

GAUSSTM

Language Reference

Information in this document is subject to change without notice and does not represent a commitment on the part of Aptech Systems, Inc. The software described in this document is furnished under a license agreement or nondisclosure agreement. The software may be used or copied only in accordance with the terms of this agreement. The purchaser may make one copy of the software for backup purposes. No part of this manual may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying or recording, for any purpose other than the purchaser's personal use without the written permission of Aptech Systems, Inc.

© 1984–2001 Aptech Systems, Inc. All rights reserved.

GAUSS and GAUSS Light are trademarks of Aptech Systems, Inc.
GEM is a trademark of Digital Research, Inc.
Lotus is a trademark of Lotus Development Corp.
HP Laser Jet and HP-GL are trademarks of Hewlett-Packard Corp.
PostScript is a trademark of Adobe Systems Inc.
IBM is a trademark of International Business Machines Corporation.
Hercules is a trademark of Hercules Computer Technology, Inc.
GraphicC is a trademark of Scientific Endeavors Corporation.
Tektronix is a trademark of Tektronix, Inc.
Windows is a registered trademark of Microsoft Corporation.
Other trademarks are the property of their respective owners.

Part Number: 1295
Version 3.6
Revised September 1, 2001

Contents

Introduction	1-1
Documentation Conventions	1-2
Using This Manual	1-2
Global Control Variables	1-3
Commands by Category	2-1
Mathematical Functions	2-1
Finance Functions.....	2-10
Matrix Manipulation.....	2-12
Data Handling	2-15
Compiler Control	2-18
Program Control.....	2-18
OS Functions	2-22
Workspace Management	2-23
Error Handling and Debugging	2-23
String Handling.....	2-24
Time and Date Functions	2-25
Console I/O	2-26
Output Functions.....	2-27
Graphics.....	2-28
Command Reference	3-1
Command Components	3-1
Obsolete Commands	A-1
Colors Appendix	B-1

Introduction 1

This GAUSSTM Language Reference describes each of the commands, procedures, and functions available in the GAUSS programming language. These functions can be divided into four categories:

- Mathematical, statistical, and scientific functions.
- Data handling routines, including data matrix manipulation and description routines, and file I/O.
- Programming statements, including branching, looping, display features, error checking, and shell commands.
- Graphics functions.

The first category contains those functions to be expected in a high-level mathematical language: trigonometric functions and other transcendental functions, distribution functions, random number generators, numerical differentiation and integration routines, Fourier transforms, Bessel functions, and polynomial evaluation routines. And, as a matrix programming language, GAUSS includes a variety of routines that perform standard matrix operations. Among these are routines to calculate determinants, matrix inverses, decompositions, eigenvalues and eigenvectors, and condition numbers.

Data handling routines include functions that return dimensions of matrices, and information about elements of data matrices, including functions to locate values lying in specific ranges or with certain values. Also under data handling routines fall all

those functions that create, save, open, and read from and write to GAUSS data sets. A variety of sorting routines that will operate on both numeric and character data are also available.

Programming statements are all of the commands that make it possible to write complex programs in GAUSS. These include conditional and unconditional branching, looping, file I/O, error handling, and system-related commands to execute OS shells and access directory and environment information.

The graphics functions of GAUSS Publication Quality Graphics (PQG) are a set of routines built on the graphics functions in GraphiC by Scientific Endeavors Corporation. GAUSS PQG consists of a set of main graphing procedures and several additional procedures and global variables for customizing the output.

Documentation Conventions

The following table describes how text formatting is used to identify GAUSS programming elements.

Text Style	Use	Example
regular text	narrative	"...text formatting is used..."
bold text	emphasis	"...not supported under UNIX ."
<i>italic text</i>	variables	"...If <i>vnames</i> is a string or has fewer elements than <i>x</i> has columns, it will be..."
monospace	code example	<pre>if scalerr(cm); cm = inv(x); endif;</pre>
monospace bold	Refers to a GAUSS programming element within a narrative paragraph.	"...as explained under create ..."

Using This Manual

Users who are new to GAUSS should make sure they have familiarized themselves with "Language Fundamentals" in the *User's Guide* before proceeding here. That chapter contains the basics of GAUSS programming.

In all, there are over 400 routines described in this GAUSS Language Reference. We suggest that new GAUSS users skim through Chapter 2, and then browse through Chapter 3, the main part of this manual. Here, users can familiarize themselves with the kinds of tasks that GAUSS can handle easily.

Chapter 2 gives a categorical listing of all functions in this GAUSS Language Reference, and a short discussion of the functions in each category. Complete syntax, description of input and output arguments, and general remarks regarding each function are given in Chapter 3.

If a function is an “extrinsic” (that is, part of the Run-Time Library), its source code can be found on the `.src` subdirectory. The name of the file containing the source code is given in Chapter 3 under the discussion of that function.

Global Control Variables

Several GAUSS functions use global variables to control various aspects of their performance. The files `gauss.ext`, `gauss.dec`, and `gauss.lcg` contain the **external** statements, **declare** statements, and **library** references to these globals. All globals used by the GAUSS Run-Time library begin with an underscore ‘_’.

Default values for these common globals can be found in the file `gauss.dec`, located on the `.src` subdirectory. The default values can be changed by editing this file.

Changing the Default Values

To permanently change the default setting of a common global, two files need to be edited: `gauss.dec` and `gauss.src`.

To change the value of the common global `__output` from 2 to 0, for example, edit the file `gauss.dec` and change the statement

```
declare matrix __output = 2;
```

to read

```
declare matrix __output = 0;
```

Also, edit the procedure **gausset**, located in the file `gauss.src`, and modify the statement

```
__output = 2;
```

similarly.

The Procedure `gausset`

The global variables affect your program, even if you have not set them directly in a particular command file. If you have changed them in a previous run, they will retain their changed values until you exit GAUSS or execute the **new** command.

The procedure **gausset** will reset the Run-Time Library globals to their default values:

```
gausset ;
```

If your program changes the values of these globals, you can use **gausset** to reset them whenever necessary. **gausset** resets the globals as a whole; you can write your own routine to reset specific ones.

Commands by Category **2**

Mathematical Functions

Scientific Functions

abs	Returns absolute value of argument.
arccos	Computes inverse cosine.
arcsin	Computes inverse sine.
atan	Computes inverse tangent.
atan2	Computes angle given a point x,y .
besselj	Computes Bessel function, first kind.
bessely	Computes Bessel function, second kind.
cos	Computes cosine.
cosh	Computes hyperbolic cosine.
exp	Computes the exponential function of x .
gamma	Computes gamma function value.
ln	Computes the natural log of each element.
lnfact	Computes natural log of factorial function.

log	Computes the \log_{10} of each element.
pi	Returns π .
sin	Computes sine.
sinh	Computes the hyperbolic sine.
spline1D	Computes a smoothing spline for a curve.
spline2D	Computes a smoothing spline for a surface.
sqrt	Computes the square root of each element.
tan	Computes tangent.
tanh	Computes hyperbolic tangent

All trigonometric functions take or return values in radian units.

Differentiation and Integration

gradp	Computes first derivative of a function.
hessp	Computes second derivative of a function.
intgrat2	Integrates a 2-dimensional function over a user-defined region.
intgrat3	Integrates a 3-dimensional function over a user-defined region.
intquad1	Integrates a 1-dimensional function.
intquad2	Integrates a 2-dimensional function over a user-defined rectangular region.
intquad3	Integrates a 3-dimensional function over a user-defined rectangular region.
intsimp	Integrates by Simpson's method.

gradp and **hessp** use a finite difference approximation to compute the first and second derivatives. Use **gradp** to calculate a Jacobian.

intquad1, **intquad2**, and **intquad3** use Gaussian quadrature to calculate the integral of the user-defined function over a rectangular region.

To calculate an integral over a region defined by functions of x and y , use **intgrat2** and **intgrat3**.

To get a greater degree of accuracy than that provided by **intquad1**, use **intsimp** for 1-dimensional integration.

Linear Algebra

balance	Balances a matrix.
----------------	--------------------

chol	Computes Cholesky decomposition, $X = Y'Y$.
choldn	Performs Cholesky downdate on an upper triangular matrix.
cholsol	Solves a system of equations given the Cholesky factorization of a matrix.
cholup	Performs Cholesky update on an upper triangular matrix.
cond	Computes condition number of a matrix.
crout	Computes Crout decomposition, $X = LU$ (real matrices only).
croutp	Computes Crout decomposition with row pivoting (real matrices only).
det	Computes determinant of square matrix.
detl	Computes determinant of decomposed matrix.
hess	Computes upper Hessenberg form of a matrix (real matrices only).
inv	Inverts a matrix.
invpd	Inverts a positive definite matrix.
invswp	Generalized sweep inverse.
lu	Computes LU decomposition with row pivoting (real and complex matrices).
null	Computes orthonormal basis for right null space.
nulll	Computes orthonormal basis for right null space.
orth	Computes orthonormal basis for column space x .
pinv	Generalized pseudo-inverse: Moore-Penrose.
qqr	QR decomposition: returns Q_1 and R .
qqre	QR decomposition: returns Q_1 , R , and a permutation vector, E .
qqrep	QR decomposition with pivot control: returns Q_1 , R , and E .
qr	QR decomposition: returns R .
qre	QR decomposition: returns R and E .
qrep	QR decomposition with pivot control: returns R and E .
qrsol	Solves a system of equations $Rx=b$ given an upper triangular matrix, typically the R matrix from a QR decomposition.
qrtsol	Solves a system of equations $R'x = b$ given an upper triangular matrix, typically the R matrix from a QR decomposition.
qtyr	QR decomposition: returns $Q'Y$ and R .

qtyre	QR decomposition: returns $Q'Y$, R , and E .
qtyrep	QR decomposition with pivot control: returns $Q'Y$, R , and E .
qyr	QR decomposition: returns QY and R .
qyre	QR decomposition: returns QY , R , and E .
qyrep	QR decomposition with pivot control: returns QY , R , and E .
rank	Computes rank of a matrix.
rcondl	Returns reciprocal of the condition number of last decomposed matrix.
rref	Computes reduced row echelon form of a matrix.
schur	Computes Schur decomposition of a matrix (real matrices only).
solpd	Solves a system of positive definite linear equations.
svd	Computes the singular values of a matrix.
svd1	Computes singular value decomposition, $X = USV'$.
svd2	Computes svd1 with compact U .

The decomposition routines are **chol** for Cholesky decomposition, **crout** and **croutp** for Crout decomposition, **qqr-qyrep** for QR decomposition, and **svd**, **svd1**, and **svd2** for singular value decomposition.

null, **null1**, and **orth** calculate orthonormal bases.

inv, **invpd**, **solpd**, **cholsol**, **qrsol**, and the “/” operator can all be used to solve linear systems of equations.

rank and **rref** will find the rank and reduced row echelon form of a matrix.

det, **det1**, and **cond** will calculate the determinant and condition number of a matrix.

Eigenvalues

eig	Computes eigenvalues of general matrix.
eigh	Computes eigenvalues of complex Hermitian or real symmetric matrix.
eighv	Computes eigenvalues and eigenvectors of complex Hermitian or real symmetric matrix.
eigv	Computes eigenvalues and eigenvectors of general matrix.

There are four eigenvalue-eigenvector routines. Two calculate eigenvalues only, and two calculate eigenvalues and eigenvectors. The types of matrices handled by these routines are:

General:	eig, eigv
Symmetric or Hermitian:	eigh, eighv

Polynomial Operations

polychar	Computes characteristic polynomial of a square matrix.
polyeval	Evaluates polynomial with given coefficients.
polyint	Calculates N^{th} order polynomial interpolation given known point pairs.
polymake	Computes polynomial coefficients from roots.
polymat	Returns sequence powers of a matrix.
polymult	Multiplies two polynomials together.
polyroot	Computes roots of polynomial from coefficients.

See also **recserrc**, **recsercp**, and **conv**.

Fourier Transforms

dfft	Computes discrete 1-D FFT.
dffti	Computes inverse discrete 1-D FFT.
fft	Computes 1- or 2-D FFT.
ffti	Computes inverse 1- or 2-D FFT.
fftm	Computes multi-dimensional FFT.
fftmi	Computes inverse multi-dimensional FFT.
fftn	Computes 1- or 2-D FFT using prime factor algorithm.
rfft	Computes real 1- or 2-D FFT.
rffti	Computes inverse real 1- or 2-D FFT.
rfftip	Computes inverse real 1- or 2-D FFT from packed format FFT.
rfftn	Computes real 1- or 2-D FFT using prime factor algorithm.
rfftnp	Computes real 1- or 2-D FFT using prime factor algorithm, returns packed format FFT.
rfftp	Computes real 1- or 2-D FFT, returns packed format FFT.

Random Numbers

rndbeta	Computes random numbers with beta distribution.
rndcon	Changes constant of the LC random number generator.
rndgam	Computes random numbers with gamma distribution.
rndi	Returns random integers, $0 \leq y < 2^{32}$.
rndKmbeta	Computes beta pseudo-random numbers.
rndKMgam	Computes gamma pseudo-random numbers.
rndKMi	Returns random integers, $0 \leq y < 2^{32}$.
rndKMn	Computes standard normal pseudo-random numbers.
rndKMnb	Computes negative binomial pseudo-random numbers.
rndKMp	Computes Poisson pseudo-random numbers.
rndKMu	Computes uniform pseudo-random numbers.
rndKMvm	Computes von Mises pseudo-random numbers.
rndLCbeta	Computes beta pseudo-random numbers.
rndLCgam	Computes gamma pseudo-random numbers.
rndLCi	Returns random integers, $0 \leq y < 2^{32}$.
rndLCn	Computes standard normal pseudo-random numbers.
rndLCnb	Computes negative binomial pseudo-random numbers.
rndLCp	Computes Poisson pseudo-random numbers.
rndLCu	Computes uniform pseudo-random numbers.
rndLCvm	Computes von Mises pseudo-random numbers.
rndmult	Changes multiplier of the LC random number generator.
rndn	Computes random numbers with Normal distribution.
rndnb	Computes random numbers with negative binomial distribution.
rndp	Computes random numbers with Poisson distribution.
rndseed	Changes seed of the LC random number generator.
rndu	Computes random numbers with uniform distribution.

Fuzzy Conditional Functions

dotfeq	Fuzzy $.==$
dotfge	Fuzzy $. \geq$
dotfgt	Fuzzy $. >$

dotfle	Fuzzy \leq
dotflt	Fuzzy $<$
dotfne	Fuzzy \neq
feq	Fuzzy $==$
fge	Fuzzy \geq
fgt	Fuzzy $>$
fle	Fuzzy \leq
flt	Fuzzy $<$
fne	Fuzzy \neq

The global variable **_fcomptol** controls the tolerance used for comparison. By default, this is 1e-15. The default can be changed by editing the file `fcompare.dec`.

Statistical Functions

acf	Computes sample autocorrelations.
conv	Computes convolution of two vectors.
corrm	Computes correlation matrix of a moment matrix.
corrvc	Computes correlation matrix from a variance-covariance matrix.
corrxx	Computes correlation matrix.
crossprd	Computes cross product.
design	Creates a design matrix of 0's and 1's.
dstat	Computes descriptive statistics of a data set or matrix.
loess	Computes coefficients of locally weighted regression.
meanc	Computes mean value of each column of a matrix.
median	Computes medians of the columns of a matrix.
moment	Computes moment matrix $(x'x)$ with special handling of missing values.
momentd	Computes moment matrix from a data set.
ols	Computes least squares regression of data set or matrix.
olsqr	Computes OLS coefficients using QR decomposition.
olsqr2	Computes OLS coefficients, residuals, and predicted values using QR decomposition.
pacf	Computes sample partial autocorrelations.
princomp	Computes principal components of a data matrix.

stdc	Computes standard deviation of the columns of a matrix.
toeplitz	Computes Toeplitz matrix from column vector.
varmall	Computes the log-likelihood of a Vector ARMA model.
varmares	Computes the residuals of a Vector ARMA model.
vcm	Computes a variance-covariance matrix from a moment matrix.
vcx	Computes a variance-covariance matrix from a data matrix.

Advanced statistics and optimization routines are available in the GAUSS Applications programs. (Contact Aptech Systems for more information.)

Optimization and Solution

eqsolve	Solves a system of nonlinear equations.
QNewton	Optimizes a function using the BFGS descent algorithm.
QProg	Solves the quadratic programming problem.
sqpSolve	Solves the nonlinear programming problem using a sequential quadratic programming method.

Statistical Distributions

cdfbeta	Computes integral of beta function.
cdfbvn	Computes lower tail of bivariate Normal cdf.
cdfchic	Computes complement of cdf of χ^2 .
cdfchii	Computes χ^2 abscissae values given probability and degrees of freedom.
cdfchinc	Computes integral of noncentral χ^2 .
cdffc	Computes complement of cdf of F .
cdffnc	Computes integral of noncentral F .
cdfgam	Computes integral of incomplete Γ function.
cdfmvn	Computes multivariate Normal cdf.
cdfn	Computes integral of Normal distribution: lower tail, or cdf.
cdfn2	Computes interval of Normal cdf.
cdfnc	Computes complement of cdf of Normal distribution (upper tail).
cdftc	Computes complement of cdf of t -distribution.
cdftnc	Computes integral of noncentral t -distribution.

cdftvn	Computes lower tail of trivariate Normal cdf.
erf	Computes Gaussian error function.
erfc	Computes complement of Gaussian error function.
lncdfbvn	Computes natural log of bivariate Normal cdf.
lncdfmvn	Computes natural log of multivariate Normal cdf.
lncdfn	Computes natural log of Normal cdf.
lncdfn2	Computes natural log of interval of Normal cdf.
lncdfnc	Computes natural log of complement of Normal cdf.
lnpdfmvn	Computes multivariate Normal log-probabilities.
lnpdfn	Computes Normal log-probabilities.
pdfn	Computes standard Normal probability density function.

Series and Sequence Functions

recserar	Computes autoregressive recursive series.
recsercp	Computes recursive series involving products.
recserrc	Computes recursive series involving division.
sega	Creates an additive sequence.
seqm	Creates a multiplicative sequence.

Precision Control

base10	Converts number to x.xxx and a power of 10.
ceil	Rounds up towards $+\infty$.
floor	Rounds down towards $-\infty$.
prcsn	Sets computational precision for matrix operations.
round	Rounds to the nearest integer.
trunc	Truncates toward 0.

All calculations in GAUSS are done in double precision, with the exception of some of the intrinsic functions on OS/2 and DOS. These may use extended precision (18-19 digits of accuracy). Use **prcsn** to change the internal accuracy used in these cases.

round, **trunc**, **ceil**, and **floor** convert floating point numbers into integers. The internal representation for the converted integer is double precision (64 bits).

Each matrix element in memory requires 8 bytes of memory.

Finance Functions

AmericanBinomCall	American binomial method Call.
AmericanBinomCall_Greeks	American binomial method call Delta, Gamma, Theta, Vega, and Rho.
AmericanBinomCall_ImpVol	Implied volatilities for American binomial method calls.
AmericanBinomPut	American binomial method Put.
AmericanBinomPut_Greeks	American binomial method put Delta, Gamma, Theta, Vega, and Rho.
AmericanBinomPut_ImpVol	Implied volatilities for American binomial method puts.
AmericanBSCall	American Black and Scholes Call.
AmericanBSCall_Greeks	American Black and Scholes call Delta, Gamma, Omega, Theta, and Vega.
AmericanBSCall_ImpVol	Implied volatilities for American Black and Scholes calls.
AmericanBSPut	American Black and Scholes Put.
AmericanBSPut_Greeks	American Black and Scholes put Delta, Gamma, Omega, Theta, and Vega.
AmericanBSPut_ImpVol	Implied volatilities for American Black and Scholes puts.
annualTradingDays	Compute number of trading days in a given year.
elapsedTradingDays	Compute number of trading days between two dates inclusively.
EuropeanBinomCall	European binomial method call.
EuropeanBinomCall_Greeks	European binomial method call Delta, Gamma, Theta, Vega and Rho.
EuropeanBinomCall_ImpVol	Implied volatilities for European binomial method calls.
EuropeanBinomPut	European binomial method Put.
EuropeanBinomPut_Greeks	European binomial method put Delta, Gamma, Theta, Vega, and Rho.
EuropeanBinomPut_ImpVol	Implied volatilities for European binomial method puts.
EuropeanBSCall	European Black and Scholes Call.

EuropeanBSCall_Greeks	European Black and Scholes call Delta, Gamma, Omega, Theta, and Vega.
EuropeanBSCall_ImpVol	Implied volatilities for European Black and Scholes calls.
EuropeanBSPut	European Black and Scholes Put.
EuropeanBSPut_Greeks	European Black and Scholes put Delta, Gamma, Omega, Theta, and Vega.
EuropeanBSPut_ImpVol	Implied volatilities for European Black and Scholes puts.
getNextTradingDay	Returns the next trading day.
getNextWeekDay	Returns the next day that is not on a weekend.
getPreviousTradingDay	Returns the previous trading day.
getPreviousWeekDay	Returns the previous day that is not on a weekend.

Matrix Manipulation

Creating Vectors and Matrices

editm	Simple matrix editor.
eye	Creates identity matrix.
let	Creates matrix from list of constants.
matalloc	Allocates a matrix with unspecified contents.
matinit	Allocates a matrix with unspecified contents.
medit	Full-screen spreadsheet-like matrix editor.
ones	Creates a matrix of ones.
zeros	Creates a matrix of zeros.

Use **zeros** or **ones** to create a constant vector or matrix.

medit is a full-screen editor that can be used to create matrices to be stored in memory, or to edit matrices that already exist.

Matrices can also be loaded from an ASCII file, from a GAUSS matrix file, or from a GAUSS data set. (See “Procedures and Keywords” in the *User Guide* for more information.)

Loading and Storing Matrices

loadd	Loads matrix from data set.
loadm	Loads matrix from ASCII or matrix file.
save	Saves matrix to matrix file.
saved	Saves matrix to data set.

Size, Ranking, and Range

cols	Returns number of columns in a matrix.
colsf	Returns number of columns in an open data set.
counts	Returns number of elements of a vector falling in specified ranges.
countwts	Returns weighted count of elements of a vector falling in specified ranges.
cumprodc	Computes cumulative products of each column of a matrix.
cumsumc	Computes cumulative sums of each column of a matrix.

indexcat	Returns indices of elements falling within a specified range.
maxc	Returns largest element in each column of a matrix.
maxindc	Returns row number of largest element in each column of a matrix.
minc	Returns smallest element in each column of a matrix.
minindc	Returns row number of smallest element in each column of a matrix.
prodc	Computes the product of each column of a matrix.
rankindx	Returns rank index of Nx1 vector. (Rank order of elements in vector.)
rows	Returns number of rows in a matrix.
rowsf	Returns number of rows in an open data set.
sumc	Computes the sum of each column of a matrix.

These functions are used to find the minimum, maximum and frequency counts of elements in matrices.

Use **rows** and **cols** to find the number of rows or columns in a matrix. Use **rowsf** and **colsf** to find the numbers of rows or columns in an open GAUSS data set.

