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 routine RANDOMOPT (page 510). 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).

The 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 the function 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

The user also can select a shuffled version of these generators using 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.

The 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 the Seed

The seed of the generator can be set and retrieved using RANDOMOPT. Prior to invoking any generator in this section, the user can call RANDOMOPT to initialize the seed, which is an integer variable with a value between 1 and 2147483647. If it is not initialized by RANDOMOPT, a random seed is obtained from the system clock. Once it is initialized, the seed need not be set again.

If the user wants to restart a simulation, 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,

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, the user must also reset some values in a table. For the shuffled generators, this is done using the routine `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 the 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 `RAND_GEN_CONT` and `RAND_GEN_DISCR`.

Additional Notes on Usage

The generators for continuous distributions are available in both single and double precision versions. This is merely for the convenience of the user; the double precision versions should not be considered more “accurate,” except possibly for the multivariate distributions.

RANDOMOPT Procedure

Uses keywords to set or retrieve the random number seed or to select the form of the IMSL random number generator.

Usage

RANDOMOPT

Input Parameters

Procedure RANDOMOPT does not have any positional Input Parameters. Keywords are required for specific actions to be taken.

Input Keywords

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.

Gen_Option	Generator
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

Set — 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 PV- WAVE:IMSL Statistics 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.

Output Keywords

Get — Named variable into which the value of the current random-number seed is stored.

Current_option — Named variable into which the value of the current random-number generator option is stored.

Discussion

Procedure RANDOMOPT 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 function 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 PV- WAVE:IMSL Statistics 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 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, RANDOMOPT 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 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 of the IMSL generators for nonuniform distributions that do not use the inverse CDF method, the distance between the sequences generated starting with *Substream_seed* and starting with the returned seed may be less than 100,000. This is because the 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.

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
RANDOMOPT, Set = seed
    ; Set the seed using the keyword Set.
r1 = 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
RANDOMOPT, Get = seed
    ; Get the current value of the seed using the keyword Get.
RANDOMOPT, Set = seed
    ; Set the seed.
r2 = 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
RANDOMOPT, Set = 123457
    ; Reset the seed to the original seed.
r3 = 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, RANDOMOPT is used to determine seeds for 4 separate streams, each 200,000 numbers apart, for a multiplicative congruential generator without shuffling. (Since 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 RANDOMOPT with keyword *Substream_seed*. All of the streams are non-overlapping, since the period of the underlying generator is 2,147,483,646.

```

RANDOMOPT, GEN_OPTION = 1
is1 = 123457;
RANDOMOPT, Get = itmp, Substream_seed = is1
RANDOMOPT, Get = is2, Substream_seed = itmp
RANDOMOPT, Get = itmp, Substream_seed = is2
RANDOMOPT, Get = is3, Substream_seed = itmp
RANDOMOPT, Get = itmp, Substream_seed = is3
RANDOMOPT, Get = is4, Substream_seed = itmp
PRINT, is1, is2, is3, is4
      123457   2016130173       85016329   979156171

```

RANDOM_TABLE Procedure

Sets or retrieves the current table used in either the shuffled or GFSR random number generator.

Usage

```
RANDOM_TABLE, table, /Get
```

```
RANDOM_TABLE, table, /Set
```

Input/Output Parameters

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.

Input Keywords

Set — If present and nonzero, then the specified table is being set.

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.

Double — If present and nonzero, double precision is used. This keyword is active only when the shuffled table is being set or retrieved.

Discussion

The values in *table* are initialized by the IMSL random number generators. The values are all positive except if the user wishes 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 `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 *Introduction* section on page [506](#) of this chapter.

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)
RANDOMOPT, Set = iseed1
r = RANDOM(nr)
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
RANDOMOPT, gen_opt = 2
RANDOMOPT, Set = iseed2
r = RANDOM(nr)
RANDOMOPT, Get = iseed2
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
RANDOMOPT,  gen_opt = 7
RANDOMOPT,  Set = iseed7
r = RANDOM(nr)
RANDOMOPT,  Get = iseed7
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
RANDOMOPT,  gen_opt = 1
RANDOMOPT,  Set = iseed1
r = RANDOM(nr)
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 ', izeed1
output seed      817878095

; Reinitialize seed and table for shuffling and
; resume second stream
RANDOMOPT, gen_opt = 2
RANDOMOPT, Set = izeed2
RANDOM_TABLE, table, /Set
r = RANDOM(nr)
RANDOMOPT, Get = izeed2
PM, r, Title = 'Second stream output'
Second stream output
0.255690
0.478770
0.225802
0.345467
0.581051
PRINT, 'output seed ', izeed2
output seed      2108806573

; Reinitialize seed and table for GFSR and
; resume third stream.
RANDOMOPT, gen_opt = 7
RANDOMOPT, Set = izeed7
RANDOM_TABLE, itable, /Set, /Gfsr
r = RANDOM(nr)
RANDOMOPT, Get = izeed7
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
```

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.

Usage

result = RANDOM(*n*)

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.

Input Parameters

n — Number of random numbers to generate.

Returned Value

result — 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.

Input Keywords

Double — If present and nonzero, double precision is used.

Parameters — Specifies parameters for the distribution used by 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.

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, the random numbers are generated from a binomial distribution. Requires the *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.

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.

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.

Normal — If present and nonzero, the random numbers are generated from a standard normal distribution using an inverse CDF method.

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.

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 β 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).

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.

Covariances — Two-dimensional, square matrix containing the variance-covariance matrix. The two-dimensional array returned by RANDOM is of the following size:

$$n \text{ by } N_ELEMENTS(Covariances(*, 0))$$

Keywords *Mvar_Normal* and *Covariances* must be specified to return numbers from a multivariate normal distribution.

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.

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.

NOTE The keywords *A*, *Pin*, *Qin*, and *Theta* are still supported, but are now deprecated. Please use the *Parameters* keyword instead.

Discussion

Function RANDOM 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 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 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 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 RANDOMOPT on page 510 for further discussion of generator options.

The RANDOMOPT procedure called with the *Set* keyword is used to initialize the seed of the random-number generator.

The user 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 $1.0 - r(i) = 1.0$.

Deviate from the distribution with uniform density over the interval (a, b) can be obtained by scaling the output. See Example 3 on page 535 for more details.

Normal Distribution

Calling 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 NORMALCDF function with keyword *Inverse*.

If the *Parameters* keyword is specified in addition to *Normal*, 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.

Deviate from the normal distribution with mean specific mean and standard deviation can be obtained by scaling the output from RANDOM. See Example 3 on page 535 for more details.

Exponential Distribution

Calling RANDOM with keyword *Exponential* generates pseudorandom numbers from a standard exponential distribution. The probability density function is $f(x) = e^{-x}$, for $x > 0$. Function RANDOM 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 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, 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 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, RANDOM generates pseudorandom deviates from an Erlang distribution with no modifications required.

Beta Distribution

Calling 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 greater than or equal to 4. Note that for p and q both greater than 1, calling

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 the 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 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 post-multiplied 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 RANDOM with keywords *Mvar_Normal* and *Covariances*, then adding the vectors of means to each row of the result.

Binomial Distribution

Calling 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 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 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, RANDOM generates two uniform (-1, 1) deviates, x_1 and x_2 . These values can be thought of as sine and cosine values. If

$$x_1^2 + x_2^2$$

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 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 RANDOM with keywords *Mix_Exponential*, and *Parameters*= [θ_1, θ_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 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 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 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 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 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 or 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), the user 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 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 RANDOM with keywords *Lognormal*, and *Parameter*=[μ , σ] 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 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, RANDOM uses the inverse CDF technique; otherwise, for each negative binomial deviate, RANDOM generates a *gamma* ($r, p/(1 - p)$) deviate Y and then generates a Poisson deviate with parameter Y .

Discrete Uniform Distribution

Calling 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 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 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 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 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.

Function RANDOM 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 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 the restrictions $0 < \alpha \leq 2$ and $-1 \leq \beta \leq 1$, the characteristic function of the distribution is

$$\varphi(t) = \exp[-|t|^\alpha \exp(-\pi i \beta (1 - |1 - \alpha| \text{sign}(t)/2))] \text{ for } \alpha \neq 1$$

and

$$\varphi(t) = \exp[-|t|^\alpha (1 + 2i\beta \ln|t|) \text{sign}(t)/\pi] \text{ for } \alpha = 1$$

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 RANDOM with keywords *Multinomial*, *Probabilites*, and *Parameters=ntrials* generates pseudorandom numbers from a K -variate multinomial distribution with parameters n and p . $k=N_ELEMENTS(Probabilites)$ and *ntrials* must be positive. Each element of *Probabilites* must be positive and the elements must sum to 1. The probability function (with $n = n$, $k = k$, and $p_i = Probabilites(i)$) is

$$f(x_1, x_2, \dots, x_k) = \frac{n!}{x_1! x_2! \dots x_k!} p_1^{x_1} p_2^{x_2} \dots p_k^{x_k}$$

for $x_i \geq 0$ and

$$\sum_{i=0}^{k-1} x_i = n$$

The deviate in each row of r is produced by generation of the binomial deviate x_0 with parameters n and p_i and then by successive generations of the condi-

tional 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 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 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 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.

Example 1

In this example, 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.