Sparse Matrix Functions

denseSubmat	Returns dense submatrix of sparse matrix.
isSparse	Tests whether a matrix is a sparse matrix.
sparseCols	Returns number of columns in sparse matrix.
sparseEye	Creates sparse identity matrix.
sparseFD	Converts dense matrix to sparse matrix.
sparseFP	Converts packed matrix to sparse matrix.
sparseHConcat	Horizontally concatenates sparse matrices.
sparseNZE	Returns the number of nonzero elements in sparse matrix.
sparseOnes	Generates sparse matrix of ones and zeros.
sparseRows	Returns number of rows in sparse matrix.
sparseSet	Resets sparse library globals.
sparseSolve	Solves $Ax = B$ for x where A is a sparse matrix.
sparseSubmat	Returns sparse submatrix of sparse matrix.
sparseTD	Multiplies sparse matrix by dense matrix.
sparseTrTD	Multiplies sparse matrix transposed by dense matrix.

sparseVConcat Vertically concatenates sparse matrices.

Miscellaneous Matrix Manipulation

complex	Creates a complex matrix from two real matrices.
delif	Deletes rows from a matrix using a logical expression.
diag	Extracts the diagonal of a matrix.
diagrv	Puts a column vector into the diagonal of a matrix.
exctsmpl	Creates a random subsample of data set, with replacement.
imag	Returns the imaginary part of a complex matrix.
intrsect	Returns the intersection of two vectors.
lowmat	Returns the main diagonal and lower triangle.
lowmat1	Returns a main diagonal of 1's and the lower triangle.
real	Returns the real part of a complex matrix.
reshape	Reshapes a matrix to new dimensions.
rev	Reverses the order of rows of a matrix.
rotater	Rotates the rows of a matrix, wrapping elements as necessary.
selif	Selects rows from a matrix using a logical expression.
setdif	Returns elements of one vector that are not in another.
shiftr	Shifts rows of a matrix, filling in holes with a specified value.
submat	Extracts a submatrix from a matrix.
trimr	Trims rows from top or bottom of a matrix.
union	Returns the union of two vectors.
upmat	Returns the main diagonal and upper triangle.
upmat1	Returns a main diagonal of 1's and the upper triangle.
vec	Stacks columns of a matrix to form a single column.
vech	Reshapes the lower triangular portion of a symmetric matrix into a column vector.
vecr	Stacks rows of a matrix to form a single column.
xpnd	Expands a column vector into a symmetric matrix.

vech and **xpnd** are complementary functions. **vech** provides an efficient way to store a symmetric matrix; **xpnd** expands the stored vector back to its original symmetric matrix.

delif and **selif** are complementary functions. **delif** deletes rows of a matrix based on a logical comparison; **selif** selects rows based on a logical comparison.

lowmat, **lowmat1**, **upmat**, and **upmat1** extract triangular portions of a matrix.

To delete rows that contain missing values from a matrix in memory, see **packr**.

Data Handling

Data Sets

close	Closes an open data set (.dat file).
closeall	Closes all open data sets.
create	Creates and opens a data set.
eof	Tests for end of file.
iscplx	Returns whether a data set is real or complex.
load	Loads a small data set.
open	Opens an existing data set.
readr	Reads rows from open data set.
saved	Creates small data sets.
seekr	Moves pointer to specified location in open data set.
typef	Returns the element size (2, 4, or 8 bytes) of data in open data set.
writer	Writes matrix to an open data set.

These functions all operate on GAUSS data sets (.dat files). (See “File I/O” in the *User’s Guide* for more information.)

To create a GAUSS data set from a matrix in memory, use **saved**. To create a data set from an existing one, use **create**. To create a data set from a large ASCII file, use the utility **atog**. (See “Utilities” in the *User’s Guide*.)

Data sets can be opened, read from, and written to using **open**, **readr**, **seekr** and **writer**. Test for the end of a file using **eof**, and close the data set using **close** or **closeall**.

The data in data sets may be specified as character or numeric. (See “File I/O” in the *User’s Guide*.) See also **create** and **vartypef**.

typef returns the element size of the data in an open data set.

Data Set Variable Names

getname	Returns column vector of variable names in a data set.
----------------	--

getnamef	Returns string array of variable names in a data set.
indcv	Returns column numbers of variables within a data set.
indices	Retrieves column numbers and names from a data set.
indices2	Similar to indices , but matches columns with names for dependent and independent variables.
makevars	Decomposes matrix to create column vectors.
mergevar	Concatenates column vectors to create larger matrix.
setvars	Creates globals using the names in a data set.
vartype	Returns column vector of variable types (numeric/character) in a data set.
vartypef	Returns column vector of variable types (numeric/character) in a data set.

Use **getnamef** to retrieve the variable names associated with the columns of a GAUSS data set, and **vartypef** to retrieve the variable types. Use **makevars** and **setvars** to create global vectors from those names. Use **indices** and **indices2** to match names with column numbers in a data set.

getname and **vartype** are supported for backwards compatibility.

Data Coding

code	Codes the data in a vector by applying a logical set of rules to assign each data value to a category.
dummy	Creates a dummy matrix, expanding values in vector to rows with ones in columns corresponding to true categories and zeros elsewhere.
dummybr	Similar to dummy .
dummydn	Similar to dummy .
issmiss	Returns 1 if matrix has any missing values, 0 otherwise.
isinfnanmiss	Returns true if the argument contains an infinity, NaN, or missing value.
miss	Changes specified values to missing value code.
missex	Changes elements to missing value using logical expression.
missrv	Changes missing value codes to specified values.
msym	Sets symbol to be interpreted as missing value.
packr	Deletes rows with missing values.

recode	Similar to code , but leaves the original data in place if no condition is met.
scalinfnanmiss	Returns true if the argument is a scalar infinity, NaN, or missing value.
scalmiss	Tests whether a scalar is the missing value code.
subscat	Simpler version of recode , but uses ascending bins instead of logical conditions.
substute	Similar to recode , but operates on matrices.

code, **recode**, and **subscat** allow the user to code data variables and operate on vectors in memory. **substute** operates on matrices, and **dummy**, **dummybr**, and **dummydn** create matrices.

missex, **missrv**, and **miss** should be used to recode missing values.

Sorting and Merging

intrleav	Produces one large sorted data file from two smaller sorted files having the same keys.
mergeby	Produces one large sorted data file from two smaller sorted files having a single key column in common.
sortc	Quick-sorts rows of matrix based on numeric key.
sortcc	Quick-sorts rows of matrix based on character key.
sortd	Sorts data set on a key column.
sorthc	Heap-sorts rows of matrix based on numeric key.
sorthcc	Heap-sortsrows of matrix based on character key.
sortind	Returns a sorted index of a numeric vector.
sortindc	Returns a sorted index of a character vector.
sortmc	Sorts rows of matrix on the basis of multiple columns.
uniqindx	Returns a sorted unique index of a vector.
unique	Removes duplicate elements of a vector.

sortc, **sorthc**, and **sortind** operate on numeric data only. **sortcc**, **sorthcc**, and **sortindc** operate on character data only.

Sortd, **sortmc**, **unique**, and **uniqindx** operate on both numeric and character data.

Use **sortd** to sort the rows of a data set on the basis of a key column.

Both **intrleav** and **mergeby** operate on data sets.

Compiler Control

#define	Defines a case-insensitive text-replacement or flag variable.
#definecs	Defines a case-sensitive text-replacement or flag variable.
#else	Alternates clause for #if-#else-#endif code block.
#endif	End of #if-#else-#endif code block.
#ifdef	Compiles code block if a variable has been #define 'd.
#ifdos	Compiles code block if running DOS.
#iflight	Compiles code block if running GAUSS Light.
#ifndef	Compiles code block if a variable has not been #define 'd.
#ifos2win	Compiles code block if running OS/2 or Windows.
#ifunix	Compiles code block if running UNIX.
#include	Includes code from another file in program.
#linesoff	Compiles program without line number and file name records.
#lineson	Compiles program with line number and file name records.
#srcfile	Inserts source file name record at this point (currently used when doing data loop translation).
#srcline	Inserts source file line number record at this point (currently used when doing data loop translation).
#undef	Undefines a text-replacement or flag variable.

These commands are compiler directives. That is, they do not generate GAUSS program instructions; rather, they are instructions that tell GAUSS how to process a program during compilation. They determine what the final compiled form of a program will be. They are not executable statements and have no effect at run-time. (See “Language Fundamentals” in the *User's Guide* for more information.)

Program Control

Execution Control

end	Terminates a program and close all files.
pause	Pauses for the specified time.
run	Runs a program in a text file.
sleep	Sleeps for the specified time.
stop	Stops a program and leave files open.

system Quits and returns to the OS.

Both **stop** and **end** will terminate the execution of a program; **end** will close all open files, and **stop** will leave those files open. Neither **stop** nor **end** is required in a GAUSS program.

Branching

goto Unconditional branching.
if..endif Conditional branching.
pop Retrieve **goto** arguments.

```
if iter > itlim;
    goto errout("Iteration limit exceeded");
elseif iter == 1;
    j = setup(x,y);
else;
    j = iterate(x,y);
endif;

.
.
.
errout:
    pop errmsg;
    print errmsg;
end;
```

Looping

break Jump out the bottom of a **do** or **for** loop.
continue Jump to the top of a **do** or **for** loop.
do while..endo

```
                Loop if TRUE.
do until..endo   Loop if FALSE.
for.. endfor     Loop with integer counter.
    iter = 0;
    do while dif > tol;
        { x,x0 } = eval(x,x0);
        dif = abs(x-x0);
        iter = iter + 1;
        if iter > maxits;
            break;
        endif;
        if not prtiter;
            continue;
        endif;
        format /rdn 1,0;
        print "Iteration: " iter;;
        format /re 16,8;
        print ", Error:    "maxc(dif);
    endo;

    for i (1, cols(x), 1);
        for j (1, rows(x), 1);
            x[i,j] = x[i,j] + 1;
        endfor;
    endfor;
```

Subroutines

gosub Branch to subroutine.

pop	Retrieve gosub arguments.
return	Return from subroutine.

Arguments can be passed to subroutines in the branch to the subroutine label and then popped, in first-in-last-out order, immediately following the subroutine label definition. See Chapter 3, “Command Reference”, for details.

Arguments can then be returned in an analogous fashion through the **return** statement.

Procedures

endp	Terminates a procedure definition.
local	Declares variables local to a procedure.
proc	Begins definition of multi-line procedure.
retp	Returns from a procedure.

Here is an example of a GAUSS procedure:

```
proc (3) = crosprod(x,y);  
    local r1, r2, r3;  
    r1 = x[2,.*y[3,.-x[3,.*y[2,.;  
    r2 = x[3,.*y[1,.-x[1,.*y[3,.;  
    r3 = x[1,.*y[2,.-x[2,.*y[1,.;  
    retp( r1,r2,r3 );  
endp;
```

The “**(3) =**” indicates that the procedure returns three arguments. All local variables, except those listed in the argument list, must appear in the **local** statement. Procedures may reference global variables. There may be more than one **retp** per procedure definition; none is required if the procedure is defined to return 0 arguments. The **endp** is always necessary and must appear at the end of the procedure definition. Procedure definitions cannot be nested. The syntax for using this example function is

```
{ a1,a2,a3 } = crosprod(u,v);
```

See “Procedures and Keywords” and “Libraries” in the *User’s Guide* for details.

Libraries

call	Calls function and discard return values.
declare	Initializes variables at compile time.
external	External symbol definitions.
lib	Builds or updates a GAUSS library.
library	Sets up list of active libraries.

call allows functions to be called when return values are not needed. This is especially useful if a function produces printed output (**dstat**, **ols** for example) as well as return values.

Compiling

compile	Compiles and saves a program to a .gcg file.
loadp	Loads compiled procedure.
save	Saves the compiled image of a procedure to disk.
saveall	Saves the contents of the current workspace to a file.
use	Loads previously compiled code.

GAUSS procedures and programs may be compiled to disk files. By then using this compiled code, the time necessary to compile programs from scratch is eliminated. Use **compile** to compile a command file. All procedures, matrices and strings referenced by that program will be compiled as well.

Stand-alone applications may be created by running compiled code under the Run-Time Module. (Contact Aptech Systems for more information on this product.)

To save the compiled images of procedures that do not make any global references, use **save**. This will create an .fcg file. To load the compiled procedure into memory, use **loadp**. (This is not recommended because of the restriction on global references and the need to explicitly load the procedure in each program that references it. It is included here to maintain backward compatibility with previous versions.)

OS Functions

cdir	Returns current directory.
-------------	----------------------------

ChangeDir	Changes directory in program.
chdir	Changes directory interactively.
dfree	Returns free space on disk.
envget	Gets an environment string.
exec	Executes an executable program file.
fileinfo	Takes a file specification, returns names and information of files that match.
files	Takes a file specification, returns names of files that match.
filesa	Takes a file specification, returns names of files that match.
shell	Shells to OS.

Workspace Management

clear	Sets matrices equal to 0.
clearg	Sets global symbols to 0.
delete	Deletes specified global symbols.
hasimag	Examines matrix for nonzero imaginary part.
iscplx	Returns whether a matrix is real or complex.
maxvec	Returns maximum allowed vector size.
new	Clears current workspace.
show	Displays global symbol table.
type	Returns types of argument (matrix or string).
typecv	Returns types of symbol (argument contains the names of the symbols to be checked).

When working with limited workspace, it is a good idea to **clear** large matrices that are no longer needed by your program.

coreleft is most commonly used to determine how many rows of a data set may be read into memory at one time.

Error Handling and Debugging

#linesoff	Omits line number and file name records from program.
#lineson	Includes line number and file name records in program.
debug	Executes a program under the source level debugger.

disable	Disables invalid operation interrupt of coprocessor.
enable	Enables invalid operation interrupt of coprocessor.
error	Creates user-defined error code.
errorlog	Sends error message to screen and log file.
ndpchk	Examines status word of coprocessor.
ndpclex	Clears coprocessor exception flags.
ndpcntrl	Sets and gets coprocessor control word.
scalerr	Tests for a scalar error code.
trace	Traces program execution for debugging.
trap	Controls trapping of program errors.
trapchk	Examines the trap flag.

To trace the execution of a program, use **trace**.

User-defined error codes may be generated using **error**.

String Handling

chrs	Converts ASCII values to a string.
ftocv	Converts an NxK matrix to a character matrix.
ftos	Converts a floating point scalar to string.
ftostrC	Converts a matrix to a string array using a C language format specification.
getf	Loads ASCII or binary file into string.
loads	Loads a string file (.fst file).
lower	Converts a string to lowercase.
putf	Writes a string to disk file.
stof	Converts a string to floating point numbers.
strindx	Finds starting location of one string in another string.
strlen	Returns length of a string.
strrindx	Finds starting location of one string in another string, searching from the end to the start of the string.
strsect	Extracts a substring of a string.
strsplit	Splits an Nx1 string vector into an NxK string array of the individual tokens.
strsplitPad	Splits a string vector into a string array of the individual tokens. Pads on the right with null strings.

strtof	Converts a string array to a numeric matrix.
strtofcp1x	Converts a string array to a complex numeric matrix.
upper	Changes a string to uppercase.
vals	Converts a string to ASCII values.
varget	Accesses the global variable named by a string.
varget1	Accesses the local variable named by a string.
varput	Assigns a global variable named by a string.
varput1	Assigns a local variable named by a string.

strlen, **strindx**, **strrindx**, and **strsect** can be used together to parse strings.

Use **ftos** to print to a string.

To create a list of generic variable names (X1, X2, X3, X4... for example), use **ftocv**.

Time and Date Functions

date	Returns current system date.
datestr	Formats date as “mm/dd/yy”.
datestring	Formats date as “mm/dd/yyyy”.
datestrymd	Formats date as “yyyymmdd”.
dayinyr	Returns day number of a date.
dayofweek	Returns day of week.
dtdate	Creates a matrix in DT scalar format.
dtday	Creates a matrix in DT scalar format containing only the year, month and day. Time of day information is zeroed out.
dttime	Creates a matrix in DT scalar format containing only the hour, minute and second. The date information is zeroed out.
dttodtv	Converts DT scalar format to DTV vector format.
dttostr	Converts a matrix containing dates in DT scalar format to a string array.
dttoutc	Converts DT scalar format to UTC scalar format.
dtvnormal	Normalizes a date and time (DTV) vector.
dtvtodt	Converts DT vector format to DT scalar format.
etdays	Difference between two times in days.
ethsec	Difference between two times in 100ths of a second.

etstr	Converts elapsed time to string.
hsec	Returns elapsed time since midnight in 100ths of a second.
strtodt	Converts a string array of dates to a matrix in DT scalar format.
time	Returns current system time.
timedt	Returns system date and time in DT scalar format.
timestr	Formats time as “hh:mm:ss”.
todaydt	Returns system date in DT scalar format. The time returned is always midnight (00:00:00), the beginning of the returned day.
utctodt	Converts UTC scalar format to DT scalar format.
utctodtv	Converts UTC scalar format to DTV vector format.

Use **hsec** to time segments of code. For example,

```
et = hsec;  
x = y*y;  
et = hsec - et;
```

will time the GAUSS multiplication operator.

Console I/O

con	Requests console input, create matrix.
cons	Requests console input, create string.
key	Gets the next key from the keyboard buffer. If buffer is empty, returns a 0.
keyw	Gets the next key from the keyboard buffer. If buffer is empty, waits for a key.
wait	Waits for a keystroke.
waitc	Flushes buffer, then waits for a keystroke.

key can be used to trap most keystrokes. For example, the following loop will trap the ALT-H key combination:

```
kk = 0;  
do until kk == 1035;  
    kk = key;  
endo;
```

Other key combinations, function keys, and cursor key movement can also be trapped. See **key**.

cons and **con** can be used to request information from the console. **keyw**, **wait**, and **waitc** will wait for a keystroke.

Output Functions

Text Output

cls	Clears the window.
color	Sets pixel, text, background colors.
comlog	Controls interactive command logging.
csrcol	Gets column position of cursor on window.
csrlin	Gets row position of cursor on window.
ed	Accesses an alternate editor.
edit	Edits a file with the GAUSS editor.
format	Defines format of matrix printing.
locate	Positions the cursor on the window.
lpos	Returns print head position in printer buffer.
lprint	Prints expression to the printer.
lprint [[on off]]	Switches auto printer mode on and off.
lprwidth	Specifies printer width.
lshow	Prints global symbol table on the printer.
output	Redirects print statements to auxiliary output.
outwidth	Sets line width of auxiliary output.
plot	Plots elements of two matrices in text mode.
plotsym	Controls data symbol used by plot .
print	Prints to window.
print [[on off]]	Turns auto window print on and off.
printdos	Prints a string for special handling by the OS.
printfm	Prints matrices using a different format for each column.
screen [[on off]]	Directs/suppresses print statements to window.
screen out	Dumps snapshot of window to auxiliary output.
scroll	Scrolls a section of the window.

tab	Positions the cursor on the current line.
------------	---

The results of all printing can be sent to an output file using **output**. This file can then be printed or ported as an ASCII file to other software.

printdos can be used to print in reverse video, or using different colors. It requires that `ansi.sys` be installed.

To produce boxes, etc. using characters from the extended ASCII set, use **chrs**.

Window Graphics

color	Sets color.
graph	Sets pixels.
line	Draws lines.
setvmode	Sets video mode.

graph allows the user to plot individual pixels.

Graphics

This section summarizes all procedures and global variables available within the Publication Quality Graphics (PQG) System. A general usage description will be found in “Publications Quality Graphics” in the *User’s Guide*.

Graph Types

bar	Generates bar graph.
box	Graphs data using the box graph percentile method.
contour	Graphs contour data.
draw	Supplies additional graphic elements to graphs.
hist	Computes and graphs frequency histogram.
histf	Graphs a histogram given a vector of frequency counts.
histp	Graphs a percent frequency histogram of a vector.
loglog	Graphs X,Y using logarithmic X and Y axes.
logx	Graphs X,Y using logarithmic X axis.
logy	Graphs X,Y using logarithmic Y axis.
surface	Graphs a 3-D surface.
xy	Graphs X,Y using Cartesian coordinate system.

xyz Graphs X, Y, Z using 3-D Cartesian coordinate system.

Axes Control and Scaling

_pxpmax	Controls precision of numbers on X axis.
_paxes	Turns axes on or off.
_pcross	Controls where axes intersect.
_pgrid	Controls major and minor grid lines.
_pticout	Controls direction of tick marks on axes.
_pxsci	Controls use of scientific notation on X axis.
_pypmax	Controls precision of numbers on Y axis.
_pysci	Controls use of scientific notation on Y axis.
_pzpmax	Controls precision of numbers on Z axis.
_pzsci	Controls use of scientific notation on Z axis.
scale	Scales X,Y axes for 2-D plots.
scale3d	Scales X,Y, and Z axes for 3-D plots.
xtics	Scales X axis and control tick marks.
ytics	Scales Y axis and control tick marks.
ztics	Scales Z axis and control tick marks.

Text, Labels, Titles, and Fonts

_pnumht	Controls size of axes numeric labels.
_ptitlht	Controls main title size.
_paxht	Controls size of axes labels.
_pdate	Dates string contents and control.
_plegctl	Sets location and size of plot legend.
_plegstr	Specifies legend text entries.
_pmsgctl	Controls message position.
_pmsgstr	Specifies message text.
_pnum	Axes numeric label control and orientation.
asclabel	Defines character labels for tick marks.
fonts	Loads fonts for labels, titles, messages and legend.
title	Specifies main title for graph.
xlabel	X axis label.

ylabel	Y axis label.
zlabel	Z axis label.

Main Curve Lines and Symbols

_pboxctl	Controls box plotter.
_pboxlim	Outputs percentile matrix from box plotter.
_pcolor	Controls line color for main curves.
_plctrl	Controls main curve and frequency of data symbols.
_pltype	Controls line style for main curves.
_plwidth	Controls line thickness for main curves.
_pstype	Controls symbol type for main curves.
_psymsiz	Controls symbol size for main curves.
_pzclr	Z level color control for contour and surface .

Extra Lines and Symbols

_parrow	Creates arrows.
_parrow3	Creates arrows for 3-D graphs.
_perrbar	Plots error bars.
_pline	Plots extra lines and circles.
_pline3d	Plots extra lines for 3-D graphs.
_psym	Plots extra symbols.
_psym3d	Plots extra symbols for 3-D graphs.

Graphic Panel, Page, and Plot Control

_pageshf	Shifts the graph for printer output.
_pagesiz	Controls size of graph for printer output.
_plotshf	Controls plot area position.
_plotsiz	Controls plot area size.
_protate	Rotates the graph 90 degrees.
axmargin	Controls axes margins and plot size.
begwind	Graphic panel initialization procedure.
endwind	End graphic panel manipulation, display graphs.
getwind	Gets current graphic panel number.

loadwind	Loads a graphic panel configuration from a file.
makewind	Creates graphic panel with specified size and position.
margin	Controls graph margins.
nextwind	Sets to next available graphic panel number.
savewind	Saves graphic panel configuration to a file.
setwind	Sets to specified graphic panel number.
window	Creates tiled graphic panels of equal size.

axmargin is preferred to the older **_plotsiz** and **_plotshf** globals for establishing an absolute plot size and position.

Output Options

_pscreen	Controls graphics output to window.
_psilent	Controls final beep.
_ptek	Controls creation and name of <code>graphics.tkf</code> file.
_pzoom	Specifies zoom parameters.
graphprt	Generates print, conversion file.
pqgwin	Sets the graphics viewer mode.
setvwrmode	Sets the graphics viewer mode.
tkf2eps	Converts .tkf file to Encapsulated PostScript file.
tkf2ps	Converts .tkf file to PostScript file.

Miscellaneous

_pbox	Draws a border around graphic panel/window.
_pcrop	Controls cropping of graphics data outside axes area.
_pframe	Draws a frame around 2-D, 3-D plots.
_pmcolor	Controls colors to be used for axes, title, x and y labels, date, box, and background.
graphset	Resets all PQG globals to default values.
rerun	Displays most recently created graph.
view	Sets 3-D observer position in workbox units.
viewxyz	Sets 3-D observer position in plot coordinates.
volume	Sets length, width, and height ratios of 3-D workbox.

Command Reference 3

Command Components

The following list describes each of the components used in the GAUSS Language Command Reference.

Purpose	Describes the function of the command.
Library	Describes where the Run-Time Library functions can be found in the GAUSS program structure.
Format	Illustrates use of the command syntax.
Input	Describes the input parameters of the command.
Global Input	Describes the global variables that are referenced by the command.
Output	Describes the parameters that will be returned as a result of the input.

Command Reference

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

Global Output	Describes the global variables that are updated by the command.
Portability	Describes how GAUSS behaves under various operating systems.
Remarks	Explanatory material pertinent to the command.
Example	Sample code strings using the command.
Source	The source file in which Run-Time Library function is defined.
Globals	Global variables that are accessed by the command.
See also	Other commands that have similar characteristics.
Technical Notes	Technical discussion and reference source citations.
References	Reference material citations.

abs

Purpose Returns the absolute value or complex modulus of x .

Format $y = \text{abs}(x);$

Input x NxK matrix.

Output y NxK matrix containing absolute values of x .

Example $x = \text{rndn}(2,2);$
 $y = \text{abs}(x);$

$x = \begin{bmatrix} 0.675243 & 1.053485 \\ -0.190746 & -1.229539 \end{bmatrix}$

$y = \begin{bmatrix} 0.675243 & 1.053485 \\ 0.190746 & 1.229539 \end{bmatrix}$

In this example, a 2x2 matrix of Normal random numbers is generated and the absolute value of the matrix is computed.

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

acf

acf

a	Purpose	Computes sample autocorrelations.
b	Format	$rk = \mathbf{acf}(y,k,d);$
c	Input	y Nx1 vector, data. k scalar, maximum number of autocorrelations to compute. d scalar, order of differencing.
d	Output	rk Kx1 vector, sample autocorrelations.
e	Example	$x = \{ 20.80,$ 18.58, 23.39, 20.47, 21.78, 19.56, 19.58, 18.91, 20.08, 21.88 $\};$ $rk = \mathbf{acf}(x,4,2);$ $\mathbf{print\ rk};$ -0.74911771 0.48360914 -0.34229330 0.17461180
f	Source	tsutil.src
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

AmericanBinomCall

Purpose American binomial method Call.

Format `c = AmericanBinomCall(S0,K,r,div,tau,sigma,N);`

Input

<i>S0</i>	scalar, current price
<i>K</i>	Mx1 vector, strike prices
<i>r</i>	scalar, risk free rate
<i>div</i>	continuous dividend yield
<i>tau</i>	scalar, elapsed time to exercise in annualized days of trading
<i>sigma</i>	scalar, volatility
<i>N</i>	number of time segments

Output *c* Mx1 vector, call premiums

Example

```

S0 = 718.46;
K = { 720, 725, 730 };
r = .0498;
sigma = .2493;
t0 = dtday(2001, 1, 30);
t1 = dtday(2001, 2, 16);
tau = elapsedTradingDays(t0,t1) /
      annualTradingDays(2001);
c = AmericanBinomCall(S0,K,r,0,tau,sigma,60);
print c;

17.190224
14.905054
12.673322

```

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

AmericanBinomCall

Source `finprocs.src`

Technical Notes The binomial method of Cox, Ross, and Rubinstein ("Option pricing: a simplified approach", Journal of Financial Economics, 7:229:264) as described in Options, Futures, and other Derivatives by John C. Hull is the basis of this procedure.

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

AmericanBinomCall_Greeks

Purpose American binomial method call Delta, Gamma, Theta, Vega, and Rho.

Format `{ d,g,t,v,rh } = AmericanBinomCall_Greeks(S0,K,r,div,tau,sigma,N);`

Input

<i>S0</i>	scalar, current price
<i>K</i>	Mx1 vector, strike price
<i>r</i>	scalar, risk free rate
<i>div</i>	continuous dividend yield
<i>tau</i>	scalar, elapsed time to exercise in annualized days of trading
<i>sigma</i>	scalar, volatility
<i>N</i>	number of time segments

Output

<i>d</i>	Mx1 vector, delta
<i>g</i>	Mx1 vector, gamma
<i>t</i>	Mx1 vector, theta
<i>v</i>	Mx1 vector, vega
<i>rh</i>	Mx1 vector, rho

Example

```

S0 = 305;
K = 300;
r = .08;
sigma = .25;
tau = .33;
div = 0;
{ d,g,t,v,rh } = AmericanBinomCall_Greeks
    (S0,K,r,div,tau,sigma,30);
print d;
print g;
print t;

```

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

AmericanBinomCall_Greeks

		<pre>print v; print rh;</pre>
a		
b		0.706312
c		0.000764
d		-17.400851
e		68.703849
f		76.691829
g	Source	finprocs.src
h	Globals	_fin_thetaType scalar, if 1, one day look ahead, else, infinitesimal. Default = 0.
i		_fin_epsilon scalar, finite difference stepsize. Default = 1e-8.
j		
k	Technical Notes	The binomial method of Cox, Ross, and Rubinstein ("Option pricing: a simplified approach", Journal of Financial Economics, 7:229:264) as described in Options, Futures, and other Derivatives by John C. Hull is the basis of this procedure.
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

AmericanBinomCall_ImpVol

Purpose	Implied volatilities for American binomial method calls.	
Format	$\sigma =$ AmericanBinomCall_ImpVol (<i>c</i> , <i>S0</i> , <i>K</i> , <i>r</i> , <i>div</i> , <i>tau</i> , <i>N</i>);	
Input	<i>c</i>	Mx1 vector, call premiums
	<i>S0</i>	scalar, current price
	<i>K</i>	Mx1 vector, strike prices
	<i>r</i>	scalar, risk free rate
	<i>div</i>	continuous dividend yield
	<i>tau</i>	scalar, elapsed time to exercise in annualized days of trading
	<i>N</i>	number of time segments
Output	σ	Mx1 vector, volatility
Example	<i>c</i> = { 13.70, 11.90, 9.10 };	
	<i>S0</i> = 718.46;	
	<i>K</i> = { 720, 725, 730 };	
	<i>r</i> = .0498;	
	<i>div</i> = 0;	
	<i>t0</i> = dtday(2001, 1, 30);	
	<i>t1</i> = dtday(2001, 2, 16);	
	<i>tau</i> = elapsedTradingDays(<i>t0</i> , <i>t1</i>) /	
	annualTradingDays(2001);	
	σ = AmericanBinomCall_ImpVol	
	(<i>c</i> , <i>S0</i> , <i>K</i> , <i>r</i> , <i>div</i> , <i>tau</i> ,30);	
	print σ ;	
	0.1981	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

AmericanBinomCall_ImpVol

0.1715

0.1301

a		
b		
c		
d		
e		
f		
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

Source finprocs.src

Technical Notes The binomial method of Cox, Ross, and Rubinstein ("Option pricing: a simplified approach", Journal of Financial Economics, 7:229:264) as described in Options, Futures, and other Derivatives by John C. Hull is the basis of this procedure.

AmericanBinomPut

Purpose American binomial method Put.

Format `c = AmericanBinomPut(S0,K,r,div,tau,sigma,N);`

Input

<i>S0</i>	scalar, current price
<i>K</i>	Mx1 vector, strike prices
<i>r</i>	scalar, risk free rate
<i>div</i>	continuous dividend yield
<i>tau</i>	scalar, elapsed time to exercise in annualized days of trading
<i>sigma</i>	scalar, volatility
<i>N</i>	number of time segments

Output *c* Mx1 vector, put premiums

Example:

```

S0 = 718.46;
K = { 720, 725, 730 };
r = .0498;
sigma = .2493;
t0 = dtday(2001, 1, 30);
t1 = dtday(2001, 2, 16);
tau = elapsedTradingDays(t0,t1) /
      annualTradingDays(2001);
c = AmericanBinomPut(S0,K,r,0,tau,sigma,60);
print c;

16.862683
19.606573
22.433590

```

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

AmericanBinomPut

Source `finprocs.src`

Technical Notes The binomial method of Cox, Ross, and Rubinstein ("Option pricing: a simplified approach", Journal of Financial Economics, 7:229:264) as described in Options, Futures, and other Derivatives by John C. Hull is the basis of this procedure.

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

AmericanBinomPut_Greeks

Purpose American binomial method put Delta, Gamma, Theta, Vega, and Rho.

Format `{ d,g,t,v,rh } = AmericanBinomPut_Greeks(S0,K,r,div,tau,sigma,N);`

Input

<i>S0</i>	scalar, current price
<i>K</i>	Mx1 vector, strike price
<i>r</i>	scalar, risk free rate
<i>div</i>	continuous dividend yield
<i>tau</i>	scalar, elapsed time to exercise in annualized days of trading
<i>sigma</i>	scalar, volatility
<i>N</i>	number of time segments

Output

<i>d</i>	Mx1 vector, delta
<i>g</i>	Mx1 vector, gamma
<i>t</i>	Mx1 vector, theta
<i>v</i>	Mx1 vector, vega
<i>rh</i>	Mx1 vector, rho

Example

```

S0 = 305;
K = 300;
r = .08;
div = 0;
sigma = .25;
tau = .33;
{ d,g,t,v,rh } = AmericanBinomPut_Greeks
    (S0,K,r,div,tau,sigma,60);
print d;
print g;
print t;

```

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

AmericanBinomPut_Greeks

		<pre>print v; print rh;</pre>
a		
b		-0.38324908
c		0.00076381912
d		8.1336630
e		68.337294
f		-27.585043
g	Source	finprocs.src
h	Globals	_fin_thetaType scalar, if 1, one day look ahead, else, infinitesimal. Default = 0.
i		_fin_epsilon scalar, finite difference stepsize. Default = 1e-8.
j		
k	Technical Notes	The binomial method of Cox, Ross, and Rubinstein ("Option pricing: a simplified approach", Journal of Financial Economics, 7:229:264) as described in Options, Futures, and other Derivatives by John C. Hull is the basis of this procedure.
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

AmericanBinomPut_ImpVol

Purpose	Implied volatilities for American binomial method puts.	
Format	<i>sigma</i> = AmericanBinomPut_ImpVol (<i>c</i> , <i>S0</i> , <i>K</i> , <i>r</i> , <i>div</i> , <i>tau</i> , <i>N</i>);	
Input	<i>c</i>	Mx1 vector, put premiums
	<i>S0</i>	scalar, current price
	<i>K</i>	Mx1 vector, strike prices
	<i>r</i>	scalar, risk free rate
	<i>div</i>	continuous dividend yield
	<i>tau</i>	scalar, elapsed time to exercise in annualized days of trading
	<i>N</i>	number of time segments
Output	<i>sigma</i>	Mx1 vector, volatility
Example	<pre>p = { 14.60, 17.10, 20.10 };</pre>	
	<pre>S0 = 718.46;</pre>	
	<pre>K = { 720, 725, 730 };</pre>	
	<pre>r = .0498;</pre>	
	<pre>div = 0;</pre>	
	<pre>t0 = dtday(2001, 1, 30);</pre>	
	<pre>t1 = dtday(2001, 2, 16);</pre>	
	<pre>tau = elapsedTradingDays(t0,t1) /</pre>	
	<pre> annualTradingDays(2001);</pre>	
	<pre>sigma = AmericanBinomPut_ImpVol</pre>	
	<pre> (p,S0,K,r,div,tau,30);</pre>	
	<pre>print sigma;</pre>	
	 0.1254	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

AmericanBinomPut_ImpVol

0.1668
0.2134

Source finprocs.src

Technical Notes The binomial method of Cox, Ross, and Rubinstein ("Option pricing: a simplified approach", Journal of Financial Economics, 7:229:264) as described in Options, Futures, and other Derivatives by John C. Hull is the basis of this procedure.

AmericanBSCall

Purpose American Black and Scholes Call.

Format `c = AmericanBSCall(S0,K,r,div,tau,sigma);`

Input

<i>S0</i>	scalar, current price
<i>K</i>	Mx1 vector, strike prices
<i>r</i>	scalar, risk free rate
<i>div</i>	continuous dividend yield
<i>tau</i>	scalar, elapsed time to exercise in annualized days of trading
<i>sigma</i>	scalar, volatility

Output: *c* Mx1 vector, call premiums

Example

```

S0 = 718.46;
K = { 720, 725, 730 };
r = .0498;
sigma = .2493;
t0 = dtday(2001, 1, 30);
t1 = dtday(2001, 2, 16);
tau = elapsedTradingDays(t0,t1) /
      annualTradingDays(2001);
c = AmericanBSCall(S0,K,r,0,tau,sigma);
print c;

16.093640
13.846830
11.829059

```

Source finprocs.src

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

AmericanBSCall_Greeks

AmericanBSCall_Greeks

a		
b	Purpose	American Black and Scholes call Delta, Gamma, Omega, Theta, and Vega.
c	Format	$\{ d,g,t,v,rh \} =$ <code>AmericanBSCall_Greeks(S0,K,r,div,tau,sigma);</code>
d		
e	Input	<i>S0</i> scalar, current price
f		<i>K</i> Mx1 vector, strike price
g		<i>r</i> scalar, risk free rate
h		<i>div</i> continuous dividend yield
i		<i>tau</i> scalar, elapsed time to exercise in annualized days of trading
j		<i>sigma</i> scalar, volatility
k	Output	<i>d</i> Mx1 vector, delta
l		<i>g</i> Mx1 vector, gamma
m		<i>t</i> Mx1 vector, theta
n		<i>v</i> Mx1 vector, vega
o		<i>rh</i> Mx1 vector, rho
p	Example	<code>S0 = 305;</code>
q		<code>K = 300;</code>
r		<code>r = .08;</code>
s		<code>sigma = .25;</code>
t		<code>tau = .33;</code>
u		<code>{ d,g,t,v,rh } = AmericanBSCall_Greeks</code>
v		<code>(S0,K,r,0,tau,sigma);</code>
w		<code>print d;</code>
x		<code>print g;</code>
y		<code>print t;</code>
z		<code>print v;</code>

AmericanBSCall_Greeks

	print rh;
	0.40034039
	0.016804021
	-55.731079
	115.36906
	46.374528
Source	finprocs.src
Globals	
	_fin_thetaType scalar, if 1, one day look ahead, else, infinitesimal. Default = 0.
	_fin_epsilon scalar, finite difference stepsize. Default = 1e-8.

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

AmericanBSCall_ImpVol

AmericanBSCall_ImpVol

a	Purpose	Implied volatilities for American Black and Scholes calls.
b	Format	<i>sigma</i> = AmericanBSCall_ImpVol (<i>c</i> , <i>S0</i> , <i>K</i> , <i>r</i> , <i>div</i> , <i>tau</i>);
c	Input	<i>c</i> Mx1 vector, call premiums
d		<i>S0</i> scalar, current price
e		<i>K</i> Mx1 vector, strike prices
f		<i>r</i> scalar, risk free rate
g		<i>div</i> continuous dividend yield
h		<i>tau</i> scalar, elapsed time to exercise in annualized days of trading
i	Output	<i>sigma</i> Mx1 vector, volatility
j	Example	<i>c</i> = { 13.70, 11.90, 9.10 };
k		<i>S0</i> = 718.46;
l		<i>K</i> = { 720, 725, 730 };
m		<i>r</i> = .0498;
n		<i>t0</i> = dtday(2001, 1, 30);
o		<i>t1</i> = dtday(2001, 2, 16);
p		<i>tau</i> = elapsedTradingDays(<i>t0</i> , <i>t1</i>) /
q		annualTradingDays(2001);
r		<i>sigma</i> = AmericanBSCall_ImpVol
s		(<i>c</i> , <i>S0</i> , <i>K</i> , <i>r</i> ,0, <i>tau</i>);
t		print <i>sigma</i> ;
u		
v		0.10350708
w		0.089202881
x y z		0.066876221

AmericanBSCall_ImpVol

Source finprocs.src

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

AmericanBSPut

AmericanBSPut

Purpose	American Black and Scholes Put.	
Format	$c = \text{AmericanBSPut}(S0, K, r, div, tau, sigma);$	
Input	$S0$	scalar, current price
	K	Mx1 vector, strike prices
	r	scalar, risk free rate
	div	continuous dividend yield
	tau	scalar, elapsed time to exercise in annualized days of trading
	$sigma$	scalar, volatility
Output	c	Mx1 vector, put premiums
Example	$S0 = 718.46;$	
	$K = \{ 720, 725, 730 \};$	
	$r = .0498;$	
	$sigma = .2493;$	
	$t0 = \text{dtday}(2001, 1, 30);$	
	$t1 = \text{dtday}(2001, 2, 16);$	
	$tau = \text{elapsedTradingDays}(t0, t1) /$	
	$\quad \text{annualTradingDays}(2001);$	
	$c = \text{AmericanBSPut}(S0, K, r, 0, tau, sigma);$	
	$\text{print } c;$	
	16.748987	
	19.41627	
	22.318856	
Source	finprocs.src	

AmericanBSPut_Greeks

Purpose American Black and Scholes put Delta, Gamma, Omega, Theta, and Vega.

Format `{ d,g,t,v,rh } = AmericanBSPut_Greeks(S0,K,r,div,tau,sigma);`

Input

<i>S0</i>	scalar, current price
<i>K</i>	Mx1 vector, strike price
<i>r</i>	scalar, risk free rate
<i>div</i>	continuous dividend yield
<i>tau</i>	scalar, elapsed time to exercise in annualized days of trading
<i>sigma</i>	scalar, volatility

Output

<i>d</i>	Mx1 vector, delta
<i>g</i>	Mx1 vector, gamma
<i>t</i>	Mx1 vector, theta
<i>v</i>	Mx1 vector, vega
<i>rh</i>	Mx1 vector, rho

Example

```
S0 = 305;
K = 300;
r = .08;
sigma = .25;
tau = .33;
{ d,g,t,v,rh } =
    AmericanBSPut_Greeks(S0,K,r,0,tau,sigma);
print d;
print g;
print t;
print v;
```

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

AmericanBSPut_Greeks

		print rh;
		-0.33296721
a		0.0091658294
b		-17.556118
c		77.614238
d		-40.575963
e	Source	finprocs.src
f	Globals	
g		_fin_thetaType scalar, if 1, one day look ahead, else, infinitesimal. Default = 0.
h		_fin_epsilon scalar, finite difference stepsize. Default = 1e-8.
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

AmericanBSPut_ImpVol

Purpose	Implied volatilities for American Black and Scholes puts.	
Format	<i>sigma</i> = AmericanBSPut_ImpVol (<i>c</i> , <i>S0</i> , <i>K</i> , <i>r</i> , <i>div</i> , <i>tau</i>);	
Input	<i>c</i>	Mx1 vector, put premiums
	<i>S0</i>	scalar, current price
	<i>K</i>	Mx1 vector, strike prices
	<i>r</i>	scalar, risk free rate
	<i>div</i>	continuous dividend yield
	<i>tau</i>	scalar, elapsed time to exercise in annualized days of trading
Output	<i>sigma</i>	Mx1 vector, volatility
Example	<i>p</i> = { 14.60, 17.10, 20.10 };	
	<i>S0</i> = 718.46;	
	<i>K</i> = { 720, 725, 730 };	
	<i>r</i> = .0498;	
	<i>t0</i> = dtday(2001, 1, 30);	
	<i>t1</i> = dtday(2001, 2, 16);	
	<i>tau</i> = elapsedTradingDays(<i>t0</i> , <i>t1</i>) /	
	annualTradingDays(2001);	
	<i>sigma</i> = AmericanBSPut_ImpVol(<i>p</i> , <i>S0</i> , <i>K</i> , <i>r</i> ,0, <i>tau</i>);	
	print <i>sigma</i> ;	
	0.12829346	
	0.16885986	
	0.21544312	
Source	finprocs.src	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

annualTradingDays

annualTradingDays

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

Purpose	Compute number of trading days in a given year.
Format	n = annualTradingDays (a);
Input	a scalar, year.
Output	n number of trading days in year
Remarks	A trading day is a weekday that is not a holiday as defined by the New York Stock Exchange from 1888 through 2004. Holidays are defined in <code>holidays.asc</code> . You may edit that file to modify or add holidays.
Source	<code>finutils.src</code>

arccos

Purpose	Computes the inverse cosine.		
Format	$y = \text{arccos}(x);$		
Input	x	NxK matrix.	
Output	y	NxK matrix containing the angle in radians whose cosine is x .	
Remarks	If x is complex or has any elements whose absolute value is greater than 1, complex results are returned.		
Example	$x = \{ -1, -0.5, 0, 0.5, 1 \};$ $y = \text{arccos}(x);$ -1.000000 -0.500000 $x =$ 0.000000 0.500000 1.000000 3.141593 2.094395 $y =$ 1.570796 1.047198 0.000000		
Source	trig.src		

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

arcsin

arcsin

a	Purpose	Computes the inverse sine.
b	Format	$y = \text{arcsin}(x);$
c	Input	x NxK matrix.
d	Output	y NxK matrix, the angle in radians whose sine is x .
e	Remarks	If x is complex or has any elements whose absolute value is greater than 1, complex results are returned.
f	Example	$x = \{ -1, -0.5, 0, 0.5, 1 \};$
g		$y = \text{arcsin}(x);$
h		-1.000000
i		-0.500000
j		$x = 0.000000$
k		0.500000
l		1.000000
m		-1.570796
n		-0.523599
o		$y = 0.000000$
p		0.523599
q		1.570796
r	Source	trig.src

asclabel

Purpose Sets up character labels for the X and Y axes.

Library **pgraph**

Format **asclabel**(*xl,yl*);

Input

<i>xl</i>	string or Nx1 character vector, labels for the tick marks on the X axis. Set to 0 if no character labels for this axis are desired.
<i>yl</i>	string or Mx1 character vector, labels for the tick marks on the Y axis. Set to 0 if no character labels for this axis are desired.