```
RANDOMOPT, Set = 123457
    ; Set the random seed.

r = RANDOM(5)
    ; Call RANDOM to compute the random numbers.

PM, r
    ; Output the results.

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.

```
RANDOMOPT, Set = 123457
r = RANDOM(5, /Poisson, Parameters = 0.5)
    ; Call RANDOM with keywords Poisson and Theta.
PM, r
2
0
1
0
1
```

Example 3: Beta Distribution

In this example, random numbers are computed from a Beta distribution.

```
RANDOMOPT, set = 123457
r = RANDOM(5, /Beta, Parameter = [3,2])
    ; Call RANDOM with keywords Beta, Pin, and Qin.
PM, r
0.281392
0.948276
0.398391
0.310306
0.829578
```

Example 4: Scaling the Results of 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.

```
RANDOMOPT, Set = 123457
    ; Set the random number seed.
a = 10
    ; Define the lowerbound.
b = 20
    ; Define the upperbound.
r = a + (b - a) * RANDOM(5)
    ; Call RANDOM to compute the deviates on (0,1) and scale the
```

```

; results to (a,b).
PM, r
; Output the results.
19.6622
12.6071
17.6626
15.6934
18.4483

stdev = 2
; Define a standard deviation.

mean = 10
; Define a mean.

r = RANDOM(6, /Normal) * stdev + mean
; Call RANDOM to compute the deviates normal deviates and scale
; the results using the specified mean and standard deviation.

PM, r
; Output the results.
6.59363
14.4635
10.5137
12.5223
9.39352
5.71021

```

Example 5: Multivariate Normal Distribution

In this example, RANDOM generates five pseudorandom normal vectors of length 2 with variance covariance matrix equal to the following:

$$\begin{bmatrix} 0.500 & 0.375 \\ 0.375 & 0.500 \end{bmatrix}$$

```

RANDOMOPT, Set = 123457
; Set the random number seed.

RM, cov, 2, 2
; Read the covariance matrix.

row 0: .5 .375
row 1: .375 .5

PM, RANDOM(5, /Mvar_Normal, Covariances = cov)

```

1.45068	1.24634
0.765975	-0.0429410
0.0583781	-0.669214
0.903489	0.462826
-0.866886	-0.933426

RANDOM_NPP Function

Generates pseudorandom numbers from a nonhomogeneous Poisson process.

Usage

result = RANDOM_NPP(*tbegin*, *tend*, *ftheta*, *theta_min*, *theta_max*, *neub*)

Input Parameters

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*.

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.

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.

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 RANDOM_NPP is less the greater *theta_max* exceeds the true maximum.

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 $X = theta_max * (tend - tbegin)$.

Returned Value

A one dimensional array containing the times to events. If then length of the result is less than *neub*, the time *tend* is reached before *neub* events are realized

Input Keywords

Double — If present and nonzero, double precision is used.

Discussion

Routine 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 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

$$\Lambda(t) = \int_{t_0}^t \lambda^*(t) dt$$

is called the *integrated rate function*. In 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 RANDOM_NPP again. Cox and Lewis (1966) discuss modeling applications of nonhomogeneous Poisson processes.

Example

In this example, 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
% Compiled module: FTHETA_NPP.

randomopt, set=123457
neub = 5
```

```
tmax = .652561
tmin = .6342
tbegin=0
tend=20

r = RANDOM_NPP(tbegin, tend, 'ftheta_npp', tmin, tmax, Neub)
PM, r
0.0526598
0.407979
0.258399
0.0197666
0.167641
```

RANDOM_ORDER Function

Generates pseudorandom order statistics from a uniform (0, 1) distribution, or optionally from a standard normal distribution.

Usage

result = RANDOM_ORDER(*ifirst*, *ilast*, *n*)

Input Parameters

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.

Input Keywords

Double — If present and nonzero, double precision is used.

Uniform — If present and nonzero, generate pseudorandom order statistics from a uniform (0, 1) distribution. (Default)

Normal — If present and nonzero, generate pseudorandom order statistics from a standard normal distribution.

Returned 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.

Discussion

Routine `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 `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 `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 `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, `RANDOM_ORDER` is used to generate the fifteenth through the nineteenth order statistics from a sample of size twenty.

```
r = random_order(15, 19, 20)
pm, r
      0.706909
      0.808627
      0.874552
```

0.922146

0.957402

***RAND_TABLE_2WAY* Function**

Generates a pseudorandom two-way table.

Usage

result = RAND_TABLE_2WAY (*row_totals*, *col_totals*)

Input Parameters

row_totals — One dimensional array containing the row totals.

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.

Returned Value

A N_ELEMENTS(*row_totals*) by N_ELEMENTS(*col_totals*) random matrix with the given row and column totals.

Discussion

Routine RAND_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, `RAND_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 = RAND_TABLE_2WAY([3, 5], [2, 4, 2])
PM, r
      2      1      0
      0      3      2
```

RAND_ORTH_MAT Function

Generates a pseudorandom orthogonal matrix or a correlation matrix.

Usage

```
result = RAND_ORTH_MAT(n)
```

Input Parameters

n — The order of the matrix to be generated.

Returned Value

A two-dimensional array containing the *n* by *n* random correlation matrix.

Input Keywords

Double — If present and nonzero, double precision is used.

Eigenvalues — A 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.

A_Matrix — A two-dimensional array containing *n* by *n* random orthogonal matrix. A random correlation matrix is generated using the orthogonal matrix input in *A_Matrix*. The keyword *Eigenvalues* must also be supplied if *A_Matrix* is used.

Discussion

Routine RAND_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 the keyword *Eigenvalues* is used, a correlation matrix is formed by applying a sequence of planar rotations to the 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. Bendel and Mickey (1978) and Johnson and Welch (1980) discuss the distribution.

For larger matrices, rounding can become severe; and the double precision results may differ significantly from single precision results.

Example

In this example, RAND_ORTH_MAT is used to generate a 4 by 4 pseudorandom correlation matrix with eigenvalues in the ratio 1:2:3:4.

```
RANDOMOPT, set = 123457
a = RAND_ORTH_MAT(4)
ev = .4d0*[1.0d0, 2.0d0, 3.0d0, 4.0d0]
cor = RAND_ORTH_MAT(n, 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
```

***RANDOM_SAMPLE* Function**

Generates a simple pseudorandom sample from a finite population.

Usage

result = RANDOM_SAMPLE(*nsamp*, *population*)

Input Parameters

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.

Returned Value

nsamp by *nvar* array containing the sample, where *nvar* is the number of columns in the argument *population*.

Input Keywords

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 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 .

Additional_Call — If present and nonzero, then this is an additional invocation of 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 RANDOM_SAMPLE. See Example 2.

Input/Output Keywords

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 RANDOM_SAMPLE with keyword *First_Call* is used for *Sample*.

Discussion

Routine 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.

Example 1

In this example, RANDOM_SAMPLE is used to generate a sample of size 5 from a population stored in the matrix *population*.

```
RANDOMOPT, Set = 123457
pop = STATDATA(2)
samp = RANDOM_SAMPLE(5, pop)
PM, samp
      1764.00      36.4000
      1828.00      62.5000
      1923.00      5.80000
      1773.00      34.8000
      1769.00      106.100
```

Example 2

Routine 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 RANDOM_SAMPLE one observation at a time. This is the way 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.

```
RANDOMOPT, Set = 123457
pop = STATDATA(2)
samp = RANDOM_SAMPLE(5, pop(0, *), /First_Call, Index = ii,
    Npop=np)
FOR i=1,175 DO samp = 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      5.80000
    1773.00      34.8000
    1769.00      106.100
```

RAND_FROM_DATA Function

Generates pseudorandom numbers from a multivariate distribution determined from a given sample.

Usage

result = RAND_FROM_DATA(*n_random*, *x*, *nn*)

Input Parameters

n_random — Number of random multivariate vectors to generate.

x — Two dimensional array of size *nsamp* by *ndim* containing the given sample.

nn — Number of nearest neighbors of the randomly selected point in *x* that are used to form the output point in the result.

Returned Value

n by $ndim$ matrix containing the random multivariate vectors in its rows.

Input 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, `RAND_FROM_DATA` generates a pseudorandom sample with approximately the same moments as the given sample. The sample obtained is essentially the same as if sampling from a Gaussian kernel estimate of the sample density. (See Thompson 1989.) Routine `RAND_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 ($= nm$) 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

$$\sum_{l=1}^m u_l (x_{jl} - \bar{x}_j) + \bar{x}_j$$

The process is then repeated until n such simulated variates are generated and stored in the rows of the result.

Example

In this example, RAND_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 STATDATA, see Chapter 13: *Utilities* of this manual).

```
RANDOMOPT, Set = 123457
r = STATDATA(9)
x = FLTARR(113, 4)
x(*, 0) = r(*, 6)
x(*, 1) = r(*, 9)
x(*, 2) = r(*, 20)
x(*, 3) = r(*, 23)
r = 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
```

CONT_TABLE Procedure

Sets up table to generate pseudorandom numbers from a general continuous distribution.

Usage

CONT_TABLE, *f*, *iopt*, *ndata*, *table*

Input Parameters

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 CONT_TABLE.

<i>iopt</i>	Action
0	CONT_TABLE fills the last four columns of table. The user inputs the points at which the CDF is to be evaluated in the first column of table. These must be in ascending order.
1	CONT_TABLE fills the last three columns of table. The user supplied function <i>f</i> 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.

Input/Output Parameters

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 $table(ndata-1, 1) = 1.0$.

Input Keywords

Double — If present and nonzero, double precision is used.

Discussion

Routine `CONT_TABLE` sets up a table that routine `RAND_GEN_CONT` (page 551) 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 `CONT_TABLE` see the example for routine `RAND_GEN_CONT` (page 551).

RAND_GEN_CONT Function

Generates pseudorandom numbers from a general continuous distribution.

Usage

```
result = RAND_GEN_CONT(n, table)
```

Input Parameters

n — Number of random numbers to generate.

table — A two-dimensional array setup using `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.

Returned Value

An array of length *n* containing the random deviates.

Input Keywords

Double — If present and nonzero, double precision is used.

Discussion

Routine `RAND_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 routine `CONT_TABLE` (page 549). 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, `RAND_GEN_CONT` (page 551) is used to set up a table for generation of beta pseudorandom deviates. The CDF for this distribution is computed by the routine `BETACDF` (Chapter 11). 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.

```
FUNCTION cdf, x
    return, BETACDF(x, 3., 2.)
END

iopt = 0
ndata = 100;
table = FLTARR(100, 5)
x = 0.0;
table(*,0) = FINDGEN(100)/100.
CONT_TABLE, 'cdf', iopt, ndata, table
RANDOMOPT, Set = 123457

r = RAND_GEN_CONT(5, table)
```

```

% BETACDF: Note: STAT_ZERO_AT_X
    Since "X" = 0.000000e+00 is less than or equal to zero,
the distribution function is zero at "x".
% CONT_TABLE: Warning: STAT_SECOND_COL_TABLE3
CDF in the second column of table did not begin at 0.0
and end at 1.0, but they have been adjusted. Prior
to adjustment, table(0, 1) = 0.000000e+00 and
table(ndata-1, 1)= 9.994079e-01.

PM, r
    0.92079391
    0.46412855
    0.76678398
    0.65357975
    0.81706959

```

DISCR_TABLE Function

Sets up table to generate pseudorandom numbers from a general discrete distribution.

Usage

result = DISCR_TABLE(*prf*, *del*, *nndx*, *imin*, *nmass*)

Input Parameters

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 $1.0 - del$.

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 $1.0 - del$.

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 RAND_GEN_DISCR will be.

Input/Out Parameters

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.

Returned 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.

Input Keywords

Double — If present and nonzero, double precision is used.

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.

Discussion

Routine DISCR_TABLE sets up a table that routine RAND_GEN_CONT (page 551) 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 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.

Example 1

In this example, DISCR_TABLE is used to set up a table to generate pseudo-random variates from the discrete distribution:

$$Pr(X = 1) = .05$$

$$Pr(X = 2) = .45$$

$$Pr(X = 3) = .31$$

$$Pr(X = 4) = .04$$

$$Pr(X = 5) = .15$$

In this simple 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.

```
function PRF, x
    return, 0
end
cum_probs = [.05, .5, .81, .85, 1]
cumpr = DISCR_TABLE('PRF', 0.00001, 4, 1, 5, cum_probs =
    cum_probs)
PM, cumpr
    0.0500000
    0.5000000
    0.8100000
    0.8500000
    1.0000000
    3.0000000
    1.0000000
    2.0000000
    5.0000000
```

Example 2

This example, DISCR_TABLE is used to set up a table to generate binomial variates with parameters 20 and 0.5. The routine BINOMIALPDF (Chapter 11, Probability Distribution and Inverses) is used to compute the probabilities.

```
FUNCTION PRF, ix
    RETURN, BINOMIALPDF(ix, 20, .5)
END

cumpr = 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

***RAND_GEN_DISCR* Function**

Generates pseudorandom numbers from a general discrete distribution using an alias method or optionally a table lookup method.

Usage

result = `RAND_GEN_DISCR`(*n*, *imin*, *nmass*, *probs*)

Input Parameters

n — Number of random numbers to generate.

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.

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.

Routine `DISCR_TABLE` (page 553) 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.)

Returned Value

An integer array of length *n* containing the random discrete deviates.

Input 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 $n_{mass} + 1$ containing in the first n_{mass} positions the cumulative probabilities and, possibly, indexes to speed access to the probabilities. Routine DISCR_TABLE (page 553) can be used to initialize *probs* properly.

Discussion

Routine RAND_GEN_DISCR generates pseudorandom numbers from a discrete distribution with probability function given in the vector *probs*; that is

$$Pr(X = i) = p_j$$

for $i = i_{min}, i_{min} + 1, \dots, i_{min} + n_m - 1$ where $j = i - i_{min} + 1$, $p_j = probs(j)$, $i_{min} = imin$, and $n_m = nmass$.

The algorithm is the *alias* method, due to Walker (1974), with modifications suggested by Kronmal and Peterson (1979).

If the keyword *Table* is used, RAND_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. The DISCR_TABLE (page 553) can be used to set up the table *probs*. RAND_GEN_DISCR uses the inverse CDF method to generate the variates.

Example 1

In this example, RAND_GEN_DISCR is used to generate five pseudorandom variates from the discrete distribution:

$$Pr(X = 1) = .05$$

$$Pr(X = 2) = .45$$

$$Pr(X = 3) = .31$$

$$Pr(X = 4) = .04$$

$$Pr(X = 5) = .15$$

```

probs = [.05, .45, .31, .04, .15]
n = 5
imin = 1
nmass = 5
RANDOMOPT, Set_seed = 123457
r = RAND_GEN_DISCR(n, imin, nmass, probs)
PM, r

```

3
2
2
3
5

Example 2

In this example, DISCR_TABLE (page 553) is used to set up a table and then RAND_GEN_DISCR is used to generate five pseudorandom variates from the binomial distribution with parameters 20 and 0.5.