Example This illustrates how to label the X axis with the months of the year:

```
let lab = JAN FEB MAR APR MAY JUN JUL AUG SEP
OCT NOV DEC;
asclabel(lab,0);
```

This will also work:

```
lab = "JAN FEB MAR APR MAY JUN JUL AUG SEP OCT
NOV DEC";
asclabel(lab,0);
```

If the string format is used, then escape characters may be embedded in the labels. For example, the following produces character labels that are multiples of λ . The font Simgrma must be previously loaded in a **fonts** command. (See Chapter 13 in *Using GAUSS for Windows95*.)

```
fonts("simplex simgrma");
lab = "\2010.25\2021 \2010.5\2021
\2010.75\2021 1";
asclabel(lab,0);
```

Here, the **2021** produces the “ λ ” symbol from Simgrma.

Source pgraph.src

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

asclabel

See also `xtics`, `ytics`, `scale`, `scale3d`, `fonts`

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

atan

Purpose	Returns the arctangent of its argument.
Format	$y = \text{atan}(x);$
Input	x NxK matrix.
Output	y NxK matrix containing the arctangent of x in radians.
Remarks	<p>y will be a matrix the same size as x, containing the arctangents of the corresponding elements of x.</p> <p>For real x, the arctangent of x is the angle whose tangent is x. The result is a value in radians in the range $\frac{-\pi}{2}$ to $\frac{+\pi}{2}$. To convert radians to degrees, multiply by $\frac{180}{\pi}$.</p> <p>For complex x, the arctangent is defined everywhere except i and $-i$. If x is complex, y will be complex.</p>
Example	<pre>x = { 2, 4, 6, 8 }; z = x/2; y = atan(z); 0.785398 y = 1.107149 1.249046 1.325818</pre>
See also	atan2, sin, cos, pi, tan

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

atan2

atan2

a		
b	Purpose	Computes an angle from an x,y coordinate.
c	Format	$z = \text{atan2}(y,x);$
d	Input	y NxK matrix, the Y coordinate. x LxM matrix, ExE conformable with y , the X coordinate.
e		
f	Output	z $\max(N,L)$ by $\max(K,M)$ matrix.
g	Remarks	Given a point x,y in a Cartesian coordinate system, atan2 will give the correct angle with respect to the positive X axis. The answer will be in radians from $-\pi$ to $+\pi$.
h		
i		To convert radians to degrees, multiply by $\frac{180}{\pi}$.
j		
k		atan2 operates only on the real component of x , even if x is complex.
l	Example	$x = 2;$ $y = \{ 2, 4, 6, 8 \};$ $z = \text{atan2}(y,x);$ $z =$ 0.785398 1.107149 1.249046 1.325818
m		
n		
o		
p		
q		
r		
s	See also	atan, sin, cos, pi, tan, arcsin, arccos
t		
u		
v		
w		
x y z		

axmargin

Purpose	Set absolute margins for the plot axes which control placement and size of plot.
Library	pgraph
Format	axmargin (<i>l,r,t,b</i>);
Input	<i>l</i> scalar, the left margin in inches. <i>r</i> scalar, the right margin in inches. <i>t</i> scalar, the top margin in inches. <i>b</i> scalar, the bottom margin in inches.
Remarks	<p>axmargin sets an absolute distance from the axes to the edge of the graphic panel. Note that the user is responsible for allowing enough space in the margin if axes labels, numbers, and title are used on the graph, since axmargin does not size the plot automatically as in the case of margin.</p> <p>All input inch values for this procedure are based on a full size window of 9 x 6.855 inches. If this procedure is used within a graphic panel, the values will be scaled to window inches automatically.</p> <p>If both margin and axmargin are used for a graph, axmargin will override any sizes specified by margin.</p>
Example	<p>The statement</p> <pre>axmargin(1,1,.5,.855);</pre> <p>will create a plot area of 7 inches horizontally by 5.5 inches vertically, and positioned 1 inch right and .855 up from the lower left corner of the graphic panel/page.</p>
Source	pgraph.src

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

balance**balance**

Purpose Balances a square matrix.

Format $\{ b, z \} = \text{balance}(x)$

Input x KxK matrix.

Output b KxK matrix, balanced matrix.

z KxK matrix, diagonal scale matrix.

Remarks **balance** returns a balanced matrix b and another matrix z with scale factors in powers of two on its diagonal. b is balanced in the sense that the absolute sums of the magnitudes of elements in corresponding rows and columns are nearly equal.

balance is most often used to scale matrices to improve the numerical stability of the calculation of their eigenvalues. It is also useful in the solution of matrix equations.

In particular,

$$b = z^{-1}xz$$

balance uses the BALANC function from EISPACK.

Example `let x[3,3] = 100 200 300`
 `40 50 60`
 `7 8 9 ;`

`{ b,z } = balance(x);`

`100.0 100.0 37.5`
`= 80.0 50.0 15.0`
 `56.0 32.0 9.0`

`4.0 0.0 0.0`
`= 0.0 2.0 0.0`
 `0.0 0.0 0.5`

band

Purpose	Extracts bands from a symmetric banded matrix.		
Format	$a = \mathbf{band}(y,n);$		
Input	y	KxK symmetric banded matrix.	
	n	scalar, number of subdiagonals.	
Output	a	Kx(N+1) matrix, 1 subdiagonal per column.	
Remarks	y can actually be a rectangular PxQ matrix. K is then defined as min(P,Q). It will be assumed that a is symmetric about the principal diagonal for $y[1:K,1:K]$.		
	The subdiagonals of y are stored right to left in a , with the principal diagonal in the rightmost (N+1'th) column of a . The upper left corner of a is unused; it is set to 0.		
	This compact form of a banded matrix is what bandchol expects.		
Example	$\mathbf{x} = \left\{ \begin{array}{cccc} 1 & 2 & 0 & 0, \\ & 2 & 8 & 1, \\ & & 0 & 1, \\ & & & 5, \\ & & & & 2, \\ & & & & & 0, \\ & & & & & & 0, \\ & & & & & & & 2, \\ & & & & & & & & 3 \end{array} \right\};$		
	$\mathbf{bx} = \mathbf{band}(\mathbf{x},1);$ <div><div>$\mathbf{bx} = \begin{bmatrix} 0.0000000 & 1.0000000 \\ 2.0000000 & 8.0000000 \\ 1.0000000 & 5.0000000 \\ 2.0000000 & 3.0000000 \end{bmatrix}$</div></div>		

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

bandchol

bandchol

a		
b	Purpose	Computes the Cholesky decomposition of a positive definite banded matrix.
c	Format	$l = \text{bandchol}(a);$
d	Input	a KxN compact form matrix.
e	Output	l KxN compact form matrix, lower triangle of the Cholesky decomposition of a .
f		
g	Remarks	Given a positive definite banded matrix A , there exists a matrix L , the lower triangle of the Cholesky decomposition of A , such that $A = L \times L'$. a is the compact form of A . See band for a description of the format of a . l is the compact form of L . This is the form of matrix that bandcholsol expects.
h		
i	Example	$x = \left\{ \begin{array}{cccc} 1 & 2 & 0 & 0, \\ & 2 & 8 & 1 & 0, \\ & & 0 & 1 & 5 & 2, \\ & & & 0 & 0 & 2 & 3 \end{array} \right\};$ $bx = \text{band}(x, 1);$ $bx = \begin{array}{cc} 0.0000000 & 1.0000000 \\ 2.0000000 & 8.0000000 \\ 1.0000000 & 5.0000000 \\ 2.0000000 & 3.0000000 \end{array}$ $cx = \text{bandchol}(bx);$ $cx = \begin{array}{cc} 0.0000000 & 1.0000000 \\ 2.0000000 & 2.0000000 \\ 0.50000000 & 2.1794495 \\ 0.91766294 & 1.4689774 \end{array}$
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

bandcholsol

Purpose Solves the system of equations $Ax = b$ for x , given the lower triangle of the Cholesky decomposition of a positive definite banded matrix A .

Format $x = \text{bandcholsol}(b,l);$

Input b KxM matrix.
 l KxN compact form matrix.

Output x KxM matrix.

Remarks Given a positive definite banded matrix A , there exists a matrix L , the lower triangle of the Cholesky decomposition of A , such that $A = L*L'$. l is the compact form of L ; see **band** for a description of the format of l .
 b can have more than one column. If so, $Ax = b$ is solved for each column. That is,

$$A*x[:,i] = b[:,i]$$

Example $x = \begin{Bmatrix} 1 & 2 & 0 & 0, \\ 2 & 8 & 1 & 0, \\ 0 & 1 & 5 & 2, \\ 0 & 0 & 2 & 3 \end{Bmatrix};$
 $bx = \text{band}(x,1);$

 $bx = \begin{bmatrix} 0.0000000 & 1.0000000 \\ 2.0000000 & 8.0000000 \\ 1.0000000 & 5.0000000 \\ 2.0000000 & 3.0000000 \end{bmatrix}$

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

bandcholsol

```
cx = bandchol(bx);
```

```
cx =
    0.0000000    1.0000000
    2.0000000    2.0000000
    0.50000000   2.1794495
    0.91766294   1.4689774
```

```
xi = bandcholsol(eye(4),cx);
```

```
xi =
    2.0731707  -0.05365854  0.14634146   0.09756098
   -0.53658537  0.26829268 -0.07317073   0.04878049
    0.14634146 -0.07317073  0.29268293  -0.19512195
   -0.09756098  0.04878049 -0.19512195   0.46341463
```

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

bandltsol

Purpose Solves the system of equations $Ax = b$ for x , where A is a lower triangular banded matrix.

Format $x = \text{bandltsol}(b,A);$

Input b KxM matrix.
 A KxN compact form matrix.

Output x KxM matrix.

Remarks A is a lower triangular banded matrix in compact form. See **band** for a description of the format of A .
 b can have more than one column. If so, $Ax = b$ is solved for each column. That is,

$$A * x[:, i] = b[:, i]$$

Example

$$bx = \begin{bmatrix} 0.0000000 & 1.0000000 \\ 2.0000000 & 8.0000000 \\ 1.0000000 & 5.0000000 \\ 2.0000000 & 3.0000000 \end{bmatrix}$$

$cx = \text{bandchol}(bx);$

$$cx = \begin{bmatrix} 0.0000000 & 1.0000000 \\ 2.0000000 & 2.0000000 \\ 0.5000000 & 2.1794495 \\ 0.91766294 & 1.4689774 \end{bmatrix}$$

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

bandltsol

```
xc1 = bandltsol(eye(4),cx);
```

$xc1 =$

1.0000000	0.0000000	0.0000000	0.0000000
-1.0000000	0.5000000	0.0000000	0.0000000
0.22941573	-0.11470787	0.45883147	0.0000000
-0.14331487	0.07165744	-0.28662975	0.68074565

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

bandrv

Purpose	Creates a symmetric banded matrix, given its compact form.		
Format	$y = \text{bandrv}(a);$		
Input	a	KxN compact form matrix.	
Output	y	KxK symmetrix banded matrix.	
Remarks	a is the compact form of a symmetric banded matrix, as generated by band . a stores subdiagonals right to left, with the principal diagonal in the rightmost (N^{th}) column. The upper left corner of a is unused. bandchol expects a matrix of this form. y is the fully expanded form of a , a KxK matrix with N-1 subdiagonals.		

Example		0.0000000	1.0000000		
	$bx =$	2.0000000	8.0000000		
		1.0000000	5.0000000		
		2.0000000	3.0000000		
	$x = \text{bandrv}(bx);$				
x		1.0000000	2.0000000	0.0000000	0.0000000
	$=$	2.0000000	8.0000000	1.0000000	0.0000000
		0.0000000	1.0000000	5.0000000	2.0000000
		0.0000000	0.0000000	2.0000000	3.0000000

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

bandsolpd

bandsolpd

Purpose	Solves the system of equations $Ax = b$ for x , where A is a positive definite banded matrix.		
Format	$x = \text{bandsolpd}(b,A);$		
Input	b	KxM matrix.	
	A	KxN compact form matrix.	
Output	x	KxM matrix.	
Remarks	A is a positive definite banded matrix in compact form. See band for a description of the format of A .		
	b can have more than one column. If so, $Ax = b$ is solved for each column. That is, $A*x[. , i] = b[. , i]$		

bar

Purpose	Bar graph.	
Library	pgraph	
Format	bar (<i>val</i> , <i>ht</i>) ;	
Input	<i>val</i>	Nx1 numeric vector, bar labels. If scalar 0, a sequence from 1 to rows (<i>ht</i>) will be created.
	<i>ht</i>	<p>NxK numeric vector, bar heights.</p> <p>K overlapping or side-by-side sets of N bars will be graphed.</p> <p>For overlapping bars, the first column should contain the set of bars with the greatest height and the last column should contain the set of bars with the least height. Otherwise, the bars that are drawn first may be obscured by the bars drawn last. This is not a problem if the bars are plotted side-by-side.</p>
Global Input	_pbarwid	<p>global scalar, width and positioning of bars in bar graphs and histograms. The valid range is 0-1. If this is 0, the bars will be a single pixel wide. If this is 1, the bars will touch each other.</p> <p>If this value is positive, the bars will overlap. If negative, the bars will be plotted side-by-side. The default is 0.5.</p>
	_pbartyp	<p>Kx2 matrix.</p> <p>The first column controls the bar shading:</p> <ul style="list-style-type: none"> 0 no shading. 1 dots. 2 vertical cross-hatch. 3 diagonal lines with positive slope. 4 diagonal lines with negative slope. 5 diagonal cross-hatch. 6 solid. <p>The second column controls the bar color. See “Colors Appendix” on page B-1.</p>

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

bar

Remarks Use **scale** or **ytics** to fix the scaling for the bar heights.

Example In this example, three overlapping sets of bars will be created. The three heights for the i^{th} bar are stored in $x[i,.]$.

```
library pgraph;
graphset;

t = seqa(0,1,10);
x = (t^2/2).*(1~0.7~0.3);

_plegctl = { 1 4 };
_plegstr = "Accnt #1\000Accnt #2\000Accnt #3";
title("Theoretical Savings Balance");
xlabel("Years");
ylabel("Dollars x 1000");
_pbartyp = { 1 10 }; /* Set color of the */
                        /* bars to 10 (magenta) */
_pnum = 2;
bar(t,x); /* Use t vector to label X axis */
```

Source pbar.src

See also **asclabel**, **xy**, **logx**, **logy**, **loglog**, **scale**, **hist**

base10

Purpose	Break number into a number of the form <code>#.####...</code> and a power of 10.	
Format	<code>{ <i>M,P</i> } = base10(<i>x</i>);</code>	
Input	<i>x</i>	scalar, number to break down.
Output	<i>M</i>	scalar, in the range $-10 < M < 10$.
	<i>P</i>	scalar, integer power such that:
	$M * 10^P = x$	
Example	<code>{ <i>b, e</i> } = base10(4500);</code>	
	<code><i>b</i> = 4.5000000</code>	
	<code><i>e</i> = 3.0000000</code>	
Source	<code>base10.src</code>	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

begwind

begwind

Purpose	Initialize global graphic panel variables.
Library	pgraph
Format	begwind;
Remarks	This procedure must be called before any other graphic panel functions are called.
Source	pwindow.src
See also	endwind, window, makewind, setwind, nextwind, getwind

besselj

Purpose	Computes a Bessel function of the first kind, $J_n(x)$.	
Format	$y = \text{besselj}(n,x);$	
Input	n	NxK matrix, the order of the Bessel function. Nonintegers will be truncated to an integer.
	x	LxM matrix, ExE conformable with n .
Output	y	max(N,L) by max(K,M) matrix.
Example	$n = \{ 0, 1 \};$	
	$x = \{ 0.1 \ 1.2, \ 2.3 \ 3.4 \};$	
	$y = \text{besselj}(n,x);$	
	$= \begin{matrix} 0.99750156 & 0.67113274 \\ 0.53987253 & 0.17922585 \end{matrix}$	
See also	bessely, mbesseli	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

bessely

bessely

a		
b	Purpose	To compute a Bessel function of the second kind (Weber's function), $Y_n(x)$.
c	Format	$y = \text{bessely}(n,x);$
d	Input	n NxK matrix, the order of the Bessel function. Nonintegers will be truncated to an integer.
e		x LxM matrix, ExE conformable with n .
f	Output	y max(N,L) by max(K,M) matrix.
g	Example	$n = \{ \ 0, \ 1 \ };$
h		$x = \{ \ 0.1 \ 1.2, \ 2.3 \ 3.4 \ };$
i		$y = \text{bessely}(n,x);$
j		$y = \begin{array}{cc} -1.5342387 & 0.22808351 \\ 0.05227732 & 0.40101529 \end{array}$
k	See also	besselj, mbesseli
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

box

Purpose	Graph data using the box graph percentile method.	
Library	pgraph	
Format	box (<i>grp</i> , <i>y</i>) ;	
Input	<i>grp</i>	1xM vector. This contains the group numbers corresponding to each column of <i>y</i> data. If scalar 0, a sequence from 1 to cols (<i>y</i>) will be generated automatically for the X axis.
	<i>y</i>	NxM matrix. Each column represents the set of <i>y</i> values for an individual percentiles box symbol.
Global Input	_pboxctl	<p>5x1 vector, controls box style, width, and color.</p> <p>[1] box width between 0 and 1. If zero, the box plot is drawn as two vertical lines representing the quartile ranges with a filled circle representing the 50th percentile.</p> <p>[2] box color. If this is set to 0, the colors may be individually controlled using the global variable _pcolor.</p> <p>[3] Min/max style for the box symbol. One of the following:</p> <ol style="list-style-type: none"> 1 Minimum and maximum taken from the actual limits of the data. Elements 4 and 5 are ignored. 2 Statistical standard with the minimum and maximum calculated according to interquartile range as follows: <div style="margin-left: 40px;"> $\begin{aligned} \text{intqrang} &= 75^{\text{th}} - 25^{\text{th}} \\ \text{min} &= 25^{\text{th}} - 1.5\text{intqrang} \\ \text{max} &= 75^{\text{th}} + 1.5\text{intqrang} \end{aligned}$ <p>Elements 4 and 5 are ignored.</p> </div> 3 Minimum and maximum percentiles taken from elements 4 and 5. <p>[4] Minimum percentile value (0-100) if _pboxctl[3] = 3.</p>

box

		[5]	Maximum percentile value (0-100) if $_pboxctl[3] = 3$.
a		<u>_plctrl</u>	1XM vector or scalar as follows:
b		0	Plot boxes only, no symbols.
c		1	Plot boxes and plot symbols that lie outside the min and max box values.
d		2	Plot boxes and all symbols.
e		-1	Plot symbols only, no boxes.
f			These capabilities are in addition to the usual line control capabilities of <u>_plctrl</u> .
g		<u>_pcolor</u>	1XM vector or scalar for symbol colors. If scalar, all symbols will be one color. See “Colors Appendix” on page B-1.
h	Remarks		If missing values are encountered in the y data, they will be ignored during calculations and will not be plotted.
i			
j	Source		<code>pbox.src</code>
k			
l			
m			
n			
o			
p			
q			
r			
s			
t			
u			
v			
w			
x y z			

boxcox

Purpose	Computes the Box-Cox function.		
Format	$y = \text{boxcox}(x, \text{lambda});$		
Input	x	MxN matrix.	
	lambda	KxL matrix, ExE conformable to x .	
Output	y	$\max(M,L) \times \max(N,K)$.	

Remarks Allowable range for x is:
 $x > 0$
 The **boxcox** function computes

$$\text{boxcox}(x) = \frac{x^\lambda - 1}{\lambda}$$

Example $x = \{.2 \ .3, \ 1.5 \ 2.5\};$
 $\text{lambda} = \{.4, \ 2\};$
 $y = \text{boxcox}(x, \text{lambda})$

$y = \begin{matrix} -1.1867361 & -0.95549787 \\ 0.62500000 & 2.62500000 \end{matrix}$

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

break

break

a		
b	Purpose	Breaks out of a do or for loop.
c	Format	break;
d	Example	<pre>x = rndn(4,4); r = 0; do while r < rows(x); r = r + 1; c = 0; do while c < cols(x); c = c + 1; if c == r; x[r,c] = 1; elseif c > r; break; /* terminate inner do loop */ else; x[r,c] = 0; endif; endo; /* break jumps to the statement */ /* after this endo */ endo; 1.000 0.326 -2.682 -0.594 x = 0.000 1.000 -0.879 0.056 0.000 0.000 1.000 -0.688 0.000 0.000 0.000 1.000</pre>
e		
f		
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w	Remarks	This command works just like in C.
x y z		

break

See also `continue`, `do`, `for`

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

call

call

a	Purpose	Calls a function or procedure when the returned value is not needed and can be ignored, or when the procedure is defined to return nothing.
b		
c	Format	call function_name(argument_list); call function_name;
d		
e	Remarks	This is useful when you need to execute a function or procedure and do not need the value that it returns. It can also be used for calling procedures that have been defined to return nothing.
f		
g		function_name can be any intrinsic GAUSS function, a procedure (proc), or any valid expression.
h	Example	<code>call chol(x);</code> <code>y = det1;</code> The above example is the fastest way to compute the determinant of a positive definite matrix. The result of chol is discarded and det1 is used to retrieve the determinant that was computed during the call to chol .
i		
j		
k		
l		
m	See also	proc
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

cdfbeta

Purpose	Computes the incomplete beta function (i.e., the cumulative distribution function of the beta distribution).		
Format	$y = \text{cdfbeta}(x,a,b);$		
Input	x	NxK matrix.	
	a	LxM matrix, ExE conformable with x .	
	b	PxQ matrix, ExE conformable with x and a .	
Output	y	max(N,L,P) by max(K,M,Q) matrix.	
Remarks	y is the integral from 0 to x of the beta distribution with parameters a and b . Allowable ranges for the arguments are: $0 \leq x \leq 1$ $a > 0$ $b > 0$ A -1 is returned for those elements with invalid inputs.		
Example	$x = \{ .1, .2, .3, .4 \};$ $a = 0.5;$ $b = 0.3;$ $y = \text{cdfbeta}(x,a,b);$ $\begin{matrix} 0.142285 \\ 0.206629 \\ 0.260575 \\ 0.310875 \end{matrix}$		
See also	cdfchic, cdffc, cdfn, cdfnc, cdftc, gamma		

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

cdfbeta

Technical
Notes

cdfbeta has the following approximate accuracy:

$\max(a,b) \leq 500$	the absolute error is approx. $\pm 5e-13$
$500 < \max(a,b) \leq 10,000$	the absolute error is approx. $\pm 5e-11$
$10,000 < \max(a,b) \leq 200,000$	the absolute error is approx. $\pm 1e-9$
$200,000 < \max(a,b)$	Normal approximations are used and the absolute error is approx. $\pm 2e-9$

References

Bol'shev, L.N. "Asymptotically Pearson's Transformations." Teor. Veroyat. Primen. (*Theory of Probability and its Applications*). Vol. 8 No. 2, 1963, 129-55.

Bosten, N. E., and E.L. Battiste. "Remark on Algorithm 179 Incomplete Beta Ratio." *Comm. ACM*. Vol. 17 No. 3, March 1974, 156-57.