```

FUNCTION PRF, ix
    RETURN, BINOMIALPDF(ix, 20, .5)
END
imin = 0
nmass = 21
RANDOMOPT, Set_seed = 123457
cumpr = DISCR_TABLE('prf', 0.00001, 12, imin, nmass)
r = RAND_GEN_DISCR(n, imin, nmass, cumpr, /table)
PM, r

```

14
9
12
10
12

RANDOM_ARMA Function

Generates a time series from a specific ARMA model.

Usage

result = RANDOM_ARMA(*n*, *nparams*)

result = RANDOM_ARMA(*n*, *nparams*, *ar*)

result = RANDOM_ARMA(*n*, *nparams*, *ma*)

result = RANDOM_ARMA(*n*, *nparams*, *ar*, *ma*)

Input Parameters

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.

Returned Value

result — One-dimensional array of length *n* containing the generated time series.

Input Keywords

Double — If present and nonzero, double precision is used.

Const — Overall constant. See the *Discussion* section.

Default: *Const* = 0

Var_Noise — If present (and *Input_Noise* is *not* used), the noise a_t will be generated from a normal distribution with mean 0 and variance *Var_Noise*.

Keywords *Var_Noise* and *Input_Noise* can not be used together.

Default: $Var_Noise = 1.0$

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* can not be used together.

Ar_Lags — One-dimensional array of length p containing the order of the non-zero autoregressive parameters.

Default: $Ar_Lags = [1, 2, \dots, p]$

Ma_Lags — One-dimensional array of length q containing the order of the non-zero moving average parameters.

Default: $Ma_Lags = [1, 2, \dots, q]$

W_Init — One-dimensional array of length $\max(Ar_Lags(i))$ containing the initial values of the time series.

Default: $W_Init(*) = Const/(1 - ar(0) - ar(1) - \dots - ar(p - 1))$

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.

Output Keywords

Output_Noise — Named variable into which a one-dimensional array of length $n + \max(Ma_Lags(i))$ containing the random noises is stored.

Discussion

Function `RANDOM_ARMA` simulates an $ARMA(p, q)$ 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

$$\theta_0 = \begin{cases} \mu & p = 0 \\ \mu(1 - \sum_{i=1}^p \phi_i) & p > 0 \end{cases}$$

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:

$$\begin{aligned} X(i) = & Const + ar(0) * X(i - Ar_Lags(0)) + \dots + \\ & ar(p - 1) * X(i - Ar_Lags(p - 1)) + \\ & A(I) - ma(0) * A(i - Ma_Lags(0)) - \dots - \\ & ma(q - 1) * A(i - Ma_Lags(q - 1)) \end{aligned}$$

where the constant is related to the mean of the series,

$$\bar{W}$$

as follows:

$$Const = \bar{W} \cdot (1 - ar(0) - \dots - ar(q - 1))$$

and where

$$X(t) = W(t), \quad t = 0, 1, \dots, n - 1$$

and

$$W(t) = W_Init(t + p), \quad t = -p, -p + 1, \dots, -2, -1$$

and A is either *Input_Noise* (if *Input_Noise* is used) or *Output_Noise* (otherwise).

Example 1

In this example, *RANDOM_ARMA* is used to generate a time series of length five, using an ARMA model with three autoregressive parameters and two moving average parameters. The start values are 0.1000, 0.0500, and 0.0375.

```

RANDOMOPT, set = 123457
n = 5
nparams = [3, 2]
ar = [0.5, 0.25, 0.125]
ma = [-0.5, -0.25]
r = RANDOM_ARMA(n, nparams, ar, ma)
PM, r, Format = "(5F10.3)", $
          Title = "                      ARMA random deviates"
                      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 ARMA model with 4 autoregressive parameters and 2 moving average parameters. The start values are 0.1, 0.05 and 0.0375.

```

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 = RANDOM_ARMA(n, nparams, ar, ma, /Accept_Reject, $
              W_Init = wi, Const = theta0, $
              Var_Noise = avar)
PM, r, Format = "(5F10.3)", $
          Title = "                      ARMA random deviates:"
                      ARMA random deviates:
1.467      1.788      2.459      3.330      3.941

```

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).

FAURE_INIT Function

Initializes the structure used for computing a shuffled Faure sequence.

Usage

result = FAURE_INIT(*ndim*)

Input Parameters

ndim — The dimension of the hyper-rectangle.

Returned Value

A structure that contains information about the sequence.

Input 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

$$E = \left[0, t_1\right) \times \dots \times \left[0, t_d\right), \quad 0 \leq t_j \leq 1, \quad 1 \leq j \leq d,$$

λ is the Lebesgue measure, and

$$(E; n)$$

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

$$D_n^{(d)} \leq c(d) \frac{(\log n)^d}{n} \left[\text{base}^{m/2-1} \right]$$

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

$$x_n^{(j)} = \sum_{k=0}^{\infty} \sum_{d=0}^{\infty} c_{kd}^{(j)} a_d(n) b^{-k-1}, \quad 1 \leq j \leq d$$

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,

$$c_{kd} = \begin{cases} \frac{d!}{c!(d-c)!} & k \leq d \\ 0 & k > d \end{cases}$$

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 `FAURE_INIT` is used to create a structure that holds the state of the sequence. Each call to `FAURE_NEXT_PT` returns the next point in the sequence and updates the state structure.

```
state = FAURE_INIT(3)
p = 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
```

***FAURE_NEXT_PT* Function**

Computes a shuffled Faure sequence.

Usage

result = `FAURE_NEXT_PT`(*npts*, *state*)

Input Parameters

npts — The number of points to generate in the hyper-rectangle.

state — State structure created by a call to `FAURE_INIT`.

Returned Value

An array of size *npts* by *state.dim* containing the *npts* next points in the shuffled Faure sequence.

Input Keywords

Double — If present and nonzero, double precision is used.

Output Keywords

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

$$(E; n)$$

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

$$x_n^{(j)} = \sum_{k=0}^{\infty} \sum_{d=0}^{\infty} c_{kd}^{(j)} a_d(n) b^{-k-1}, \quad 1 \leq j \leq d$$

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,

$$c_{kd} = \begin{cases} \frac{d!}{c!(d-c)!} & k \leq d \\ 0 & k > d \end{cases}$$

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 FAURE_INIT is used to create a structure that holds the state of the sequence. Each call to FAURE_NEXT_PT returns the next point in the sequence and updates the state structure.

```
state = FAURE_INIT(3)
p = 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
```

Utilities

Contents of Chapter

Constants and Data Sets

Machine constants	MACHINE Function
Statistical data sets	STATDATA Function

Mathematical Support

Evaluate the binomial coefficient	BINOMIALCOEF Function
Evaluate the complete beta function	BETA Function
Evaluate the real incomplete beta function	BETAI Function
Evaluate the log of the real beta function	LN BETA Function
Evaluate the real gamma function	GAMMA_ADV Function
Evaluate the incomplete gamma function	GAMMAI Function
Evaluate the logarithm of the absolute value of the gamma function	LNGAMMA Function

Error Handling

Informational Error codes

for routines [CMAST_ERR_TRANS Function](#)

Sets options for error

recovery [CMAST_ERR_STOP Function](#)

Sets options for error

printing [CMAST_ERR_PRINT Function](#)

MACHINE Function

Returns information describing the computer's arithmetic.

Usage

result = MACHINE()

Returned Value

result — The information describing the computer's arithmetic is returned in a structure.

Output Keywords

Float — If present and nonzero, a structure containing the information describing the single-precision, floating-point arithmetic is returned.

Double — If present and nonzero, a structure containing the information describing the single-precision, floating-point arithmetic is returned.

Discussion

Function MACHINE returns information describing the computer's arithmetic. This can be used to make programs machine independent. The information returned by MACHINE is in the form of a structure. A different structure is used for each type: integer, float, and double. Depending on how MACHINE is called, a different structure is returned.

The default action of 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 these structures are described below.

Integer Information: IMACHINE

Assume that integers are represented in M -digit, base A form as

$$\sigma \sum^{m} x_k A^k$$

where σ is the sign and $0 \leq x_k < A$ for $k = 0, \dots, M$. Then, the following table 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>

Assume that floating-point numbers are represented in N -digit, base B form as

$$\sigma B^E \sum^{N-1} x_k B^{-k}$$

where σ is the sign and $0 \leq x_k < B$ for $k = 1, \dots, N$ for and $E_{\min} \leq E \leq E_{\max}$.

Tag	Definition
FLOAT_BASE	B , the base
FLOAT_DIGITS	N_f , the number of base- B digits in <i>float</i>
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_DIGITS	N_d , the number of base- B 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- B digits in <i>double</i>

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

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

$$\sigma B^E \sum_{k=1}^{N_f} x_k B^{-k},$$

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 $\text{imach} = \text{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 $\text{amach} = \text{MACHINE}(\textit{Float})$ 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 the following table:

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

The structure DMACHINE contains machine constants that define the computer's double arithmetic. Note that for *double*, if the assignment `imach = MACHINE()` is made, then it

$$B = \text{imach.FLOAT_BASE}, N_f = \text{imach.DOUBLE_DIGITS},$$

$$E_{\min_f} = \text{imach.DOUBLE_MIN_EXP},$$

and

$$E_{\max_f} = \text{imach.DOUBLE_MAX_EXP}.$$

Missing values in PV-WAVE:IMSL Statistics 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 MACHINE are printed on a machine with IEEE (Institute for Electrical and Electronics Engineering) arithmetic.

```
i = machine()
f = machine(/Float)
d = machine(/Double)
; Call INFO with the keyword Structure set to view the contents of the
; structures.
```

INFO, 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_LONG	LONG	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

STATDATA Function

Retrieves commonly analyzed data sets.

Usage

result = STATDATA(*choice*)

Input Parameters

choice — Data set indicator.

CHOICE	Number of Rows	Number of Columns	Description of Data Set
1	16	7	Longley
2	176	2	Wolfers 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
7	296	2	Box and Jenkins Series J
8	100	4	Robinson Multichannel Time Series
9	113	34	Afifi and Azen Data Set A

Returned Value

result — An array containing the desired data set is returned.

Input Keyword

Double — If present and nonzero, double precision is used.

Discussion

Function STATDATA retrieves a standard data set frequently cited in statistics text books or in this manual. The following table 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)
8	Robinson (1967, p. 204)
9	Afifi and Azen (1979, pp. 16–22)

Example

In this example, STATDATA is used to copy the Draper and Smith (1981, Appendix B) data set into X.

```
x = 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			

BINOMIALCOEF Function

Evaluates the binomial coefficient.

Usage

result = BINOMIALCOEF(*n*, *m*)

Input Parameters

n — First parameter of the binomial coefficient. Parameter *n* must be nonnegative.

m — Second parameter of the binomial coefficient. Parameter *m* must be nonnegative.

Returned Value

result — The binomial coefficient

$$\binom{n}{m}$$

is returned.

Input Keywords

Double — If present and nonzero, double precision is used.

Discussion

The binomial function is defined to be

$$\binom{n}{m} = \frac{n!}{m!(n-m)!}$$

with $n \geq m \geq 0$. Also, n must not be so large that the function overflows.

Example

In this example,

$$\binom{9}{5}$$

is computed and printed.

```
n = 9
m = 5
ans = BINOMIALCOEF(n, m)
PRINT, "binomial coefficient =", ans
binomial coefficient =      126.000
```

BETA Function

Evaluates the complete beta function.

Usage

result = BETA(*x*, *y*)

Input Parameters

x — First beta parameter. *x* must be positive.

y — Second beta parameter. *y* must be positive.

Returned Value

result — The value of the beta function $\beta(x, y)$. If no result can be computed, then NaN is returned.

Input Keywords

Double — If present and nonzero, double precision is used.

Discussion

The beta function, $\beta(x, y)$, is defined to be

$$\beta(x, y) = \frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)} = \int_0^1 t^{x-1}(1-t)^{y-1} dt$$

The beta function requires that $x > 0$ and $y > 0$. It underflows for large arguments.

Example

Evaluate the beta function $\beta(0.5, 0.2)$.

```
x = 0.5
y = 0.2
ans = BETA(x, y)
PRINT, "beta(", x, ", ", y, ") =", ans
beta( 0.500000, 0.200000) = 6.26865
```

Alert Errors

STAT_BETA_UNDERFLOW — The arguments must not be so large that the result underflows.

Fatal Errors

STAT_ZERO_ARG_OVERFLOW — One of the arguments is so close to zero that the result overflows.

BETAI Function

Evaluates the real incomplete beta function $I_x = \beta_x(y, z)/\beta(y, z)$.

Usage

result = BETAI(*x*, *y*, *z*)

Input Parameters

x — Point at which the incomplete beta function is to be evaluated.

y — Point at which the incomplete beta function is to be evaluated.

z — Point at which the incomplete beta function is to be evaluated.

Returned Value

result — The value of the incomplete beta function.

Input Keywords

Double — If present and nonzero, double precision is used.

Discussion

The incomplete beta function is defined to be

$$I_x(y, z) = \frac{\beta_x(y, z)}{\beta(y, z)} = \frac{1}{\beta(y, z)} \int_0^x t^{y-1} (1-t)^{z-1} dt$$

The incomplete beta function requires that $0 \leq x \leq 1$, $y > 0$, and $z > 0$. It underflows for sufficiently small x and large y . This underflow is not reported as an error. Instead, the value zero is returned.

Example

Evaluate the log of the incomplete beta function $I_{0.61} = \beta_{0.61}(2.2, 3.7)/\beta(2.2, 3.7)$.

x = 0.61

y = 2.2

```
z = 3.7
ans = BETAI(x, y, z)
PRINT, "beta incomplete =", ans
beta incomplete = 0.882172
```

LNBETA Function

Evaluates the logarithm of the real beta function $\ln \beta(x, y)$.

Usage

result = LNBETA(*x*, *y*)

Input Parameter

x — Point at which the logarithm of the beta function is to be evaluated.
x must be positive.

y — Point at which the logarithm of the beta function is to be evaluated.
y must be positive.

Returned Value

result — The value of the logarithm of the beta function $\beta(x, y)$.

Input Keywords

Double — If present and nonzero, double precision is used.

Discussion

The beta function, $\beta(x, y)$, is defined to be

$$\beta(x, y) = \frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)} = \int_0^1 t^{x-1}(1-t)^{y-1} dt$$

and 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 arguments.

Example

Evaluate the log of the beta function $\ln \beta(0.5, 0.2)$.

```
x = 0.5
y = 0.2
ans = LNBETA(x, y)
PRINT, "log beta(", x, ", ", y, ") =", ans
log beta( 0.500000, 0.200000) = 1.83556
```

Warning Errors

STAT_X_IS_TOO_CLOSE_TO_NEG_1 — The result is accurate to less than one precision because the expression $-x/(x + y)$ is too close to -1 .

GAMMA_ADV Function

Evaluates the real gamma function.

Usage

result = GAMMA_ADV(*x*)

Input Parameters

x — Point at which the gamma function is to be evaluated.

Returned Value

result — The value of the gamma function $\Gamma(x)$.

Input 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 = GAMMA_ADV(x)
PRINT, "Gamma(", x, ") =", ans
Gamma( 1.50000) = 0.886227
```

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.

GAMMAI Function

Evaluates the incomplete gamma function $\gamma(x, y)$.

Usage

result = GAMMAI(*x*, *y*)

Input Parameters

x — Parameter of the incomplete gamma function is to be evaluated. *x* must be positive.

y — Point at which the incomplete gamma function is to be evaluated. *y* must be nonnegative.

Returned Value

result — The value of the incomplete gamma function $\gamma(x, y)$.

Input Keywords

Double — If present and nonzero, double precision is used.

Discussion

The incomplete gamma function, $\gamma(x, y)$, is defined to be

$$\gamma(x, y) = \int_0^x t^{y-1} e^{-t} dt$$

for $y > 0$. The incomplete gamma function is defined only for $x > 0$. Although $\gamma(x, y)$ is well defined for $y > -\infty$, this algorithm does not calculate $\gamma(x, y)$ for negative y . For large x and sufficiently large y , $\gamma(x, y)$ may overflow. $\gamma(x, y)$ is

bounded by $\Gamma(x)$, and users may find this bound a useful guide in determining legal values for x .

Example

Evaluates the incomplete gamma function at $x = 1$ and $y = 3$.

```
x = 1.0
y = 3.0
ans = GAMMAI(x, y)
PRINT, "incomplete gamma(", x, ",", y, ") =", ans
incomplete gamma(      1.00000,      3.00000) =      0.950213
```

Fatal Errors

STAT_NO_CONV_200_TS_TERMS — The function did not converge in 200 terms of Taylor series.

STAT_NO_CONV_200_CF_TERMS — The function did not converge in 200 terms of the continued fraction.

LNGAMMA Function

Evaluates the logarithm of the absolute value of the gamma function $\log |\Gamma(x)|$.

Usage

result = LNGAMMA(*x*)

Input Parameters

x — Point at which the logarithm of the absolute value of the gamma function is to be evaluated.

Returned Value

result — The value of the logarithm of gamma function $\log |\Gamma(x)|$.

Input 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.

```
x = 3.5
ans = LNGAMMA(x)
PRINT, "log gamma(", x, ") =", ans
log gamma( 3.50000) = 1.20097
```

Warning Errors

STAT_NEAR_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_NEGATIVE_INTEGER — The parameter for the function cannot be a negative integer.

STAT_NEAR_NEG_INT_FATAL — The parameter for the function is too close to a negative integer.

STAT_LARGE_ABS_ARG_OVERFLOW — $|x|$ must not be so large that the result overflows.

CMAST_ERR_TRANS Function

Determines if an *Informational Error* has occurred.

Usage

result = CMAST_ERR_TRANS(*arg*)

Output Parameters

arg — Can be either a scalar string specifying a particular *Informational Error* or an integer specifying the internal code of an *Informational Error*.