Ludwig, O.G. "Algorithm 179 Incomplete Beta Ratio." *Comm. ACM*. Vol. 6 No. 6, June 1963, 314.

Mardia, K.V., and P.J. Zemroch. "Tables of the F- and related distributions with algorithms." Academic Press, NY, 1978. ISBN 0-12-471140-5

Peizer, D.B., and J.W. Pratt. "A Normal Approximation for Binomial, F, Beta, and Other Common, Related Tail Probabilities, I." *Journal of American Statistical Association*. Vol. 63, Dec. 1968, 1416-56.

Pike, M.C., and I.D. Hill. "Remark on Algorithm 179 Incomplete Beta Ratio." *Comm. ACM*. Vol. 10 No. 6, June 1967, 375-76.

cdfbvn

Purpose	Computes the cumulative distribution function of the standardized bivariate Normal density (lower tail).		
Format	$c = \text{cdfbvn}(h,k,r);$		
Input	h	$N \times K$ matrix, the upper limits of integration for variable 1.	
	k	$L \times M$ matrix, $E \times E$ conformable with h , the upper limits of integration for variable 2.	
	r	$P \times Q$ matrix, $E \times E$ conformable with h and k , the correlation coefficients between the two variables.	
Output	c	$\max(N,L,P)$ by $\max(K,M,Q)$ matrix, the result of the double integral from $-\infty$ to h and $-\infty$ to k of the standardized bivariate Normal density $f(x,y,r)$.	

Remarks The function integrated is:

$$f(x, y, r) = \frac{e^{-0.5w}}{2\pi\sqrt{1-r^2}}$$

with

$$w = \frac{x^2 - 2rxy + y^2}{1 - r^2}$$

Thus, x and y have 0 means, unit variances, and correlation = r .

Allowable ranges for the arguments are:

$$-\infty < h < +\infty$$

$$-\infty < k < +\infty$$

$$-1 \leq r \leq 1$$

A -1 is returned for those elements with invalid inputs.

To find the integral under a general bivariate density, with x and y having nonzero means and any positive standard deviations, use the transformation equations:

$$h = (ht - ux) ./ sx;$$

$$k = (kt - uy) ./ sy;$$

cdfbvn

where **ux** and **uy** are the (vectors of) means of x and y , **sx** and **sy** are the (vectors of) standard deviations of x and y , and **ht** and **kt** are the (vectors of) upper integration limits for the untransformed variables, respectively.

See also **cdfn**, **cdftvn**

Technical Notes The absolute error for **cdfbvn** is approximately $\pm 5.0\text{e-}9$ for the entire range of arguments.

References Daley, D.J. “Computation of Bi- and Tri-variate Normal Integral.” *Appl. Statist.* Vol. 23 No. 3, 1974, 435-38.
Owen, D.B. “A Table of Normal Integrals.” *Commun. Statist.-Simula. Computa.*, B9(4). 1980, 389-419.

cdfbvn2

Purpose	Returns cdfbvn of a bounded rectangle.
Format	$y = \text{cdfbvn2}(h, dh, k, dk, r);$
Input	<p>h Nx1 vector, starting points of integration for variable 1.</p> <p>dh Nx1 vector, increments for variable 1.</p> <p>k Nx1 vector, starting points of integration for variable 2.</p> <p>dk Nx1 vector, increments for variable 2.</p> <p>r Nx1 vector, correlation coefficients between the two variables.</p>
Output	y Nx1 vector, the integral over the rectangle bounded by $h, h+dh, k,$ and $k+dk$ of the standardized bivariate Normal distribution.
Remarks	<p>Scalar input arguments are okay; they will be expanded to Nx1 vectors.</p> <p>cdfbvn2 computes:</p> $\text{cdfbvn}(h+dh, k+dk, r) + \text{cdfbvn}(h, k, r) - \text{cdfbvn}(h, k+dk, r) - \text{cdfbvn}(h+dh, k, r).$ <p>cdfbvn2 computes an error estimate for each set of inputs. The size of the error depends on the input arguments. If trap 2 is set, a warning message is displayed when the error reaches $0.01 * \text{abs}(y)$.</p> <p>For an estimate of the actual error, see cdfbvn2e.</p>
Example	<p>Example 1</p> <pre>cdfbvn2(1, -1, 1, -1, 0.5);</pre> <p>produces:</p> <pre>1.4105101488974692e-001</pre> <p>Example 2</p> <pre>cdfbvn2(1, -1e-15, 1, -1e-15, 0.5);</pre> <p>produces:</p> <pre>4.9303806576313238e-32</pre>

cdfbvn2

Example 3

```
cdfbvn2(1,-1e-45,1,-1e-45,0.5);
```

produces:

```
0.0000000000000000e+000
```

Example 4

```
trap 2,2;
```

```
cdfbvn2(1,-1e-45,1,1e-45,0.5);
```

produces:

```
WARNING: Dubious accuracy from cdfbvn2:
```

```
0.000e+000 +/- 2.8e-060
```

```
0.0000000000000000e+000
```

Source `lncdfn.src`

See also `cdfbvn2e`, `lncdfbvn2`

cdfbvn2e

Purpose	Returns cdfbvn of a bounded rectangle.		
Format	$\{ y, e \} = \text{cdfbvn2e}(h, dh, k, dk, r);$		
Input	h	Nx1 vector, starting points of integration for variable 1.	
	dh	Nx1 vector, increments for variable 1.	
	k	Nx1 vector, starting points of integration for variable 2.	
	dk	Nx1 vector, increments for variable 2.	
	r	Nx1 vector, correlation coefficients between the two variables.	
Output	y	Nx1 vector, the integral over the rectangle bounded by h , $h+dh$, k , and $k+dk$ of the standardized bivariate Normal distribution.	
	e	Nx1 vector, an error estimate.	
Remarks	<p>Scalar input arguments are okay; they will be expanded to Nx1 vectors.</p> <p>cdfbvn2e computes cdfbvn($h+dh, k+dk, r$) + cdfbvn(h, k, r) - cdfbvn($h, k+dk, r$) - cdfbvn($h+dh, k, r$).</p> <p>The real answer is $y \pm e$. The size of the error depends on the input arguments.</p>		
Example	Example 1		
	<code>cdfbvn2e(1, -1, 1, -1, 0.5);</code>		
	produces:		
	1.4105101488974692e-001		
	1.9927918166193113e-014		
	Example 2		
	<code>cdfbvn2e(1, -1e-15, 1, -1e-15, 0.5);</code>		
	produces:		

cdfbvn2e

7.3955709864469857e-032
2.8306169312687801e-030

Example 3
`cdfbvn2e(1,-1e-45,1,-1e-45,0.5);`

produces:
0.0000000000000000e+000
2.8306169312687770e-060

See also **cdfbvn2, lncdfbvn2**

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

cdfchic

Purpose	Computes the complement of the cdf of the chi-square distribution.		
Format	$y = \text{cdfchic}(x,n)$		
Input	x	NxK matrix.	
	n	LxM matrix, ExE conformable with x .	
Output	y	max(N,L) by max(K,M) matrix.	
Remarks	y is the integral from x to ∞ of the chi-square distribution with n degrees of freedom.		
	The elements of n must all be positive integers. The allowable ranges for the arguments are:		
	$x \geq 0$		
	$n > 0$		
	A -1 is returned for those elements with invalid inputs.		
	This equals $1-F(x,n)$, where F is the chi-square cdf with n degrees of freedom. Thus, to get the chi-square cdf, subtract cdfchic (x,n) from 1. The complement of the cdf is computed because this is what is most commonly needed in statistical applications, and because it can be computed with fewer problems of roundoff error.		
Example	$x = \{ .1, .2, .3, .4 \};$		
	$n = 3;$		
	$y = \text{cdfchic}(x,n);$		
	$\begin{array}{r} 0.991837 \\ 0.977589 \\ 0.960028 \\ 0.940242 \end{array}$		
	See also	cdfbeta , cdffc , cdfn , cdfnc , cdftc , gamma	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

cdfchic

Technical
Notes

For $n \leq 1000$, the incomplete gamma function is used and the absolute error is approx. $\pm 6e-13$. For $n > 1000$, a Normal approximation is used and the absolute error is $\pm 2e-8$.

For higher accuracy when $n > 1000$, use:

```
1-cdfgam(0.5*x,0.5*n);
```

References

Bhattacharjee, G.P. "Algorithm AS 32, The Incomplete Gamma Integral." *Applied Statistics*. Vol. 19, 1970, 285-87.

Mardia, K.V., and P.J. Zemroch. "Tables of the F- and related distributions with algorithms." Academic Press, NY, 1978. ISBN 0-12-471140-5

Peizer, D.B., and J.W. Pratt. "A Normal Approximation for Binomial, F, Beta, and Other Common, Related Tail Probabilities, I." *Journal of American Statistical Association*. Vol. 63, Dec. 1968, 1416-56.

cdfchii

Purpose	Compute chi-square abscissae values given probability and degrees of freedom.	
Format	<code>c = cdfchii(p,n);</code>	
Input	<i>p</i>	MxN matrix, probabilities.
	<i>n</i>	LxK matrix, EXE conformable with <i>p</i> , degrees of freedom.
Output	<i>c</i>	max(M,L) by max(N,K) matrix, abscissae values for chi-square distribution.
Example	<p>The following generates a 3x3 matrix of pseudo-random numbers with a chi-squared distribution with expected value of 4:</p> <pre> rndseed 464578; x = cdfchii(rndu(3,3),4+zeros(3,3)); 2.1096456 1.9354989 1.7549182 x = 4.4971008 9.2643386 4.3639694 4.5737473 1.3706243 2.5653688 </pre>	
Source	cdfchii.src	
See also	gammaii	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

cdfchinc

cdfchinc

a		
b		
c		
d		
e		
f		
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		
	Purpose	The integral under noncentral chi-square distribution, from 0 to x . It can return a vector of values, but the degrees of freedom and noncentrality parameter must be the same for all values of x .
	Format	$y = \text{cdfchinc}(x,v,d);$
	Input	x Nx1 vector, values of upper limits of integrals, must be greater than 0. v scalar, degrees of freedom, $v > 0$. d scalar, noncentrality parameter, $d > 0$. This is the square root of the noncentrality parameter that sometimes goes under the symbol lambda. (See Scheffe, <i>The Analysis of Variance</i> , App. IV. 1959.)
	Output	y Nx1 vector, integrals from 0 to x of noncentral chi-square.
	Example	$x = \{ .5, 1, 5, 25 \};$ $p = \text{cdfchinc}(x,4,2);$ 0.0042086234 0.016608592 0.30954232 0.99441140
	Source	<code>cdfnonc.src</code>
	See also	<code>cdffnc</code> , <code>cdftnc</code>
	Technical Notes	Relation to <code>cdfchic</code> : $\text{cdfchic}(x,v) = 1 - \text{cdfchinc}(x,v,0);$ The formula used is taken from Abramowitz and Stegun, <i>Handbook of Mathematical Functions</i> . Formula 26.4.25. 1970, 942.

cdfFc

Purpose Computes the complement of the cdf of the F distribution.

Format $y = \text{cdfFc}(x, n1, n2);$

Input

- x NxK matrix.
- $n1$ LxM matrix, ExE conformable with x .
- $n2$ PxQ matrix, ExE conformable with x and $n1$.

Output y max(N,L,P) by max(K,M,Q) matrix.

Remarks y is the integral from x to ∞ of the F distribution with $n1$ and $n2$ degrees of freedom.

Allowable ranges for the arguments are:

$$\begin{aligned} x &\geq 0 \\ n1 &> 0 \\ n2 &> 0 \end{aligned}$$

A -1 is returned for those elements with invalid inputs.

This equals $1-G(x, n1, n2)$, where G is the F cdf with $n1$ and $n2$ degrees of freedom. Thus, to get the F cdf, subtract $\text{cdfFc}(x, n1, n2)$ from 1. The complement of the cdf is computed because this is what is most commonly needed in statistical applications, and because it can be computed with fewer problems of roundoff error.

Example $x = \{ .1, .2, .3, .4 \};$

$n1 = 0.5;$

$n2 = 0.3;$

$y = \text{cdfFc}(x, n1, n2);$

$y =$

0.751772
0.708152
0.680365
0.659816

cdffc

See also **cdfbeta, cdfchic, cdfn, cdfnc, cdftc, gamma**

Technical Notes For $\max(n1,n2) \leq 1000$, the absolute error is approximately $\pm 5e-13$. For $\max(n1,n2) > 1000$, Normal approximations are used and the absolute error is approximately $\pm 2e-6$.

For higher accuracy when $\max(n1,n2) > 1000$, use:

$$cdfbeta(n2/(n2+n1*x), n2/2, n1/2);$$

References Bol'shev, L.N. "Asymptotically Pearson's Transformations." Teor. Veroyat. Primen. (*Theory of Probability and its Applications*). Vol. 8 No. 2, 1963, 129-55.

Bosten, N. E., and E.L. Battiste. "Remark on Algorithm 179 Incomplete Beta Ratio." *Comm. ACM*. Vol. 17 No. 3, March 1974, 156-57.

Kennedy, W.J., Jr., and J.E. Gentle. *Statistical Computing*. Marcel Dekker, Inc., NY, 1980.

Ludwig, O.G. "Algorithm 179 Incomplete Beta Ratio." *Comm. ACM*. Vol. 6 No. 6, June 1963, 314.

Mardia, K.V., and P.J. Zemroch. "Tables of the F- and related distributions with algorithms." Academic Press, NY, 1978. ISBN 0-12-471140-5

Peizer, D.B., and J.W. Pratt. "A Normal Approximation for Binomial, F, Beta, and Other Common, Related Tail Probabilities, I." *Journal of American Statistical Association*. Vol. 63, Dec. 1968, 1416-56.

Pike, M.C., and I.D. Hill. "Remark on Algorithm 179 Incomplete Beta Ratio." *Comm. ACM*. Vol. 10 No. 6, June 1967, 375-76.

cdffnc

Purpose	The integral under noncentral F distribution, from 0 to x .	
Format	$y = \text{cdffnc}(x, v1, v2, d);$	
Input	x	Nx1 vector, values of upper limits of integrals, $x > 0$.
	$v1$	scalar, degrees of freedom of numerator, $v1 > 0$.
	$v2$	scalar, degrees of freedom of denominator, $v2 > 0$.
	d	scalar, noncentrality parameter, $d > 0$.
		This is the square root of the noncentrality parameter that sometimes goes under the symbol lambda. (See Scheffe, <i>The Analysis of Variance</i> , App. IV. 1959.)
Output	y	Nx1 vector of integrals from 0 to x of noncentral F .
Source	cdfnonc.src	
See also	cdftnc , cdfchinc	
Technical Notes	Relation to cdffc :	
	$\text{cdffc}(x, v1, v2) = 1 - \text{cdffnc}(x, v1, v2, 0);$	
	The formula used is taken from Abramowitz and Stegun, <i>Handbook of Mathematical Functions</i> . Formula 26.6.20. 1970, 947.	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

cdfgam

cdfgam

a		
b		
c		
d		
e		
f		
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		
	Purpose	Incomplete gamma function.
	Format	$g = \text{cdfgam}(x, \text{intlim}) ;$
	Input	x NxK matrix of data. intlim LxM matrix, ExE compatible with x , containing the integration limit.
	Output	g max(N,L) by max(K,M) matrix.
	Remarks	The incomplete gamma function returns the integral $\int_0^{\text{intlim}} \frac{e^{-t} t^{(x-1)}}{\text{gamma}(x)} dt$ The allowable ranges for the arguments are: $x > 0$ $\text{intlim} \geq 0$ A -1 is returned for those elements with invalid inputs.
	Example	$x = \{ 0.5 \ 1 \ 3 \ 10 \} ;$ $\text{intlim} = \text{sega}(0, .2, 6) ;$ $g = \text{cdfgam}(x, \text{intlim}) ;$ $x = 0.500000 \ 1.00000 \ 3.00000 \ 10.00000$ $\text{intlim} =$ $\begin{matrix} 0.000000 \\ 0.200000 \\ 0.400000 \\ 0.600000 \\ 0.800000 \\ 1.000000 \end{matrix}$

cdfgam

	0.000000	0.000000	0.000000	0.000000
	0.472911	0.181269	0.00114848	2.35307E - 014
$g =$	0.628907	0.329680	0.00792633	2.00981E - 011
	0.726678	1.451188	0.0231153	9.66972E - 010
	0.794097	0.550671	0.0474226	1.43310E - 008
	0.842701	0.632120	0.0803014	1.11425E - 007

This computes the integrals over the range from 0 to 1, in increments of .2, at the parameter values 0.5, 1, 3, 10.

Technical Notes

cdfgam has the following approximate accuracy:

$x < 500$	the absolute error is approx. $\pm 6e-13$
$500 \leq x \leq 10,000$	the absolute error is approx. $\pm 3e-11$
$10,000 < x$	a Normal approximation is used and the absolute error is approx. $\pm 3e-10$

References

Bhattacharjee, G.P. "Algorithm AS 32, The Incomplete Gamma Integral." *Applied Statistics*. Vol. 19, 1970, 285-87.

Peizer, D.B., and J.W. Pratt. "A Normal Approximation for Binomial, F, Beta, and Other Common, Related Tail Probabilities, I." *Journal of American Statistical Association*. Vol. 63, Dec. 1968, 1416-56.

Pike, M.C., and I.D. Hill. "Remark on Algorithm 179 Incomplete Beta Ratio." *Comm. ACM*. Vol. 10 No. 6, June 1967, 375-76.

cdfmvn

cdfmvn

a	Purpose	Computes multivariate Normal cumulative distribution function.
b		
c	Format	$y = \text{cdfmvn}(x,r);$
d	Input	x KxL matrix, abscissae.
e		r KxK matrix, correlation matrix.
f	Output	y Lx1 vector, $Pr(X < x \mid r)$.
g	Source	lncdfn.src
h	See also	cdfbvn, cdfn, cdftvn, lncdfmvn
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

cdfn, cdfnc

Purpose **cdfn** computes the cumulative distribution function (cdf) of the Normal distribution. **cdfnc** computes 1 minus the cdf of the Normal distribution.

Format `n = cdfn(x);`
`nc = cdfnc(x);`

Input `x` NxK matrix.

Output `n` NxK matrix.
`nc` NxK matrix.

Remarks `n` is the integral from $-\infty$ to x of the Normal density function, and `nc` is the integral from x to $+\infty$.

Note that: **cdfn**(x) + **cdfnc**(x) = 1. However, many applications expect **cdfn**(x) to approach 1, but never actually reach it. Because of this, we have capped the return value of **cdfn** at 1 - machine epsilon, or approximately 1 - 1.11e-16. As the relative error of **cdfn** is about $\pm 5e-15$ for **cdfn**(x) around 1, this does not invalidate the result. What it does mean is that for **abs**(x) > (approx.) 8.2924, the identity does not hold true. If you have a need for the uncapped value of **cdfn**, the following code will return it:

```
n = cdfn(x);
if n >= 1-eps;
    n = 1;
endif;
```

where the value of machine epsilon is obtained as follows:

```
x = 1;
do while 1-x /= 1;
    eps = x;
    x = x/2;
end;
```

Note that this is an alternate definition of machine epsilon. Machine epsilon is usually defined as the smallest number such that 1 + machine

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

cdfn, cdfnc

$\epsilon > 1$, which is about $2.23e-16$. This defines machine epsilon as the smallest number such that $1 - \text{machine epsilon} < 1$, or about $1.11e-16$.

The **erf** and **erfc** functions are also provided, and may sometimes be more useful than **cdfn** and **cdfnc**.

Example `x = { -2 -1 0 1 2 };`
`n = cdfn(x);`
`nc = cdfnc(x);`

<i>x</i> =	-2.00000	-1.00000	0.00000	1.00000	2.00000
<i>n</i> =	0.02275	0.15866	0.50000	0.84134	0.97725
<i>nc</i> =	0.97725	0.84134	0.50000	0.15866	0.02275

See also **erf, erfc, cdfbeta, cdfchic, cdftc, cdffc, gamma**

Technical Notes For the integral from $-\infty$ to x :

$x \leq -37$,	cdfn underflows and 0.0 is returned
$-36 < x < -10$,	cdfn has a relative error of approx. $\pm 5e-12$
$-10 < x < 0$,	cdfn has a relative error of approx. $\pm 1e-13$
$0 < x$,	cdfn has a relative error of approx. $\pm 5e-15$

For **cdfnc**, i.e., the integral from x to $+\infty$, use the above accuracies but change x to $-x$.

References Adams, A.G. "Remark on Algorithm 304 Normal Curve Integral." *Comm. ACM*. Vol. 12 No. 10, Oct. 1969, 565-66.

Hill, I.D., and S.A. Joyce. "Algorithm 304 Normal Curve Integral." *Comm. ACM*. Vol. 10 No. 6, June 1967, 374-75.

Holmgren, B. "Remark on Algorithm 304 Normal Curve Integral." *Comm. ACM*. Vol. 13 No. 10, Oct. 1970.

Mardia, K.V., and P.J. Zemroch. "Tables of the F- and related distributions with algorithms." Academic Press, NY, 1978. ISBN 0-12-471140-5

cdfn2

Purpose Computes the integral over a Normal density function interval.

Format `y = cdfn2(x,dx);`

Input

<i>x</i>	MxN matrix, abscissae.
<i>dx</i>	KxL matrix, EXE conformable to <i>x</i> , intervals.

Output

<i>y</i>	max(M,K) by max(N,L) matrix, the integral from <i>x</i> to <i>x+dx</i> of the Normal distribution, i.e., $Pr(x \leq X \leq x + dx)$.
----------	---

Remarks The relative error is:

$ x \leq 1$ and $dx \leq 1$	$\pm 1e-14$
$1 < x < 37$ and $ dx < 1/ x $	$\pm 1e-13$
$\min(x, x+dx) > -37$ and $y > 1e-300$	$\pm 1e-11$ or better

A relative error of $\pm 1e-14$ implies that the answer is accurate to better than ± 1 in the 14th digit.

Example

```
print cdfn2(1,0.5);
9.1848052662599017e-02
print cdfn2(20,0.5);
2.7535164718736454e-89
print cdfn2(20,1e-2);
5.0038115018684521e-90
print cdfn2(-5,2);
1.3496113800582164e-03
print cdfn2(-5,0.15);
3.3065580013000255e-07
```

Source `lncdfn.src`

See also `lncdfn2`

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

cdfni

cdfni

a	<div>Purpose</div> <div>Computes the inverse of the cdf of the Normal distribution.</div>	
b		
c		
d		
e		
f	<div>Format</div>	<div>$x = \text{cdfni}(p);$</div>
g	<div>Input</div>	<div>p NxK real matrix, Normal probability levels, $0 \leq p \leq 1$.</div>
h	<div>Output</div>	<div>x NxK real matrix, Normal deviates, such that $\text{cdfn}(x) = p$</div>
i	<div>Remarks</div>	<div>$\text{cdfn}(\text{cdfni}(p)) = p$ to within the errors given below:</div>
j		<div>$p \leq 4.6\text{e-}308$ -37.5 is returned</div>
k		<div>$4.6\text{e-}308 < p < 5\text{e-}24$ accurate to ± 5 in 12th digit</div>
l		<div>$5\text{e-}24 < p < 0.5$ accurate to ± 1 in 13th digit</div>
m		<div>$0.5 < p < 1 - 2.22045\text{e-}16$ accurate to ± 5 in 15th digit</div>
n		<div>$p \geq 1 - 2.22045\text{e-}16$ 8.12589... is returned</div>
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

cdf_{tc}

Purpose Computes the complement of the cdf of the Student's t distribution.

Format `y = cdftc(x,n);`

Input

<i>x</i>	NxK matrix.
<i>n</i>	LxM matrix, ExE conformable with <i>x</i> .

Output *y* max(N,L) by max(K,M) matrix.

Remarks *y* is the integral from *x* to ∞ of the t distribution with *n* degrees of freedom.

Allowable ranges for the arguments are:

$$-\infty < x < +\infty$$

$$n > 0$$

A -1 is returned for those elements with invalid inputs.

This equals $1-F(x,n)$, where F is the t cdf with *n* degrees of freedom. Thus, to get the t cdf, subtract **cdf_{tc}(*x*,*n*)** from 1. The complement of the cdf is computed because this is what is most commonly needed in statistical applications, and because it can be computed with fewer problems of roundoff error.

Example `x = { .1, .2, .3, .4 };`

`n = 25;`

`y = cdftc(x,n);`

`print y;`

```

          0.473165
y =      0.447100
          0.422428
          0.399555

```

See also **cdfbeta**, **cdfchic**, **cdffnc**, **cdfn**, **cdfnc**, **gamma**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

cdftc

Technical
Notes

For results greater than 0.5e-30, the absolute error is approximately $\pm 1\text{e-}14$ and the relative error is approximately $\pm 1\text{e-}12$. If you multiply the relative error by the result, then take the minimum of that and the absolute error, you have the maximum actual error for any result. Thus, the actual error is approximately $\pm 1\text{e-}14$ for results greater than 0.01. For results less than 0.01, the actual error will be less. For example, for a result of 0.5e-30, the actual error is only $\pm 0.5\text{e-}42$.

References

Abramowitz, M., and I. A. Stegun, eds. *Handbook of Mathematical Functions*. 7th ed. Dover, NY, 1970. ISBN 0-486-61272-4

Hill, G.W. "Algorithm 395 Student's t-Distribution." *Comm. ACM*. Vol. 13 No. 10, Oct. 1970.

Hill, G.W. "Student's t-Distribution Quantiles to 20D." *Division of Mathematical Statistics Technical Paper No. 35*. Commonwealth Scientific and Industrial Research Organization, Australia, 1972.

cdftci

Purpose	Computes the inverse of the complement of the Student's t cdf.	
Format	$x = \text{cdftci}(p,n);$	
Input	p	NxK real matrix, complementary Student's t probability levels, $0 \leq p \leq 1$.
	n	LxM real matrix, degrees of freedom, $n \geq 1$, n need not be integral. EXE conformable with p .
Output	x	max(N,L) by max(K,M) real matrix, Student's t deviates, such that $\text{cdftc}(x,n) = p$.
Remarks	<p>$\text{cdftc}(\text{cdftci}(p,n)) = p$ to within the errors given below:</p> <p> $0.5\text{e-}30 < p < 0.01$ accurate to ± 1 in 12th digit $0.01 < p$ accurate to $\pm 1\text{e-}14$ </p> <p>Extreme values of arguments can give rise to underflows, but no overflows are generated.</p>	

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

cdftnc

cdftnc

a		
b		
c		
d		
e		
f		
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		
	Purpose	The integral under noncentral Student's t distribution, from $-\infty$ to x . It can return a vector of values, but the degrees of freedom and noncentrality parameter must be the same for all values of x .
	Format	$y = \text{cdftnc}(x,v,d);$
	Input	x $N \times 1$ vector, values of upper limits of integrals. v scalar, degrees of freedom, $v > 0$. d scalar, noncentrality parameter. This is the square root of the noncentrality parameter that sometimes goes under the symbol lambda. (See Scheffe, <i>The Analysis of Variance</i> , App. IV. 1959.)
	Output	y $N \times 1$ vector, integrals from $-\infty$ to x of noncentral t .
	Source	cdfnonc.src
	See also	cdffnc , cdfchin
	Technical Notes	Relation to cdftc : $\text{cdftc}(x,v) = 1 - \text{cdftnc}(x,v,0);$ The formula used is based on the formula in <i>SUGI Supplemental Library User's Guide</i> . SAS Institute. 1983, 232 (which is attributed to Johnson and Kotz, 1970). The formula used here is a modification of that formula. It has been tested against direct numerical integration, and against simulation experiments in which noncentral t random variates were generated and the cdf found directly.

cdftvn

Purpose	Computes the cumulative distribution function of the standardized trivariate Normal density (lower tail).		
Format	$c = \text{cdftvn}(x1,x2,x3,\text{rho12},\text{rho23},\text{rho31}) ;$		
Input	$x1$	Nx1 vector of upper limits of integration for variable 1.	
	$x2$	Nx1 vector of upper limits of integration for variable 2.	
	$x3$	Nx1 vector of upper limits of integration for variable 3.	
	rho12	scalar or Nx1 vector of correlation coefficients between the two variables $x1$ and $x2$.	
	rho23	scalar or Nx1 vector of correlation coefficients between the two variables $x2$ and $x3$.	
	rho31	scalar or Nx1 vector of correlation coefficients between the two variables $x1$ and $x3$.	
Output	c	Nx1 vector containing the result of the triple integral from $-\infty$ to $x1$, $-\infty$ to $x2$, and $-\infty$ to $x3$ of the standardized trivariate Normal density: $f(x1,x2,x3,\text{rho12},\text{rho23},\text{rho31})$	
Remarks	Allowable ranges for the arguments are: $-\infty < x1 < +\infty$ $-\infty < x2 < +\infty$ $-\infty < x3 < +\infty$ $-1 < \text{rho12} < 1$ $-1 < \text{rho23} < 1$ $-1 < \text{rho31} < 1$ In addition, rho12 , rho23 , and rho31 must come from a legitimate positive definite matrix. A -1 is returned for those rows with invalid inputs. A separate integral is computed for each row of the inputs. The first 3 arguments ($x1,x2,x3$) must be the same length, N . The second 3 arguments ($\text{rho12},\text{rho23},\text{rho31}$) must also be the same length, and this		

cdftvn

length must be N or 1. If it is 1, then these values will be expanded to apply to all values of $x1,x2,x3$. All inputs must be column vectors.

To find the integral under a general trivariate density, with $x1$, $x2$, and $x3$ having nonzero means and any positive standard deviations, transform by subtracting the mean and dividing by the standard deviation. For example:

```
x1 = ( x1 - meanc(x1) ) / stdc(x1);
```

See also **cdfn**, **cdfbvn**

Technical Notes The absolute error for **cdftvn** is approximately $\pm 2.5e-8$ for the entire range of arguments.

References Daley, D.J. "Computation of Bi- and Tri-variate Normal Integral." *Appl. Statist.* Vol. 23 No. 3, 1974, 435-38.

Steck, G.P. "A Table for Computing Trivariate Normal Probabilities." *Ann. Math. Statist.* Vol. 29, 780-800.

cdir

Purpose	Returns the current directory.	
Format	<code>y = cdir(s);</code>	
Input	<i>s</i>	string, if the first character is 'A'-'Z' and the second character is a colon ':' then that drive will be used. If not, the current default drive will be used.
Output	<i>y</i>	string containing the drive and full path name of the current directory on the specified drive.
Remarks	<p>If the current directory is the root directory, the returned string will end with a backslash, otherwise it will not.</p> <p>A null string or scalar zero can be passed in as an argument to obtain the current drive and path name.</p>	
Example	<pre>x = cdir(0); y = cdir("d:"); print x; print y; C:\GAUSS D:\</pre>	
See also	files	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

ceil

ceil

a	Purpose	Round up toward $+\infty$.
b	Format	$y = \text{ceil}(x);$
c	Input	x NxK matrix.
d	Output	y NxK matrix.
e	Remarks	This rounds every element in the matrix x to an integer. The elements are rounded up toward $+\infty$.
f	Example	$x = 100*\text{rndn}(2,2);$
g		$y = \text{ceil}(x);$
h		$x = \begin{array}{rr} 77.68 & -14.10 \\ 4.73 & -158.88 \end{array}$
i		$y = \begin{array}{rr} 78.00 & -14.00 \\ 5.00 & -158.00 \end{array}$
j	See also	floor, trunc
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

ChangeDir

Purpose	Changes the working directory.	
Format	<code>d = ChangeDir(s);</code>	
Input	<i>s</i>	string, directory to change to.
Output	<i>d</i>	string, new working directory, or null string if change failed.
See also	chdir	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

chdir

chdir

Purpose	Changes working directory.
Format	chdir <i>dirstr</i> ;
Input	<i>dirstr</i> literal or ^string, directory to change to.
Remarks	This is for interactive use. Use ChangeDir in a program. If the directory change fails, chdir prints an error message.

chol

Purpose	Computes the Cholesky decomposition of a symmetric, positive definite matrix.		
Format	$y = \text{chol}(x);$		
Input	x	NxN matrix.	
Output	y	NxN matrix containing the Cholesky decomposition of x .	
Remarks	<p>y is the “square root” matrix of x. That is, it is an upper triangular matrix such that $x = y'y$.</p> <p>chol does not check to see that the matrix is symmetric. chol will look only at the upper half of the matrix including the principal diagonal.</p> <p>If the matrix x is symmetric but not positive definite, either an error message or an error code will be generated, depending on the lowest order bit of the trap flag:</p> <p style="padding-left: 40px;">trap 0 Print error message and terminate program.</p> <p style="padding-left: 40px;">trap 1 Return scalar error code 10.</p> <p>See scalerr and trap for more details about error codes.</p>		
Example	<pre>x = moment(rndn(100,4),0); y = chol(x); ypy = y'y;</pre> <div>$x = \begin{bmatrix} 90.746566 & -6.467195 & -1.927489 & -15.696056 \\ -6.467195 & 87.806557 & 6.319043 & -2.435953 \\ -1.927489 & 6.319043 & 101.973276 & 4.355520 \\ -15.696056 & -2.435953 & 4.355520 & 99.042850 \end{bmatrix}$</div> <div>$y = \begin{bmatrix} 9.526099 & -0.678892 & -0.202338 & -1.647690 \\ 0.000000 & 9.345890 & 0.661433 & -0.380334 \\ 0.000000 & 0.000000 & 10.074465 & 0.424211 \\ 0.000000 & 0.000000 & 0.000000 & 9.798130 \end{bmatrix}$</div>		

chol

$$y p y = \begin{bmatrix} 90.746566 & -6.467195 & -1.927489 & -15.696056 \\ -6.467195 & 87.806557 & 6.319043 & -2.435953 \\ -1.927489 & 6.319043 & 101.973276 & 4.355520 \\ -15.696056 & -2.435953 & 4.355520 & 99.042850 \end{bmatrix}$$

See also `crout`, `solpd`

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

choldn

Purpose	Performs a Cholesky downdate of one or more rows on an upper triangular matrix.		
Format	$r = \text{choldn}(C,x);$		
Input	C	KxK upper triangular matrix.	
	x	NxK matrix, the rows to downdate C with.	
Output	r	KxK upper triangular matrix, the downdated matrix.	
Remarks	<p>C should be a Cholesky factorization.</p> <p>choldn(C,x) is equivalent to chol(C'C - x'x), but choldn is numerically much more stable.</p> <p>Warning: it is possible to render a Cholesky factorization non-positive definite with choldn. You should keep an eye on the ratio of the largest diagonal element of r to the smallest — if it gets very large, r may no longer be positive definite. This ratio is a rough estimate of the condition number of the matrix.</p>		
Example	<pre>let C[3,3] = 20.16210005 16.50544413 9.86676135 0 11.16601462 2.97761666 0 0 11.65496052; let x[2,3] = 1.76644971 7.49445820 9.79114666 6.87691156 4.41961438 4.32476921; r = choldn(C,x); 18.87055964 15.32294435 8.04947012 r = 0.00000000 9.30682813 -2.12009339 0.00000000 0.00000000 7.62878355</pre>		
See also	cholup		

cholsol

cholsol

a		
b		
c		
d		
e		
f		
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		
	Purpose	Solves a system of linear equations given the Cholesky factorization of the system.
	Format	$x = \text{cholsol}(b,C);$
	Input	b NxK matrix. C NxN matrix.
	Output	x NxK matrix.
	Remarks	C is the Cholesky factorization of a linear system of equations A . x is the solution for $Ax = b$. b can have more than one column. If so, the system is solved for each column, i.e., $A*x[:,i] = b[:,i]$. $\text{cholsol}(\text{eye}(N),C)$ is equivalent to $\text{invpd}(A)$. Thus, if you have the Cholesky factorization of A , cholsol is the most efficient way to obtain the inverse of A .
	Example	<pre>let b[3,1] = 0.03177513 0.41823100 1.70129375; let C[3,3] = 1.73351215 1.53201723 1.78102499 0 1.09926365 0.63230050 0 0 0.67015361; x = cholsol(b,C); -1.94396905 x = -1.52686768 3.21579513 3.00506436 2.65577048 3.08742844 A0 = 2.65577048 3.55545737 3.42362593 3.08742844 3.42362593 4.02095978</pre>

cholup

Purpose	Performs a Cholesky update of one or more rows on an upper triangular matrix.		
Format	$r = \text{cholup}(C,x);$		
Input	C	KxK upper triangular matrix.	
	x	NxK matrix, the rows to update C with.	
Output	r	KxK upper triangular matrix, the updated matrix.	
Remarks	C should be a Cholesky factorization. cholup (C , x) is equivalent to chol (C' C + x' x), but cholup is numerically much more stable.		
Example	<pre>let C[3,3] = 18.87055964 15.32294435 8.04947012 0 9.30682813 -2.12009339 0 0 7.62878355; let x[2,3] = 1.76644971 7.49445820 9.79114666 6.87691156 4.41961438 4.32476921; r = cholup(C,x); 20.16210005 16.50544413 9.86676135 r = 0.00000000 11.16601462 2.97761666 0.00000000 0.00000000 11.65496052</pre>		
See also	choldn		

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

chrs

chrs

Purpose	Converts a matrix of ASCII values into a string containing the appropriate characters.		
Format	$y = \text{chrs}(x);$		
Input	x	NxK matrix.	
Output	y	string of length N*K containing the characters whose ASCII values are equal to the values in the elements of x .	
Remarks	This function is useful for embedding control codes in strings and for creating variable length strings when formatting printouts, reports, etc.		
Example	<pre>n = 5; print chrs(ones(n,1)*42); ***** Since the ASCII value of the asterisk character is 42, the program above will print a string of n asterisks. y = chrs(67~65~84); print y; CAT</pre>		
See also	vals, ftos, stof		

clear

Purpose	Clears space in memory by setting matrices equal to scalar zero.
Format	clear <i>x, y</i> ;
Remarks	<p>clear x; is equivalent to x = 0;</p> <p>Matrix names are retained in the symbol table after they are cleared.</p> <p>Matrices can be clear'ed even though they have not previously been defined. clear can be used to initialize matrices to scalar 0.</p>
Example	<code>clear x;</code>
See also	clearg, new, show, delete

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

clearg

clearg

a	Purpose	This command clears global symbols by setting them equal to scalar zero.
b	Format	clearg <i>a,b,c</i> ;
c	Output	<i>a,b,c</i> scalar global matrices containing 0.
d	Remarks	clearg x ; is equivalent to x = 0 ;, where x is understood to be a global symbol. clearg can be used to initialize symbols not previously referenced. This command can be used inside procedures to clear global matrices. It will ignore any locals by the same name.
e	Example	<pre>x = 45; clearg x; x = 0.0000000</pre>
f	See also	clear, delete, new, show, local
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

close

Purpose	Close a GAUSS file.
Format	<code>y = close(handle);</code>
Input	<i>handle</i> scalar, the file handle given to the file when it was opened with the open , create , or fopen command.
Output	<i>y</i> scalar, 0 if successful, -1 if unsuccessful.
Remarks	<p><i>handle</i> is the scalar file handle created when the file was opened. It will contain an integer which can be used to refer to the file.</p> <p>close will close the file specified by <i>handle</i>, and will return a 0 if successful and a -1 if not successful. The handle itself is not affected by close unless the return value of close is assigned to it.</p> <p>If <i>fl</i> is a file handle and it contains the value 7, then after:</p> <pre>call close(fl);</pre> <p>the file will be closed but <i>fl</i> will still have the value 7. The best procedure is to do the following:</p> <pre>fl = close(fl);</pre> <p>This will set <i>fl</i> to 0 upon a successful close.</p> <p>It is important to set unused file handles to zero because both open and create check the value that is in a file handle before they proceed with the process of opening a file. During open or create, if the value that is in the file handle matches that of an already open file, the process will be aborted and a “File already open” message will be given. This gives you some protection against opening a second file with the same handle as a currently open file. If this happened, you would no longer be able to access the first file.</p> <p>An advantage of the close function is that it returns a result which can be tested to see if there were problems in closing a file. The most common reason for having a problem in closing a file is that the disk on which the file is located is no longer in the disk drive — or the handle was invalid. In both of these cases, close will return a -1.</p> <p>Files are not automatically closed when a program terminates. This allows users to run a program that opens files, and then access the files</p>

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

close

from interactive mode after the program has been run. Files are automatically closed when GAUSS exits to the operating system or when a program is terminated with the **end** statement. **stop** will terminate a program but not close files.

As a rule it is good practice to make **end** the last statement in a program, unless further access to the open files is desired from interactive mode. You should close files as soon as you are done writing to them to protect against data loss in the case of abnormal termination of the program due to a power or equipment failure.

The danger in not closing files is that anything written to the files may be lost. The disk directory will not reflect changes in the size of a file until the file is closed and system buffers may not be flushed.

Example

```
open f1 = dat1 for append;  
y = writer(f1,x);  
f1 = close(f1);
```

See also `closeall`

closeall

Purpose Close all currently open GAUSS files.

Format `closeall;`
`closeall list_of_handles;`

Remarks `list_of_handles` is a comma-delimited list of file handles.

`closeall` with no specified list of handles will close all files. The file handles will not be affected. The main advantage of using `closeall` is ease of use; the file handles do not have to be specified, and one statement will close all files.

When a list of handles follows `closeall`, all files are closed and the file handles listed are set to scalar 0. This is safer than `closeall` without a list of handles because the handles are cleared.

It is important to set unused file handles to zero because both `open` and `create` check the value that is in a file handle before they proceed with the process of opening a file. During `open` or `create`, if the value that is in the file handle matches that of an already open file, the process will be aborted and a “File already open” message will be given. This gives you some protection against opening a second file with the same handle as a currently open file. If this happened, you would no longer be able to access the first file.

Files are not automatically closed when a program terminates. This allows users to run a program that opens files, and then access the files from interactive mode after the program has been run. Files are automatically closed when GAUSS exits to the operating system or when a program is terminated with the `end` statement. `stop` will terminate a program but not close files.

As a rule it is good practice to make `end` the last statement in a program, unless further access to the open files is desired from interactive mode. You should close files as soon as you are done writing to them to protect against data loss in the case of abnormal termination of the program due to a power or equipment failure.

The danger in not closing files is that anything written to the files may be lost. The disk directory will not reflect changes in the size of a file until the file is closed and system buffers may not be flushed.

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

closeall

Example `open f1 = dat1 for read;`
 `open f2 = dat1 for update;`
 `x = readr(f1,rowsf(f1));`
 `x = sqrt(x);`
 `call writer(f2,x);`
 `closeall f1,f2;`

See also **close, open**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

cls

Purpose	Clear the window.
Format	cls;
Portability	<p>UNIX 3.2 only</p> <p>cls clears the active graphic panel. For Text graphic panels, this means the graphic panel buffer is cleared to the background color. For TTY graphic panels, the current output line is panned to the top of the graphic panel, effectively clearing the display. The output log is still intact. To clear the output log of a TTY graphic panel, use WinClearTTYLog. For PQG graphic panels, the graphic panel is cleared to the background color, and the related graphics file is truncated to zero length.</p> <p>UNIX 3.5+</p> <p>cls will clear the screen on some terminals.</p> <p>Windows</p> <p>cls clears the Command window if you're in Cmnd I/O mode, the Output window if you're in Split I/O mode.</p> <p>OS/2</p> <p>cls clears the Main window.</p>
Remarks	This command will cause the window to clear and will locate the cursor at the upper left hand corner of the window.
See also	locate

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

code

code

a	Purpose	Allows a new variable to be created (coded) with different values depending upon which one of a set of logical expressions is true.	
b			
c	Format	$y = \text{code}(e,v);$	
d	Input	e	NxK matrix of 1's and 0's. Each column of this matrix is created by a logical expression using "dot" conditional and boolean operators. Each of these expressions should return a column vector result. The columns are horizontally concatenated to produce e . If more than one of these vectors contains a 1 in any given row, the code function will terminate with an error message.
e		v	(K+1)x1 vector containing the values to be assigned to the new variable.
f	Output	y	Nx1 vector containing the new values.
g			
h	Remarks	If none of the K expressions is true, the new variable is assigned the default value, which is given by the last element of v .	
i	Example	let x1 = 0 /* column vector of original values */	
j		5	
k		10	
l		15	
m		20;	
n		let v = 1 /* column vector of new values */	
o		2	
p		3; /* the last element of v is the	
q		:: "default"	
r		*/	
s			
t			
u			
v			
w			
x y z			

code

```

e1 = (0 .lt x1) .and (x1 .le 5); /* expression 1
                                */
e2 = (5 .lt x1) .and (x1 .le 25); /* expression 2
                                */
e = e1~e2; /* concatenate e1 & e2 to make a 1,0
           :: mask with one less column than the
           :: number of new values in v.
           */

y = code(e,v);

      0
      5
x1[5, 1] = 10      (column vector of original values)
      15
      20

v[3, 1] = 1 2 3      (Note: v is a column vector)

      0 0
      1 0
e[5, 2] = 0 1
      0 1
      0 1

      3
      1
y[5, 1] = 2
      2
      2

```

For every row in *e*, if a 1 is in the first column, the first element of *v* is used. If a 1 is in the second column, the second element of *v* is used, and

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

code

so on. If there are only zeros in the row, the last element of v is used. This is the default value.

If there is more than one 1 in any row of e , the function will terminate with an error message.