Returned Value

result — If *arg* is a scalar string specifying a valid *Informational Error*, then the return value is the integer error-code value of the *Informational Error*. If *arg* is an integer specifying a valid *Informational Error* code, then a string specifying the *Informational Error* is returned.

Discussion

Function CMAST_ERROR_TRANS is designed to check programs for specific *Informational Errors*. PV-WAVE:IMSL Statistics statistical functions attempt to detect user errors and handle them in a way that provides as much information to the user as possible. To do this, five levels of *Informational Error* severity, in addition to the basic PV-WAVE:IMSL Statistics error-handling facility, are recognized. Following a call to a mathematical or statistical function, the system variables !Error and !Cmast_Err contain information concerning the current error state. Variable !Error contains the error *number* of the last error, and !Cmast_Err is set either to zero, which indicates that an *Informational Error* did not occur, or to the error *code* of the last *Informational Error* that did occur.

The user can interact with the PV-WAVE:IMSL Statistics error-handling system with respect to *Informational Errors* in two ways: (1) change the default printing actions and (2) determine the code of an *Informational Error* so as to take corrective action. To change the default printing action, the system variable !Quiet is set to a nonzero value. To allow for corrective action to be taken based on the existence of a particular *Informational Error*, function CMAST_ERR_TRANS retrieves the integer code for an *Informational Error* given a scalar string specifying the name given to the error.

In the program segment below, the Cholesky factorization of a matrix is to be performed. If it is determined that the matrix is not nonnegative definite (and often this is not immediately obvious), the program is to take a different branch.

```
x = CHNNDFAC, a, fac
    ; Call CHNNDFAC with a matrix that may not be nonnegative definite.

IF (CMAST_ERROR_TRANS($
    'MATH_NOT_NONNEG_DEFINITE') EQ $
    !Cmast_Err) $
    ; Check the system variable Cmast_Err to see if it contains the
    ; error code for the error MATH_NOT_NONNEG_DEFINITE.

THEN ;... Handle matrix that is not nonnegative definite.
```

CMAST_ERR_STOP Function

Sets options for error recovery in Math and Stat options.

Usage

`CMAST_ERR_STOP, lev`

Input Parameters

lev — Integer specifying the stopping level.

Discussion

Function `CMAST_ERR_STOP` allows users to define how the Math and Stat options will behave when a Terminal or Fatal error occurs. Setting *lev* to one will force the Math/Stat routine to stop execution when a Terminal or Fatal error occurs (default). Setting *lev* to zero will force the Math/Stat routine to continue execution when a Terminal or Fatal error occurs.

CMAST_ERR_PRINT Function

Sets options for error printing in Math and Stat options.

Usage

`CMAST_ERR_PRINT, lev`

Input Parameters

lev — Integer specifying the printing level.

Discussion

Function `CMAST_ERR_PRINT` allows users to define how the Math and Stat options will behave when an error occurs. Setting *lev* to two will force the Math/Stat routine to print all error messages that occur (default). Setting *lev* to one will force the Math/Stat routine to print only Terminal and Fatal error messages that occur. Setting *lev* to zero will force the Math/Stat routine to not print any error messages.

Example

In this example, the function CSTRENDS is called with a data set that will generate a warning error. After the first call to CSTRENDS, a call is made to CMAST_ERR_PRINT to shut off printing of all but Terminal and Fatal errors.

```
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]
pstat = CSTRENDS(x)
% CSTRENDS: Warning: STAT_AT_LEAST_ONE_TIE
    At least one tie is detected between the samples.
PM, pstat
    0.999996
    7.24792e-05
    1.00000
    3.81470e-06
    1.00000
    0.000244141
    1.00000
    0.000244141
    ; Call CMAST_ERR_PRINT to shut off printing of NOTE, ALERT and
    ; WARNING errors.
CMAST_ERR_PRINT, 1
Call CSTRENDS again. Note that the error message ids not print-
ed.
```

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.
- 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*, Clarendon 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.
- 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.
- 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.
- 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.
- 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.
- 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 Zufalls-zahlen, *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.
- 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 Hillstrom (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.
- 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.
- Spurrier, 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.
- 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.

Summary of Routines

ALLBEST Procedure	page 100
Selects the best multiple linear regression models.	
ANOVA1 Function	page 212
Analyzes a one-way classification model.	
ANOVABALANCED Function	page 242
Balanced fixed, random, or mixed model	
ANOVAFACT Function	page 221
Analyzes a balanced factorial design with fixed effects.	
ANOVANESTED Function	page 231
Nested random mode	
ARMA Function	page 366
Computes method-of-moments or least-squares estimates of parameters for a nonseasonal ARMA model.	
AUTOCORRELATION Function	page 391
Sample autocorrelation function	
BETA Function	page 581
Evaluate the complete beta function.	
BETACDF Function	page 495
Evaluates the beta probability distribution function.	
BETAI Function	page 583
Evaluate the real incomplete beta function.	

BINOMIALCDF Function	page 497
Evaluates the binomial distribution function.	
BINOMIALCOEF Function	page 580
Evaluate the binomial coefficient.	
BINOMIALPDF Function	page 498
Evaluates the binomial probability function.	
BINORMALCDF Function	page 480
Evaluates the bivariate normal distribution function.	
BOXCOXTRANS Function	page 387
Perform a Box-Cox transformation	
CAT_GLM Function	page 280
Generalized linear models.	
CHISQCDF Function	page 482
Evaluates the chi-squared distribution function.	
Using a keyword, the inverse of the chi-squared distribution can be evaluated.	
CHISQTEST Function	page 334
Performs a chi-squared goodness-of-fit test.	
CMAST_ERR_PRINT Procedure	page 591
Sets options for error printing.	
CMAST_ERR_STOP Procedure	page 591
Sets options for error recovery.	
CMAST_ERR_TRANS Function	page 589
Determines if an Informational Error has occurred.	
COCHRANQ Function	page 324
Cochran's <i>Q</i> test.	
CONT_TABLE Procedure	page 549
Sets up a table to generate pseudorandom numbers from a general continuous distribution.	
CONTINGENCY Function	page 261
Performs a chi-squared analysis of a two-way contingency table.	
COVARIANCES Function	page 190
Computes the sample variance-covariance or correlation matrix.	

CSTRENDS Function	page 310
Cox and Stuarts' sign test for trends in location and dispersion.	
DIFFERENCE Function	page 382
Differences a seasonal or nonseasonal time series.	
DISCR_ANALYSIS Procedure	page 437
Perform discriminant function analysis.	
DISCR_TABLE Function	page 553
Sets up a table to generate pseudorandom numbers from a general discrete distribution.	
EXACT_ENUM Function	page 273
Exact probabilities in a table; total enumeration.	
EXACT_NETWORK Function	page 275
Exact probabilities in a table.	
FACTOR_ANALYSIS Function	page 428
Extracts initial factor-loading estimates in factor analysis.	
FAURE_INIT Function	page 564
Initializes the structure used for computing a shuffled Faure sequence.	
FAURE_NEXT_PT Function	page 567
Generates a shuffled Faure sequence.	
FCDF Function	page 487
Evaluates the F distribution function. Using a keyword, the inverse of the F distribution function can be evaluated.	
FREQTABLE Function	page 36
Tallies observations into a one-way frequency table.	
FRIEDMANS_TEST Function	page 319
Friedman's test.	
GAMMA_ADV Function	page 585
Evaluate the real gamma function.	
GAMMACDF Function	page 492
Evaluates the gamma distribution function.	
GAMMAI Function	page 587
Evaluate the incomplete gamma function.	
GARCH Function	page 401
Compute estimates of the parameters of a GARCH(p,q)	

model	
HYPERGEOCDF Function	page 500
Evaluates the hypergeometric distribution function.	
HYPOTH_PARTIAL Function	page 141
Constructs an equivalent completely testable multivariate general linear hypothesis $H\beta U = G$ from a partially testable hypothesis $H_p\beta U = G_p$.	
HYPOTH_SCPH Function	page 147
Computes the matrix of sums of squares and crossproducts for the multivariate general linear hypothesis $H\beta U = G$ given the regression fit.	
HYPOTH_TEST Function	page 151
Performs tests for a multivariate general linear hypothesis $H\beta U = G$ given the hypothesis sums of squares and crossproducts matrix S_H .	
K_MEANS Function	page 419
Performs a K-means (centroid) cluster analysis.	
KALMAN Procedure	page 406
Performs Kalman filtering and evaluates the likelihood function for the state-space model.	
KOLMOGOROV1 Function	page 342
One-sample continuous data Kolmogorov-Smirnov.	
KOLMOGOROV2 Function	page 345
Two-sample continuous data Kolmogorov-Smirnov.	
KTRENDS Function	page 326
K-sample trends test.	
KW_TEST Function	page 317
Kruskal-Wallis test.	
LACK_OF_FIT Function	page 398
Lack-of-fit test based on the correlation function	
LN BETA Function	page 584
Evaluate the log of the real beta function.	
LNGAMMA Function	page 588
Evaluate the logarithm of the absolute value of the gamma function.	
LNORMREGRESS Function	page 169