Source `datatran.src`

See also **`recode`**, **`substute`**

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

code (dataloop)

Purpose	Creates new variables with different values based on a set of logical expressions.	
Format	<pre>code [[#]] [[\$] var [[default defval]] with val_1 for expression_1, val_2 for expression_2, . . . val_n for expression_n;</pre>	
Input	<div> <div><i>var</i></div> <div>literal, the new variable name.</div> </div> <div> <div><i>defval</i></div> <div>scalar, the default value if none of the expressions are <i>TRUE</i>.</div> </div> <div> <div><i>val</i></div> <div>scalar, value to be used if corresponding expression is <i>TRUE</i>.</div> </div> <div> <div><i>expression</i></div> <div>logical scalar-returning expression that returns nonzero <i>TRUE</i> or zero <i>FALSE</i>.</div> </div>	
Remarks	<p>If '\$' is specified, the new variable will be considered a character variable. If '#' or nothing is specified, the new variable will be considered numeric.</p> <p>The logical expressions must be mutually exclusive; i.e., only one may return <i>TRUE</i> for a given row (observation).</p> <p>Any variables referenced must already exist, either as elements of the source data set, as externs, or as the result of a previous make, vector, or code statement.</p> <p>If no default value is specified, 999 is used.</p>	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

code (dataloop)

Example `code agecat default 5 with`
 `1 for age < 21,`
 `2 for age >= 21 and age < 35,`
 `3 for age >= 35 and age < 50,`
 `4 for age >= 50 and age < 65;`
`code $ sex with`
 `"MALE" for gender == 1,`
 `"FEMALE" for gender == 0;`

See also **recode**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

color

Purpose	Set pixel, text, background color, or VGA palette color registers.	
Format	<code>y = color(cv)</code>	
Input	<i>cv</i>	<p>scalar, 2x1 or 3x1 vector of color values or Nx4 matrix of palette color values. See Portability, below, for platform specifics.</p> <p>If the input vector is smaller than 3x1 or the corresponding element in the input vector is -1, the corresponding color will be left unchanged.</p> <p>If the input is an Nx4 matrix, it will initialize the VGA palette (DOS) or active graphic panel's colormap (UNIX) with user-defined RGB colors interpreted as follows:</p> <ul style="list-style-type: none"> [N,1] palette register index 0-255 [N,2] red value 0-63 [N,3] green value 0-63 [N,4] blue value 0-63
Output	<i>y</i>	vector, or Nx4 matrix the same size as the input which contains the original color values or palette values.
Portability	<p>DOS</p> <ul style="list-style-type: none"> [1] pixel color [2] text color [3] ignored <p>UNIX 3.2 only</p> <p>color affects the active graphic panel. X supports foreground and background colors. The color command makes no distinction between text and pixel colors; both affect the foreground color of the active graphic panel. If both a pixel color and text color are specified, the pixel color will be ignored, and the text color will be used to set the foreground color. Thus:</p> <ul style="list-style-type: none"> [1] foreground <p>or</p> <ul style="list-style-type: none"> [1] ignored 	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

color

[2] foreground

or

[1] ignored

[2] foreground

[3] background

OS/2, Windows, UNIX 3.5+

This function is not supported under OS/2 or Windows.

Remarks This changes the window colors for your program’s output. The editor and interactive mode will not be affected.

See “Colors Appendix” on page B-1 for a color value table.

Under DOS, the VGA color palette registers may be set only if the display adapter has been already been initialized to VGA graphics mode 19 (320x200, 256 colors) with the **setvmode** command. The registers will retain the new values until the adapter is reset to text mode, which resets the palette to the default VGA colors.

This function is useful for obtaining 64 shades of a single color and/or mixing colors to user-specification.

See also **graph, line, setvmode**

cols, colsf

Purpose	cols returns the number of columns in a matrix. colsf returns the number of columns in a GAUSS data (.dat) file or GAUSS matrix (.fmt) file.	
Format	$y = \text{cols}(x);$ $yf = \text{colsf}(fh);$	
Input	x any valid expression that returns a matrix. fh file handle of an open file.	
Output	y number of columns in x . yf number of columns in the file that has the handle fh .	
Remarks	If x is an empty matrix, rows (x) and cols (x) return 0. For colsf , the file must be open.	
Example	<pre> x = rndn(100,3); y = cols(x); y = 3.000000 create fp = myfile with x,10,4; b = colsf(fp); b = 10.000000 </pre>	
See also	rows, rowsf, show, lshow	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

comlog

comlog

a	Purpose	Controls logging of interactive mode commands to a disk file.	
b			
c	Format	comlog [[file=filename]] [[on off reset]];	
d	Input	<i>filename</i>	literal or ^string.
e			The file=filename subcommand selects the file to log interactive mode statements to. This can be any legal file name.
f			If the name of the file is to be taken from a string variable, the name of the string must be preceded by the ^ (caret) operator.
g			There is no default file name.
h		on, off, reset	literal, mode command:
i			
j			on turns on command logging to the current log file. If the file already exists, subsequent commands will be appended.
k			off closes the log file and turns off command logging.
l			reset similar to the on subcommand, except that it resets the log file by deleting any previous commands.
m	Remarks	Interactive mode statements are always logged into the file specified in the log_file configuration variable, regardless of the state of comlog .	
n		The command comlog file=filename selects the file but does not turn on logging.	
o		The command comlog off will turn off logging. The filename will remain the same. A subsequent comlog on will cause logging to resume. A subsequent comlog reset will cause the existing contents of the log file to be destroyed and a new file created.	
p		The command comlog by itself will cause the name and status of the current log file to be printed in the window.	
q		In interactive mode under DOS, F10 will load the current log file into the editor if logging is on . If logging is off , the default log file listed in the log_file configuration variable will be loaded into the editor.	
r			
s			
t			
u			
v			
w			
x y z			

compile

Purpose	Compiles a source file to a compiled code file. See also “Compiler” in the <i>User’s Guide</i> .	
Format	compile <i>source fname</i> ;	
Input	<i>source</i>	literal or ^string, the name of the file to be compiled.
	<i>fname</i>	literal or ^string, optional, the name of the file to be created. If not given, the file will have the same filename and path as <i>source</i> . It will have a .gcf extension.
Remarks	<p>The <i>source</i> file will be searched for in the src_path if the full path is not specified and it is not present in the current directory.</p> <p>The source file is a regular DOS text file containing a GAUSS program. There can be references to global symbols, Run-Time Library references, etc.</p> <p>If there are library statements in <i>source</i>, they will be used during the compilation to locate various procedures and symbols used in the program. Since all of these library references are resolved at compile time, the library statements are not transferred to the compiled file. The compiled file can be run without activating any libraries.</p> <p>If you do not want extraneous matter saved in the compiled image, put a new at the top of the <i>source file</i> or execute a new from interactive level before compiling.</p> <p>The program saved in the compiled file can be run with the run command. If no extension is given, the run command will look for a file with the correct extension for the version of GAUSS. The src_path will be used to locate the file if the full path name is not given and it is not located on the current directory.</p> <p>When the compiled file is run, all previous symbols and procedures are deleted before the program is loaded. It is therefore unnecessary to execute a new before run’ning a compiled file.</p> <p>If you want line number records in the compiled file you can put a #lineson statement in the <i>source</i> file or turn line tracking on from the Options menu.</p> <p>Do not try to include compiled files with #include.</p>	

compile

Example `compile qxy.e;`

In this example, the **src_path** would be searched for qxy.e, which would be compiled to a file called qxy.gcg on the same subdirectory qxy.e was found.

`compile qxy.e xy;`

In this example, the **src_path** would be searched for qxy.e which would be compiled to a file called xy.gcg on the current subdirectory.

See also **run, use, saveall**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

complex

Purpose	Converts a pair of real matrices to a complex matrix.	
Format	$z = \text{complex}(x_r, x_i);$	
Input	x_r	NxK real matrix, the real elements of z .
	x_i	NxK real matrix or scalar, the imaginary elements of z .
Output	z	NxK complex matrix.
Example	$x = \begin{Bmatrix} 4 & 6 \\ 9 & 8 \end{Bmatrix};$	
	$y = \begin{Bmatrix} 3 & 5 \\ 1 & 7 \end{Bmatrix};$	
	$t = \text{complex}(x, y);$	
	$t = \begin{matrix} 4.0000000 + 3.0000000i & 6.0000000 + 5.0000000i \\ 9.0000000 + 1.0000000i & 8.0000000 + 7.0000000i \end{matrix}$	
See also	imag , real	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

con

con

Purpose	Requests input from the keyboard, and returns it in a matrix.				
Format	$x = \text{con}(r,c);$				
Input	r	scalar, row dimension of matrix.			
	c	scalar, column dimension of matrix.			
Output	x	RxC matrix.			
Remarks	Enter ? to get a help screen at the con function prompt. The following commands are available:				
	u	Up one row	U	First row	
	d	Down one row	D	Last row	
	l	Left one column	L	First column	
	r	Right one column	R	Last column	
	t	First element			
	b	Last element			
	g #, #	Goto element			
	g #	Goto element of vector			
	h	Move horizontally, default			
	v	Move vertically			
	\	Move diagonally			
	s	Show size of matrix			
	n	Display element as numeric, default			
	c	Display element as character			
	e	exp(1)			
	p	pi			
	.	missing value			
	?	help			

x exit

Use a leading single quote for character input.

See also **cons, let, load**

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

cond

cond

a	Purpose	This procedure will compute the condition number of a matrix using the singular value decomposition.	
b			
c	Format	$c = \text{cond}(x);$	
d			
e	Input	x	NxK matrix.
f	Output	c	scalar, an estimate of the condition number of x . This equals the ratio of the largest singular value to the smallest. If the smallest singular value is zero or not all of the singular values can be computed, the return value is 10^{300} .
g			
h	Example	$x = \begin{Bmatrix} 4 & 2 & 6, \\ 8 & 5 & 7, \\ 3 & 8 & 9 \end{Bmatrix};$ $y = \text{cond}(x);$ $y = 9.8436943$	
i			
j			
k			
l			
m	Source	svd.src	
n			
o			
p			
q			
r			
s			
t			
u			
v			
w			
x y z			

conj

Purpose Returns the complex conjugate of a matrix.

Format `y = conj(x);`

Input *x* N×K matrix.

Output *y* N×K matrix, the complex conjugate of *x*.

Remarks Compare **conj** with the transpose (') operator.

Example `x = { 1+9i 2,
 4+4i 5i,
 7i 8-2i };`
`y = conj(x);`

	$1.0000000 + 9.0000000i$	2.0000000
$x =$	$4.0000000 + 4.0000000i$	$0.0000000 + 5.0000000i$
	$0.0000000 + 7.0000000i$	$8.0000000 - 2.0000000i$
	$1.0000000 - 9.0000000i$	2.0000000
$y =$	$4.0000000 - 4.0000000i$	$0.0000000 - 5.0000000i$
	$0.0000000 - 7.0000000i$	$8.0000000 + 2.0000000i$

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

cons

cons

Purpose	Retrieves a character string from the keyboard.
Format	$x = \text{cons};$
Output	The characters entered from the keyboard. The output will be of type string.
Remarks	If you are working in terminal mode GAUSS will not “see” any input until you press ENTER. x is assigned the value of a character string typed in at the keyboard. The program will pause to accept keyboard input. The maximum length of the string that can be entered is 254 characters. The program will resume execution when the ENTER key is pressed. The standard DOS editing keys will be in effect.
Example	$x = \text{cons};$ At the cursor enter: probability $x = \text{“probability”}$
See also	con

continue

Purpose Jumps to the top of a **do** or **for** loop.

Format **continue;**

Remarks This command works just like in C.

Example

```
x = rndn(4,4);
r = 0;
do while r < rows(x);
    r = r + 1;
    c = 0;
    do while c < cols(x); /* continue jumps here */
        c = c + 1;
        if c == r;
            continue;
        endif;
        x[r,c] = 0;
    endo;
endo;
```

$x =$

-1.032195	0000000	0.000000	0.000000
0.000000	-1.033763	0.000000	0.000000
0.000000	0.000000	0.061205	0.000000
0.000000	0.000000	0.000000	-0.225936

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

contour

contour

Purpose	To graph a matrix of contour data.		
Library	pgraph		
Format	contour (<i>x,y,z</i>) ;		
Input	<i>x</i>	1xK vector, the X axis data. K must be odd.	
	<i>y</i>	Nx1 vector, the Y axis data. N must be odd.	
	<i>z</i>	NxK matrix, the matrix of height data to be plotted.	
Global Input	_plev	Kx1 vector, user-defined contour levels for contour . Default 0.	
	_pzclr	Nx1 or Nx2 vector. This controls the Z level colors. See surface for a complete description of how to set this global.	
Remarks	A vector of evenly spaced contour levels will be generated automatically from the <i>z</i> matrix data. Each contour level will be labeled. For unlabeled contours, use ztics .		
	To specify a vector of your own unequal contour levels, set the vector _plev before calling contour .		
	To specify your own evenly spaced contour levels, see ztics .		
Source	pcontour.src		
See also	surface		

conv

Purpose	Computes the convolution of two vectors.		
Format	$c = \text{conv}(b,x,f,l);$		
Input	b	Nx1 vector.	
	x	Lx1 vector.	
	f	scalar, the first convolution to compute.	
	l	scalar, the last convolution to compute.	
Output	c	Qx1 result, where $Q = (l - f + 1)$.	
		If f is 0, the first to the l 'th convolutions are computed. If l is 0, the f 'th to the last convolutions are computed. If f and l are both zero, all the convolutions are computed.	
Remarks	If x and b are vectors of polynomial coefficients, this is the same as multiplying the two polynomials.		
Example	$x = \{ 1, 2, 3, 4 \};$		
	$y = \{ 5, 6, 7, 8 \};$		
	$z1 = \text{conv}(x,y,0,0);$		
	$z2 = \text{conv}(x,y,2,5);$		
		5	
		16	
		34	
	$z1 =$	60	
		61	
		52	
		32	

conv

$$\begin{matrix} & & 16 \\ & & 34 \\ z2 = & & 60 \\ & & 61 \end{matrix}$$

See also **polymult**

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

coreleft

Purpose	Returns the amount, in bytes, of free workspace memory.
Format	<code>y = coreleft;</code>
Output	<code>y</code> scalar, number of bytes free.
Portability	<p>DOS only</p> <p>All others will return the coreleft value specified in the GAUSS configuration (<code>.cfg</code>) file.</p>
Remarks	<p>The amount of free memory is dynamic and can change rapidly as expressions and procedures are being executed. coreleft returns the amount of workspace memory free at the time it is called. Workspace memory is used for storing matrices, strings, procedures, and for manipulating matrices and strings.</p> <p>This function can be used to write programs that automatically adjust their use of memory so they do not crash with the “Insufficient memory” error if they are used on machines with less free memory than the one used for development, or if the size of the data used becomes larger. A common use is to adjust the number of rows that are read per iteration of a read loop in programs that access data from a disk.</p>
Example	<pre>open fp = myfile; k = colsf(fp); /* columns in file */ fac = 4; /* check amount of memory available */ nr = coreleft/(fac*k*8);</pre> <p>In this example, nr, the number of rows to read, is computed by taking the number of bytes free (coreleft) divided by fac*k*8. fac is a guesstimate of the number of copies of the data read each iteration that the algorithm we are using will require, plus a little. k*8 is the number of columns times the number of bytes per element.</p>
See also	dfree, new

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

corrm, corrv, corrx

Purpose	Computes a correlation matrix.
---------	--------------------------------

Format

```
cx = corrm(m) ;  
cx = corrvc(vc) ;  
cx = corrx(x) ;
```

Input	m	KxK moment ($x'x$) matrix. A constant term MUST have been the first variable when the moment matrix was computed.
	vc	KxK variance-covariance matrix (of data or parameters).
	x	NxK matrix of data.

Output	<i>cx</i>	PxP correlation matrix. For corr_m , $P = K-1$. For corr_{vc} and corr_x , $P = K$.
--------	-----------	--

Source `corr.src`

See also **momentd**

cos

Purpose	Returns the cosine of its argument.
Format	$y = \text{cos}(x);$
Input	x NxK matrix.
Output	y NxK matrix.
Remarks	For real matrices, x should contain angles measured in radians. To convert degrees to radians, multiply the degrees by $\frac{\pi}{180}$.
Example	$x = \{ 0, .5, 1, 1.5 \};$ $y = \text{cos}(x);$ $y = \begin{matrix} 1.00000000 \\ 0.87758256 \\ 0.54030231 \\ 0.07073720 \end{matrix}$
See also	atan, atan2, pi

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

cosh

cosh

a	Purpose	Computes the hyperbolic cosine.
b	Format	$y = \cosh(x);$
c	Input	x NxK matrix.
d	Output	y NxK matrix containing the hyperbolic cosines of the elements of x .
e	Example	$x = \{ -0.5, -0.25, 0, 0.25, 0.5, 1 \};$
f		$x = x * \pi;$
g		$y = \cosh(x);$
h		-1.570796
i		-0.785398
j		0.000000
k		0.785398
l		1.570796
m		3.141593
n		2.509178
o		1.324609
p		1.000000
q		1.324609
r		2.509178
s		11.591953
t	Source	trig.src

counts

Purpose	Count the numbers of elements of a vector that fall into specified ranges.	
Format	$c = \text{counts}(x,v);$	
Input	x	Nx1 vector containing the numbers to be counted.
	v	Px1 vector containing breakpoints specifying the ranges within which counts are to be made. The vector v MUST be sorted in ascending order.
Output	c	Px1 vector, the counts of the elements of x that fall into the regions:
		$\begin{aligned} &x \leq v[1], \\ &v[1] < x \leq v[2], \\ &\cdot \\ &\cdot \\ &\cdot \\ &v[p-1] < x \leq v[p]. \end{aligned}$
Remarks	If the maximum value of x is greater than the last element (the maximum value) of v , the sum of the elements of the result, c , will be less than N , the total number of elements in x .	
	If	
	$x = \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{matrix} \text{ and } v = \begin{matrix} 4 \\ 5 \\ 8 \end{matrix}$	

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

counts

then

$$c = \begin{matrix} 4 \\ 1 \\ 3 \end{matrix}$$

The first category can be a missing value if you need to count missings directly. Also $+\infty$ or $-\infty$ are allowed as breakpoints. The missing value must be the first breakpoint if it is included as a breakpoint and infinities must be in the proper location depending on their sign. $-\infty$ must be in the [2,1] element of the breakpoint vector if there is a missing value as a category as well, otherwise it has to be in the [1,1] element. If $+\infty$ is included, it must be the last element of the breakpoint vector.

Example

```
x = { 1, 3, 2,
      4, 1, 3 };
v = { 0, 1, 2, 3, 4 };

c = counts(x,v);
```

$$c = \begin{matrix} 0.0000000 \\ 2.0000000 \\ 1.0000000 \\ 2.0000000 \\ 1.0000000 \end{matrix}$$

countwts

Purpose	Returns a weighted count of the numbers of elements of a vector that fall into specified ranges.	
Format	$c = \text{countwts}(x, v, w);$	
Input	x	Nx1 vector, the numbers to be counted.
	v	Px1 vector, the breakpoints specifying the ranges within which counts are to be made. This MUST be sorted in ascending order (lowest to highest).
	w	Nx1 vector, containing weights.
Output	c	Px1 vector, the counts of the elements of x that fall into the regions: $\begin{aligned} &x \leq v[1], \\ &v[1] < x \leq v[2], \\ &\cdot \\ &\cdot \\ &v[p-1] < x \leq v[p]. \end{aligned}$
		That is, when $x[i]$ falls into region j , the weight $w[i]$ is added to the j^{th} counter.
Remarks	If any elements of x are greater than the last element of v , they will not be counted.	
	Missing values are not counted unless there is a missing in v . A missing value in v MUST be the first element in v .	

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

countwts

Example `x = { 1, 3, 2, 4, 1, 3 };`
 `w = { .25, 1, .333, .1, .25, 1 };`
 `v = { 0, 1, 2, 3, 4 };`
 `c = countwts(x,v,w);`

 0.000000
 0.500000
`c =` 0.333000
 2.00000
 0.100000

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

create

Purpose	Creates and opens a GAUSS data set for subsequent writing.
Format	<pre>create [[vflag]] [[complex]] fh = filename with vnames,col,dtyp,vtyp; create [[vflag]] [[complex]] fh = filename using comfile;</pre>
Input	<p><i>vflag</i> version flag.</p> <ul style="list-style-type: none"> -v89 supported for read -v92 for read/write -v96 for read/write <p>For details on the various versions, see “File I/O” in the <i>User’s Guide</i>. The default format can be specified in gauss.cfg by setting the dat_fmt_version configuration variable. If dat_fmt_version is not set, the default is v96.</p> <p><i>filename</i> literal or ^string.</p> <p><i>filename</i> is the name to be given the file on the disk. The name can include a path if the directory to be used is not the current directory. This file will automatically be given the extension .dat. If an extension is specified, the .dat will be overridden. If the name of the file is to be taken from a string variable, the name of the string must be preceded by the ^ (caret) operator.</p> <p>create... with...</p> <p><i>vnames</i> literal or ^string or ^character matrix.</p> <p><i>vnames</i> controls the names to be given to the columns of the data file. If the names are to be taken from a string or character matrix, the ^ (caret) operator must be placed before the name of the string or character matrix. The number of columns parameter, <i>col</i>, also has an effect on the way the names will be created. See below and see the examples for details on the ways names are assigned to a data file.</p>

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

create

col scalar expression.
col is a scalar expression containing the number of columns in the data file. If *col* is 0, the number of columns will be controlled by the contents of *vnames*. If *col* is positive, the file will contain *col* columns and the names to be given each column will be created as necessary depending on the *vnames* parameter. See the examples.

dtyp scalar expression.
dtyp is the precision used to store the data. This is a scalar expression containing 2, 4, or 8, which is the number of bytes per element.

2	signed integer
4	single precision
8	double precision

Data Type	Digits	Range
integer	4	-32768 ≤ X ≤ 32767
single	6-7	8.43x10 ⁻³⁷ ≤ X ≤ 3.37x10 ⁺³⁸
double	15-16	4.19x10 ⁻³⁰⁷ ≤ X ≤ 1.67x10 ⁺³⁰⁸

If the integer type is specified, numbers will be rounded to the nearest integer as they are written to the data set. If the data to be written to the file contains character data, the precision must be 8 or the character information will be lost.

vtyp matrix, types of variables.
The types of the variables in the data set. If **rows(vtyp)*cols(vtyp) < col** only the first element is used. Otherwise nonzero elements indicate a numeric variable and zero elements indicate character variables. *vtyp* is ignored for **v89** files.

create... using...

comfile literal or ^string.

comfile is the name of a command file that contains the information needed to create the file. The default extension for the command file is .gcf, which can be overridden.

There are three possible commands in this file:

```
numvar n str;
outvar varlist;
outtyp dtyp;
```

numvar and **outvar** are alternate ways of specifying the number and names of the variables in the data set to be created.

When **numvar** is used, *n* is a constant which specifies the number of variables (columns) in the data file and *str* is a string literal specifying the prefix to be given to all the variables. Thus:

```
numvar 10 xx;
```

says that there are 10 variables and that they are to be named **xx01** through **xx10**. The numeric part of the names will be padded on the left with zeros as necessary so the names will sort correctly:

```
xx1,      ... xx9      1-9 names
xx01,     ... xx10     10-99 names
xx001,    ... xx100    100-999 names
xx0001,   ... xx1000   1000-8100 names
```

If *str* is omitted, the variable prefix will be “X”.