	Fits a multiple linear regression model using criteria other than least squares. Namely, LNORMREGRESS allows the user to choose Least Absolute Value (L_1), Least L_p norm (L_p), or Least Maximum Value (Minimax or L_∞) method of multiple linear regression.	
MACHINE Function	Returns information describing the computer's arithmetic.	page 573
MULTICOMP Function	Performs Student-Newman-Keuls multiple-comparisons test.	page 230
MULTIPREDICT Function	Computes predicted values, confidence intervals, and diagnostics after fitting a regression model.	page 93
MULTIREGRESS Function	Fits a multiple linear regression model using least squares and optionally compute summary statistics for the regression model.	page 77
MVAR_NORMALITY Function	Mardia's test for multivariate normality.	page 347
NCTRENDS Function	Noether's test for cyclical trend.	page 308
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.	page 160
NONLINREGRESS Function	Fits a nonlinear regression model.	page 132
NORM1SAMP Function	Computes statistics for mean and variance inferences using a sample from a normal population.	page 25
NORM2SAMP Function	Computes statistics for mean and variance inferences using samples from two independently normal populations.	page 29
NORMALCDF Function	Evaluates the standard normal (Gaussian) distribution function. Using a keyword, the inverse of the standard normal (Gaussian) distribution can be evaluated.	page 478

NORMALITY Function	page 339
Performs a test for normality.	
PARTIAL_AC Function	page 395
Sample partial autocorrelation function	
PARTIAL_COV Function	page 194
Partial correlations and covariances.	
POISSONCDF Function	page 502
Evaluates the Poisson distribution function.	
POLYPREDICT Function	page 125
Computes predicted values, confidence intervals, and diagnostics after fitting a polynomial regression model.	
POLYREGRESS Function	page 118
Performs a polynomial least-squares regression.	
POOLED_COV Function	page 199
Pooled covariance matrix.	
PRINC_COMP Function	page 423
Computes principal components.	
RAND_GEN_CONT Function	page 551
Generates pseudorandom numbers from a general continuous distribution.	
RAND_GEN_DISCR Function	page 557
Generates pseudorandom numbers from a general discrete distribution using an alias method or optionally a table lookup method.	
RANDOM Function	page 518
Generates pseudorandom numbers. The default distribution is a uniform (0, 1) distribution, but many different distributions can be specified through the use of keywords.	
RANDOM_ARMA Function	page 560
Generate pseudorandom ARMA process numbers	
RANDOM_FROM_DATA Function	page 547
Generates pseudorandom numbers from a multivariate distribution determined from a given sample.	
RANDOM_NPP Function	page 537
Generates pseudorandom numbers from a nonhomogeneous Poisson process.	
RANDOM_ORDER Function	page 540
Generates pseudorandom order statistics from a standard normal distribution.	

RANDOM_ORTH_MAT Function	page 543
Generates a pseudorandom orthogonal matrix or a correlation matrix	
RANDOM_SAMPLE Function	page 545
Generates a simple pseudorandom sample from a finite population	
RANDOM_TABLE Function	page 553
Sets or retrieves the current table used in either the shuffled or GFSR random number generator	
RANDOM_TABLE_TWOWAY Function	page 542
Generates a pseudorandom two-way table.	
RANDOMNESS_TEST Function	page 352
Runs test, Paris-serial test, d^2 test or triplets tests.	
RANDOMOPT	page 510
Uses keywords to set or retrieve the random number seed or to select the uniform (0, 1) multiplicative, congruential pseudorandom-number generator.	
RANKS Function	page 48
Computes the ranks, normal scores, or exponential scores for a vector of observations.	
REGRESSORS Function	page 70
Generates regressors for a general linear model.	
ROBUST_COV Function	page 202
Robust estimate of covariance matrix.	
SIGNTEST Function	page 296
Performs a sign test.	
SIMPLESTAT Function	page 19
Computes basic univariate statistics.	
SORTDATA Function	page 42
Sorts observations by specified keys, with option to tally cases into a multiway frequency table.	
STATDATA Function	page 578
Retrieves commonly analyzed data sets.	
STEPWISE Procedure	page 109
Builds multiple linear regression models using forward, backward, or stepwise selection.	
SURVIVAL_GLM Function	page 450
Analyzes survival data using a generalized linear model	

and estimates using various parametric modes.

TCDF Function page [489](#)

Evaluates the Student's t distribution function.

TIE_STATS Function page [315](#)

Tie statistics.

WILCOXON Function page [300](#)

Performs a Wilcoxon rank sum test.

Index

A

- alpha-factor analysis method 434
- analysis of variance 211
 - factorial design 221
 - general linear model 70
 - n -way design 221
 - one-way design 214
- ANOVA, *see* analysis of variance
- ANSI/IEEE 754-1985 575
- ARMA
 - least-squares procedure 366, 367
 - method-of-moments procedure 367
 - stationary 384
- association, measures of 267
- asymptotic variances 425
- autoregressive parameters 370

B

- backcasting 367
- backward difference operator 383
- backward glance 114
- backward selection 109
- balanced experimental design 242
- basic uniform generator 507
- beta distribution 524, 535
- beta functions 581, 583, 584
- binomial coefficient 580
- binomial distribution 525
- binomial distributions 505, 506, 542, 543, 545, 547, 553, 557
- binomial probability 296
- Blom normal scores 50
- Bonferroni method 216

C

- cauchy distribution 526
- characteristic roots 424
- characteristic vectors 424
- chi-square distribution 526
- chi-squared
 - analysis 261
 - goodness-of-fit test 334, 336
 - measures relating to 264
 - statistic 264
 - test 26, 31, 261
- chi-squared statistics 259
- chi-squared test 334
- classification model, one-way 212
- classification variables 70
- cluster analysis 417, 419
- Cochran Q test 324
- coefficient of variation 22
- compiler 572, 591
- confidence intervals 93, 125
 - Bonferroni method 213, 214, 216
 - Dunn-Sidák method 213, 214, 216
 - means 94
 - One-at-a-Time t (Fisher's LSD) method 213, 214, 217
 - prediction 94
 - Scheffé method 94, 213, 214, 216
 - Tukey method 213, 214, 215
 - Tukey-Kramer method 213, 214, 215
- constants
 - computer 573
- contingency coefficient 262, 264, 353
- contingency tables 273
 - two-way 261
- continuous variables 70
- Cook's D statistics 94, 126

Cornish-Fisher expansion 490
correlation coefficient
 multiple 80
correlation matrix 190, 543, 547
correlations 190, 194
counts 19, 42
covariances 190
 sample 425
Cox and Stuart sign test 310
Cramer's V 264
curvilinear regression 120

D

data sets, statistical 571
 retrieving 578
degrees of freedom
 for error 79, 112, 119
 for the model 79, 112, 119
 total corrected 79, 112, 119
DFFITS statistics 66, 94, 126
diagnostics 93, 125
discrete uniform distribution 530
distribution functions
 beta probability 495
 binomial distribution 497
 binomial probability 498
 chi-squared, noncentral 482
 F distribution 487
 gamma distribution 492
 hypergeometric 500
 normal
 bivariate 480
 Gaussian 478
 inverse 478
 inverse 478
 Poisson 502
 Student's t 489
dummy variables 71
Dunn-Sidak method 216

E

eigensystem analysis 418
Erlang distribution 493
error handling
 informational error codes 589
errors

alert xv
fatal xv
note xv
terminal xv
warning xv
excess, coefficient of 19, 22
exponential distribution 523
exponential mix distribution 527
exponential order statistics 51
exponential scores 48

F

F test statistic 31, 33
factor analysis 417, 418, 428
factorial design, balanced 221
factor-loading estimates 428
fatal errors xv
Faure 565, 568
Faure sequence 564, 567
 faure_next_point 567
finite differences, forward 136
Fisher's LSD 217
forecasts
 GARCH 401
forward finite differences 136
forward selection 109
frequency tables
 multiway 42
 one-way 36
frequency tabulation 44
Friedman's test 319
F-statistic 79

G

gamma distribution 524
gamma functions 585, 587, 588
gamma statistic 262
GARCH
 (Generalized Autoregressive
 Conditional Heteroskedastic)
 401
general discrete distribution 498, 506,
 540, 542, 545, 549, 551, 553, 557,
 558
general distributions 334
general linear models 58, 70

generalized feedback shift register method 507
generalized linear models 260
generators
 basic uniform 507
 random-number 17
 shuffled 508
geometric distribution 527
GFSR 510
GFSR generator 508
GFSR method 507
Givens transformations 81
Goodman and Kruskal τ 268
 for columns 262
 for rows 262
goodness-of-fit tests 334
Gray code 566, 570
G-squared test 261

H

hypergeometric distribution 528

I

IEEE arithmetic 576
image analysis method 430, 433
indicator variables 71
inferences about the mean 32
Informational Error xiv
inverse
 g_3 85
 generalized 84
 Moore-Penrose 84

J

Jacobian matrix 133

K

Kalman filtering 406
Kappa analysis 259
kappa statistic 263, 269
Kendall's τ_b 262
key sort 44
K-means analysis 419
Kolmogorov one-sample test 342

Kolmogorov two-sample test 345
Kruskal-Wallis test 263, 268
k-sample trends test 326
kurtosis 19, 22

L

lack-of-fit statistics 119
lack-of-fit tests 65
Least Absolute Value 69
Least Maximum Value 69
Least Squares
 Alternatives
 Least Absolute Value 69
 Least Maximum Value 69
 Lp Norm 69
least-squares fit 77, 242, 308, 310,
 315, 319, 342, 345
 weighted 87
least-squares method 432
 generalized 430
 unweighted 430
 weighted 62
Lebesgue measure 565, 568
Levenberg-Marquardt algorithm, modified
 136
leverages 66, 94
library version 572, 591
Lilliefors test 339, 340
linear dependence 61
linear regression
 multiple 56
 simple 56
linear trend test 269
linearly dependent regressors 83
logarithmic distribution 528
lognormal distribution
 random numbers
 lognormal distribution 529
low-discrepancy 566, 569
Lp Norm 69

M

machine constants 571
MAD, *see* median absolute deviation 23
Mallows C_p criterion 101
Mann-Whitney U test 302

maximum 19, 22
 maximum likelihood estimates 412
 maximum likelihood method 430, 432
 McNemar test 263, 269
 mean 19, 22, 25, 79, 112, 119
 exact 262, 353
 for two normal populations 29
 inferences about 32
 lower confidence limit 19
 normal population 25
 return value 26
 upper confidence limit 19
 mean square
 error 79, 112, 119
 model 79, 112, 119
 measures of
 association 265
 prediction 267
 uncertainty 267
 measures of association 259
 median 23
 median absolute deviation 23
 method of provisional means 192
 minimum 19, 22
 missing values xiv, 69
 models
 general linear 70
 multiple linear regression 77, 100
 nonlinear regression 62, 132
 polynomial 58
 polynomial regression 125
 regression 93
 Moore-Penrose inverses 84
 multiple linear regression models 56, 70, 77, 100, 109, 242, 308, 310, 315, 319, 342, 345
 multiple-comparisons test
 Student-Newman-Keuls 230
 multiplicative congruential generator 507
 multiplicative generator 507
 multivariate analysis
 cluster analysis 419
 factor analysis 428
 principal components 423
 multivariate distribution 506, 547
 multivariate normal distribution 521, 525, 536
 multiway frequency table 42

N

NaN (Not a Number) xiv, 69
 negative binomial 529
 nested random model 211
 Noether test 308
 noncentral chi-squared distribution function 482
 nonlinear regression models 62, 132
 nonuniform generators 509
 normal distribution 523
 normal populations
 mean 25
 variances 25
 normal scores 48
 normality test 339
 numerical ranking 48

O

observations, number of 19
 One-at-a-Time t method 217
 one-way classification model 212
 one-way frequency table 36
 operating system 572, 591
 optimal prediction 263
 overflow xiv

P

partial correlations 194
 partial covariances 194
 phi 262, 264, 353
 Poisson distribution 523, 535
 polynomial models 58
 polynomial regression models 125
 pooled variances 30
 predicted values 93, 125, 135
 prediction coefficient 267
 principal components 417, 423
 principal components method 430, 431
 principal factor method 430, 431
 probability distribution functions, *see* distribution functions 477
 product moment correlation 262
 provisional means, method of 192
 pseudorandom number generators 334

pseudorandom numbers 505, 506, 537, 547,
549, 551, 553, 557, 558
pseudorandom order statistics 505, 540
pseudorandom orthogonal matrix 505, 543
pseudorandom sample 506, 545
 p -values 79, 112, 119, 265

R

R matrix 61, 84, 135
 R^2 criterion 79, 100, 112, 119
adjusted 79, 100, 112, 119
random numbers 506
beta distribution 524, 535
binomial distribution 525
cauchy distribution 526
chi-squared distribution 526
control the seed 510
discrete uniform distribution 530
exponential distribution 523
exponential mix distribution 527
gamma distribution 524
generate pseudorandom numbers 518
geometric distribution 527
hypergeometric distribution 528
logarithmic distribution 528
multivariate normal distribution 521, 525,
536
negative binomial 529
normal distribution 523
Poisson distribution 523, 535
select the form 510
Student's t distribution 530
triangular distribution 530
von Mises distribution 530
Weibull distribution 531
randomness test 352
range 19, 22
ranks 48
regression
all best 100
curvilinear 118
general linear model 70
multiple linear 77
nonlinear 132
polynomial least-squares 118
simple linear 56
stepwise 109

regression coefficients 79, 102, 113,
132
regression models 56, 93
regression simple linear 77
regressors 70
residuals 94, 126, 135
deleted 66, 94, 126
jackknife 66
standardized 66, 94, 126

S

sample covariance 425
Satterthwaite's procedure 33
Savage scores 49
scaling results of RANDOM 535
Scheffé confidence intervals 94
Scheffé method 216
serial number 572, 591
Shapiro-Wilk W test 339, 340
shuffled generators 508
shuffling 510
sign test 296
skewness, coefficient of 19, 22
Snedecor's F random variable 487
Somers' D
for columns 262
for rows 262
sorting 42, 44
key 44
Spearman rank correlation 262
standard deviation 19, 26, 27, 31, 79,
112, 119
exact 262, 353
standard errors 265
standard errors, for characteristic roots
424
state vector 406
statespace model 406
stationary ARMA 384
stepwise selection 109
Stuart's τ_c 262
Student's t distribution 530
Student's t distribution function 489
Student-Newman-Keuls multiple-compar-
isons test 230
sum of squares
for error 79, 112, 119
for the model 79, 112, 119

- sequential 84, 119
- total corrected 79, 112, 119
- summary statistics 63, 77
- sum-of-squares and crossproducts matrix 190
- sums-of-squares
 - within-cluster 420
- system variables
 - !Cmath_Err xiv
 - !Error xiv
 - !Quiet xiv

T

- t test statistic 26, 30, 32
- terminal errors xv
- test for linear trend 269
- test for normality 339
- tests for randomness 334
- tie statistics 315
- time series
 - autoregressive parameters 366
 - backward differences 383
 - Box-Jenkins forecasts 374
 - difference 382
 - moving average parameters 366
- transformations 68
- triangular distribution 530
- trust region 136
- Tucker reliability coefficient 430
- Tukey method 215
- Tukey normal scores 51
- Tukey-Kramer method 215

U

- uncertainty
 - coefficients 262, 268
 - measures of 267
- unit circle 520
- univariate statistics 19, 280, 450

V

- Van der Waerden normal scores 51
- variable selection 57, 100, 109
- variables
 - classification 70
 - continuous 70

- dummy 71
- indicator 71
- variance-covariance matrix 190, 521
- variances 19, 22, 25, 190
 - asymptotic 425
 - for two normal populations 29
 - inferences about 33
 - inflation factor 80
 - lower confidence limit 20
 - normal population 25
 - upper confidence limit 20
- variation, coefficient of 19, 22, 79, 112, 119
- von Mises distribution 530

W

- warning errors xv
- Weibull distribution 531
- weighted least-squares fit 62, 84, 87
- Wilcoxon rank sum test 300
- Wilson-Hilferty approximation 483