When **outvar** is used, *varlist* is a list of variable names, separated by spaces or commas. For instance:

```
outvar x1, x2, zed;
```

specifies that there are to be 3 variables per row of the data set, and that they are to be named **x1**, **x2**, **zed**, in that order.

outtyp specifies the precision. It can be a constant: 2, 4, or 8, or it can be a literal: I, F, or D. For an explanation of the available data types, see *dtyp* in **create... with...**, previously.

The **outtyp** statement does not have to be included. If it is not, then all data will be stored in 4 bytes as single precision floating point numbers.

create

Output	<i>fh</i> scalar. <i>fh</i> is the file handle which will be used by most commands to refer to the file within GAUSS. This file handle is actually a scalar containing an integer value that uniquely identifies each file. This value is assigned by GAUSS when the create (or open) command is executed.
Remarks	If the complex flag is included, the new data set will be initialized to store complex number data. Complex data is stored a row at a time, with the real and imaginary halves interleaved, element by element.
Example	<pre>let vnames = age sex educat wage occ; create f1 = simdat with ^vnames,0,8; obs = 0; nr = 1000; do while obs < 10000; data = rndn(nr,colsf(f1)); if writer(f1,data) /= nr; print "Disk Full"; end; endif; obs = obs+nr; endo; closeall f1;</pre> <p>uses create... with... to create a double precision data file called simdat.dat on the default drive with 5 columns. The writer command is used to write 10000 rows of Normal random numbers into the file. The variables (columns) will be named: AGE, SEX, EDUCAT, WAGE, OCC.</p> <p>Following are examples of the variable names that will result when using a character vector of names in the argument to the create function.</p>

create

```

vnames = { AGE PAY SEX JOB };
typ = { 1, 1, 0, 0 };
create fp = mydata with ^vnames,0,2,typ;

```

The names in this example will be: **AGE PAY SEX JOB**

AGE and PAY are numeric variables, SEX and JOB are character variables.

```

create fp = mydata with ^vnames,3,2;

```

The names will be: **AGE PAY SEX**

```

create fp = mydata with ^vnames,8,2;

```

The names will now be: **AGE PAY SEX JOB1 JOB2 JOB3 JOB4 JOB5**

If a literal is used for the *vnames* parameter, the number of columns should be explicitly given in the *col* parameter and the names will be created as follows:

```

create fp = mydata with var,4,2;

```

giving the names: **var1 var2 var3 var4**

The next example assumes a command file called *comd.gcf* containing the following lines created using a text editor:

```

outvar age, pay, sex;
outtyp i;

```

Then the following could be used to write 100 rows of random integers into a file called *smpl.dat* in the subdirectory called */gauss/data*:

```

filename = "/gauss/data/smpl";
create fh = ^filename using comd;
x = rndn(100,3)*10;
if writer(fh,x) /= rows(x);
    print Disk Full;
end;
endif;
closeall fh;

```

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

create

For platforms using the backslash as a path separator, remember that two backslashes (\\) are required to enter one backslash inside double quotes. This is because a backslash is the escape character used to embed special characters in strings.

See also **open, readr, writer, eof, close, output, iscplxf**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

crossprd

Purpose	Computes the cross-products (vector products) of sets of 3x1 vectors.	
Format	$z = \text{crossprd}(x,y);$	
Input	x	3xK matrix, each column is treated as a 3x1 vector.
	y	3xK matrix, each column is treated as a 3x1 vector.
Output	z	3xK matrix, each column is the cross-product (sometimes called vector product) of the corresponding columns of x and y .
Remarks	The cross-product vector z is orthogonal to both x and y . sumc ($x.*z$) and sumc ($y.*z$) will be Kx1 vectors all of whose elements are 0 (except for rounding error).	
Example	<pre> x = { 10 4, 11 13, 14 13 }; y = { 3 11, 5 12, 7 9 }; z = crossprd(x,y); 7.0000000 -39.000000 z = -28.000000 107.00000 17.000000 -95.000000 </pre>	
Source	crossprd.src	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

crout

crout

a		
b		
c		
d		
e		
f		
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		
	Purpose	Computes the Crout decomposition of a square matrix without row pivoting, such that: $X = LU$.
	Format	$y = \text{crout}(x);$
	Input	x NxN square nonsingular matrix.
	Output	y NxN matrix containing the lower (L) and upper (U) matrices of the Crout decomposition of x . The main diagonal of y is the main diagonal of the lower matrix L . The upper matrix has an implicit main diagonal of ones. Use lowmat and upmat1 to extract the L and U matrices from y .
	Remarks	Since it does not do row pivoting, it is intended primarily for teaching purposes. (See croutp for a decomposition with pivoting.)
	Example	$x = \begin{Bmatrix} 1 & 2 & -1, \\ 2 & 3 & -2, \\ 1 & -2 & 1 \end{Bmatrix};$ $y = \text{crout}(x);$ $L = \text{lowmat}(y);$ $U = \text{upmat1}(y);$ $y = \begin{Bmatrix} 1 & 2 & -1 \\ 2 & -1 & 0 \\ 1 & -4 & 2 \end{Bmatrix}$ $L = \begin{Bmatrix} 1 & 0 & 0 \\ 2 & -1 & 0 \\ 1 & -4 & 2 \end{Bmatrix}$

crout

$$U = \begin{pmatrix} 1 & 2 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

See also `croutp`, `chol`, `lowmat`, `lowmat1`, `lu`, `upmat`, `upmat1`

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

croutp**croutp**

Purpose	Computes the Crout decomposition of a square matrix with partial (row) pivoting.	
Format	$y = \text{croutp}(x);$	
Input	x	$N \times N$ square nonsingular matrix.
Output	y	$(N+1) \times N$ matrix containing the lower (L) and upper (U) matrices of the Crout decomposition of a permuted x . The $N+1$ row of the matrix y gives the row order of the y matrix. The matrix must be reordered prior to extracting the L and U matrices. Use lowmat and upmat1 to extract the L and U matrices from the reordered y matrix.
Example	This example illustrates a procedure for extracting L and U of the permuted x matrix. It continues by sorting the result of LU to compare with the original matrix x .	

```

X = { 1  2 -1,
      2  3 -2,
      1 -2  1 };
y = croutp(x);

r = rows(y);      /* the number of rows of y */
indx = y[r,.]'; /* get the index vector */
z = y[indx,.]; /* z is indexed R x R matrix y */
L = lowmat(z); /* obtain L and U of permuted */
                /* matrix X */

U = upmat1(z);

q = sortc(indx~(L*U),1); /* sort L*U against */
                        /* index */

x2 = q[.,2:cols(q)]; /* remove index column */

```

$$X = \begin{bmatrix} 1 & 2 & -1 \\ 2 & 3 & -2 \\ 1 & -2 & 1 \end{bmatrix}$$

$$y = \begin{bmatrix} 1 & 0.5 & 0.2857 \\ 2 & 1.5 & -1 \\ 1 & -3.5 & -0.5714 \\ 2 & 3 & 1 \end{bmatrix}$$

$$r = 4$$

$$indx = \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix}$$

$$z = \begin{bmatrix} 2 & 1.5 & -1 \\ 1 & -3.5 & -0.5714 \\ 1 & 0.5 & 0.2857 \end{bmatrix}$$

$$L = \begin{bmatrix} 2 & 0 & 0 \\ 1 & -3.5 & 0 \\ 1 & 0.5 & 0.2857 \end{bmatrix}$$

$$U = \begin{bmatrix} 1 & 1.5 & -1 \\ 0 & 1 & -0.5714 \\ 0 & 0 & 1 \end{bmatrix}$$

$$q = \begin{bmatrix} 1 & 1 & 2 & -1 \\ 2 & 2 & 3 & -2 \\ 3 & 1 & -2 & 1 \end{bmatrix}$$

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

croutp

$$x2 = \begin{pmatrix} 1 & 2 & -1 \\ 2 & 3 & -2 \\ 1 & -2 & 1 \end{pmatrix}$$

See also `crout`, `chol`, `lowmat`, `lowmat1`, `lu`, `upmat`, `upmat1`

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

csrcol, csrlin

Purpose	Returns the position of the cursor.
Format	<pre>y = csrcol; y = csrlin;</pre>
Portability	<p>UNIX 3.2 only</p> <p>csrcol returns the cursor column for the active graphic panel. For Text graphic panels, this value is the cursor column with respect to the text buffer. For TTY graphic panels, this value is the cursor column with respect to the current output line, i.e., it will be the same whether the text is wrapped or not. For PQG graphic panels, this value is meaningless.</p> <p>csrlin returns the cursor line for the active graphic panel. For Text graphic panels, this value is the cursor row with respect to the text buffer. For TTY graphic panels, this value is the current output line number (i.e., the number of lines logged + 1). For PQG graphic panels, this value is meaningless.</p> <p>UNIX 3.5+</p> <p>csrcol and csrlin always return 1.</p> <p>OS/2, Windows</p> <p>csrcol returns the cursor column with respect to the current output line, i.e., it will return the same value whether the text is wrapped or not. csrlin returns the cursor line with respect to the top line in the graphic panel.</p> <p>DOS</p> <p>Under DOS, columns are usually numbered 1-80, rows are usually numbered 1-25. setvmode will return the current window dimensions.</p>
Remarks	<p>y will contain the current column or row position of the cursor in the graphic panel. The upper left corner is (1,1).</p> <p>csrcol returns the column position of the cursor. csrlin returns the row position.</p> <p>The locate statement allows the cursor to be positioned at a specific row and column.</p>
Example	<pre>r = csrlin;</pre>

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

csrcol, csrln

```
c = csrcol;
```

```
cls;
```

```
locate r,c;
```

In this example the window is cleared without affecting the cursor position.

See also **cls, locate, lpos**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

csrtype

Purpose	To set the cursor shape.	
Format	<i>old</i> = csrtype (<i>mode</i>);	
Portability	UNIX This function is not supported in terminal mode. OS/2, Windows This function is not supported under OS/2 or Windows.	
Input	<i>mode</i>	scalar, cursor type to set. DOS 0 cursor off 1 normal cursor 2 large cursor UNIX 3.2 0 cursor off 1 normal cursor 2 large cursor 3 triangular cursor
Output	<i>old</i>	scalar, original cursor type.
Remarks	Under DOS, this function will set the same cursor shape that GAUSS is already using for its three modes.	
Example	<i>x</i> = csrtype (2);	
See also	csrcol , csrlin	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

cumprodc

cumprodc

a	Purpose	Computes the cumulative products of the columns of a matrix.
b	Format	$y = \text{cumprodc}(x);$
c	Input	x NxK matrix.
d	Output	y NxK matrix containing the cumulative products of the columns of x .
e	Remarks	This is based on the recursive series recsercp . recsercp could be called directly as follows: recserp(x, zeros(1,cols(x))) to accomplish the same thing.
f	Example	$x = \begin{Bmatrix} 1 & -3, \\ 2 & 2, \\ 3 & -1 \end{Bmatrix};$ $y = \text{cumprodc}(x);$ $y = \begin{Bmatrix} 1.00 & -3.00 \\ 2.00 & -6.00 \\ 6.00 & 6.00 \end{Bmatrix}$
g	Source	cumprodc.src
h	See also	cumsumc, recsercp, recserar
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

cumsumc

Purpose	Computes the cumulative sums of the columns of a matrix.
Format	$y = \text{cumsumc}(x);$
Input	x NxK matrix.
Output	y NxK matrix containing the cumulative sums of the columns of x .
Remarks	<p>This is based on the recursive series function recserar. recserar could be called directly as follows:</p> <pre>recserar(x,x[1,.], ones(1,cols(x)))</pre> <p>to accomplish the same thing.</p>
Example	<pre>x = { 1 -3, 2 2, 3 -1 }; y = cumsumc(x);</pre> $y = \begin{bmatrix} 1 & -3 \\ 3 & -1 \\ 6 & -2 \end{bmatrix}$
Source	cumsumc.src
See also	cumprodc , recsercp , recserar

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

curve

curve

a	<table><tr><td>Purpose</td><td colspan="2">Computes a one-dimensional smoothing curve.</td></tr><tr><td>Format</td><td colspan="2">{ <i>u</i>, <i>v</i> } = curve(<i>x</i>,<i>y</i>,<i>d</i>,<i>s</i>,<i>sigma</i>,<i>G</i>) ;</td></tr><tr><td>c</td><td rowspan="6">Input</td><td><i>x</i> Kx1 vector, x-abcissae (x-axis values).</td></tr><tr><td>d</td><td><i>y</i> Kx1 vector, y-ordinates (y-axis values).</td></tr><tr><td>e</td><td><i>d</i> Kx1 vector or scalar, observation weights.</td></tr><tr><td>f</td><td><i>s</i> scalar, smoothing parameter. If <i>s</i> = 0, curve performs an interpolation. If <i>d</i> contains standard deviation estimates, a reasonable value for <i>s</i> is K.</td></tr><tr><td>g</td><td><i>sigma</i> scalar, tension factor.</td></tr><tr><td>h</td><td><i>G</i> scalar, grid size factor.</td></tr><tr><td>i</td><td rowspan="2">Output</td><td><i>u</i> K*Gx1 vector, x-abcissae, regularly spaced.</td></tr><tr><td>j</td><td><i>v</i> K*Gx1 vector, y-ordinates, regularly spaced.</td></tr><tr><td>k</td><td rowspan="3">Remarks</td><td><i>sigma</i> contains the tension factor. This value indicates the curviness desired. If <i>sigma</i> is nearly zero (e.g., .001), the resulting curve is approximately the tensor product of cubic curves. If <i>sigma</i> is large, (e.g., 50.0) the resulting curve is approximately bi-linear. If <i>sigma</i> equals zero, tensor products of cubic curves result. A standard value for <i>sigma</i> is approximately 1.</td></tr><tr><td>l</td><td><i>G</i> is the grid size factor. It determines the fineness of the output grid. For <i>G</i> = 1, the input and output vectors will be the same size. For <i>G</i> = 2, the output grid is twice as fine as the input grid, i.e., <i>u</i> and <i>v</i> will have twice as many rows as <i>x</i> and <i>y</i>.</td></tr><tr><td>m</td><td></td></tr><tr><td>n</td><td rowspan="2">Source</td><td>spline.src</td></tr><tr><td>o</td><td></td></tr><tr><td>p</td><td rowspan="2"></td><td></td></tr><tr><td>q</td><td></td></tr><tr><td>r</td><td rowspan="2"></td><td></td></tr><tr><td>s</td><td></td></tr><tr><td>t</td><td rowspan="2"></td><td></td></tr><tr><td>u</td><td></td></tr><tr><td>v</td><td rowspan="3"></td><td></td></tr><tr><td>w</td><td></td></tr><tr><td>x y z</td><td></td></tr></table>		Purpose	Computes a one-dimensional smoothing curve.		Format	{ <i>u</i> , <i>v</i> } = curve (<i>x</i> , <i>y</i> , <i>d</i> , <i>s</i> , <i>sigma</i> , <i>G</i>) ;		c	Input	<i>x</i> Kx1 vector, x-abcissae (x-axis values).	d	<i>y</i> Kx1 vector, y-ordinates (y-axis values).	e	<i>d</i> Kx1 vector or scalar, observation weights.	f	<i>s</i> scalar, smoothing parameter. If <i>s</i> = 0, curve performs an interpolation. If <i>d</i> contains standard deviation estimates, a reasonable value for <i>s</i> is K.	g	<i>sigma</i> scalar, tension factor.	h	<i>G</i> scalar, grid size factor.	i	Output	<i>u</i> K*Gx1 vector, x-abcissae, regularly spaced.	j	<i>v</i> K*Gx1 vector, y-ordinates, regularly spaced.	k	Remarks	<i>sigma</i> contains the tension factor. This value indicates the curviness desired. If <i>sigma</i> is nearly zero (e.g., .001), the resulting curve is approximately the tensor product of cubic curves. If <i>sigma</i> is large, (e.g., 50.0) the resulting curve is approximately bi-linear. If <i>sigma</i> equals zero, tensor products of cubic curves result. A standard value for <i>sigma</i> is approximately 1.	l	<i>G</i> is the grid size factor. It determines the fineness of the output grid. For <i>G</i> = 1, the input and output vectors will be the same size. For <i>G</i> = 2, the output grid is twice as fine as the input grid, i.e., <i>u</i> and <i>v</i> will have twice as many rows as <i>x</i> and <i>y</i> .	m		n	Source	spline.src	o		p			q		r			s		t			u		v			w		x y z	
Purpose	Computes a one-dimensional smoothing curve.																																																											
Format	{ <i>u</i> , <i>v</i> } = curve (<i>x</i> , <i>y</i> , <i>d</i> , <i>s</i> , <i>sigma</i> , <i>G</i>) ;																																																											
c	Input	<i>x</i> Kx1 vector, x-abcissae (x-axis values).																																																										
d		<i>y</i> Kx1 vector, y-ordinates (y-axis values).																																																										
e		<i>d</i> Kx1 vector or scalar, observation weights.																																																										
f		<i>s</i> scalar, smoothing parameter. If <i>s</i> = 0, curve performs an interpolation. If <i>d</i> contains standard deviation estimates, a reasonable value for <i>s</i> is K.																																																										
g		<i>sigma</i> scalar, tension factor.																																																										
h		<i>G</i> scalar, grid size factor.																																																										
i	Output	<i>u</i> K*Gx1 vector, x-abcissae, regularly spaced.																																																										
j		<i>v</i> K*Gx1 vector, y-ordinates, regularly spaced.																																																										
k	Remarks	<i>sigma</i> contains the tension factor. This value indicates the curviness desired. If <i>sigma</i> is nearly zero (e.g., .001), the resulting curve is approximately the tensor product of cubic curves. If <i>sigma</i> is large, (e.g., 50.0) the resulting curve is approximately bi-linear. If <i>sigma</i> equals zero, tensor products of cubic curves result. A standard value for <i>sigma</i> is approximately 1.																																																										
l		<i>G</i> is the grid size factor. It determines the fineness of the output grid. For <i>G</i> = 1, the input and output vectors will be the same size. For <i>G</i> = 2, the output grid is twice as fine as the input grid, i.e., <i>u</i> and <i>v</i> will have twice as many rows as <i>x</i> and <i>y</i> .																																																										
m																																																												
n	Source	spline.src																																																										
o																																																												
p																																																												
q																																																												
r																																																												
s																																																												
t																																																												
u																																																												
v																																																												
w																																																												
x y z																																																												

cvtos

Purpose	Converts a character vector to a string.
Format	<code>s = cvtos(v);</code>
Input	<code>v</code> Nx1 character vector, to be converted to a string.
Output	<code>s</code> string, contains the contents of <code>v</code> .
Remarks	<p>cvtos in effect appends the elements of <code>v</code> together into a single string.</p> <p>cvtos was written to operate in conjunction with stocv. If you pass it a character vector that does not conform to the output of stocv, you may get unexpected results. For example, cvtos DOES NOT look for 0 terminating bytes in the elements of <code>v</code>; it assumes every element except the last is 8 characters long. If this is not true, there will be 0's in the middle of <code>s</code>.</p> <p>If the last element of <code>v</code> does not have a terminating 0 byte, cvtos supplies one for <code>s</code>.</p>
Example	<pre>let v = { "Now is t" "he time " "for all " "good men" }; s = cvtos(v); s = "Now is the time for all good men"</pre>
See also	stocv , vget , vlist , vput , vread

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

datalist

datalist

a	Purpose	List selected variables from a data set.	
b	Format	datalist <i>dataset</i> [<i>var1</i> [<i>var2...</i>]] ;	
c	Input	<i>dataset</i>	literal, name of the dataset.
d		<i>var#</i>	literal, the names of the variables to list.
e	Global Input	_range	global scalar, the range of rows to list. The default is all rows.
f		_miss	global scalar, controls handling of missing values.
g			0 display rows with missing values.
h			1 do not display rows with missing values.
i			The default is 0.
j		_prec	global scalar, the number of digits to the right of the decimal point to display. The default is 3.
k	Remarks	The variables are listed in an interactive mode. As many rows and columns as will fit in the window are displayed. You can use the cursor keys to pan and scroll around in the listing.	
l	Example	<code>datalist freq age sex pay;</code>	
m		This command will display the variables age , sex , and pay from the data set <code>freq.dat</code> .	
n	Source	<code>datalist.src</code>	
o			
p			
q			
r			
s			
t			
u			
v			
w			
x y z			

dataloop (dataloop)

Purpose	Specifies the beginning of a data loop.
Format	dataloop <i>infile</i> <i>outfile</i> ;
Input	<i>infile</i> string variable or literal, the name of the source data set.
Output	<i>outfile</i> string variable or literal, the name of the output data set.
Remarks	The statements between the dataloop... endata commands are assumed to be metacode to be translated at compile time. The data from <i>infile</i> is manipulated by the specified statements, and stored to the data set <i>outfile</i> . Case is not significant within the dataloop... endata section, except for within quoted strings. Comments can be used as in any GAUSS code.
Example	<pre>src = "source"; dataloop ^src dest; make newvar = x1 + x2 + log(x3); x6 = sqrt(x4); keep x6, x5, newvar; endata;</pre> <p>Here, src is a string variable requiring the caret operator (^) , while dest is a string literal.</p>

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

date

date

Purpose	Returns the current date in a 4-element column vector, in the order: year, month, day, and hundredths of a second since midnight.
Format	<code>y = date;</code>
Remarks	The hundredths of a second since midnight can be accessed using hsec .
Example	<pre>print date;</pre> <div><div>1998.0000</div><div>6.0000000</div><div>15.000000</div><div>4011252.7</div></div>
See also	<code>time, timestr, ethsec, hsec, etstr</code>

datestr

Purpose	Returns a date in a string.		
Format	<code>str = datestr(d);</code>		
Input	<i>d</i>	4x1 vector, like the date function returns. If this is 0, the date function will be called for the current system date.	
Output	<i>str</i>	8 character string containing current date in the form: mo/dy/yr	
Example	<pre>d = { 1998, 6, 15, 0 }; y = datestr(d); print y; 6/15/98</pre>		
Source	time.src		
See also	date , datestring , datestrymd , time , timestr , ethsec		

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

datestring

datestring

a	Purpose	Returns a date in a year-2000-compliant string.
b	Format	<code>str = datestring(d);</code>
c	Input	<code>d</code> 4x1 vector, like the date function returns. If this is 0, the date function will be called for the current system date.
d	Output	<code>str</code> 10 character string containing current date in the form: mm/dd/yyyy
e	Example	<code>y = datestring(0);</code> <code>print y;</code> 6/15/1998
f	Source	<code>time.src</code>
g	See also	date, datestr, datestrymd, time, timestr, ethsec
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

datestrymd

Purpose	Returns a date in a string.		
Format	<code>str = datestrymd(d);</code>		
Input	<i>d</i>	4x1 vector, like the date function returns. If this is 0, the date function will be called for the current system date.	
Output	<i>str</i>	8 character string containing current date in the form: yyyymmdd	
Example	<pre>d = { 1998, 6, 15, 0 }; y = datestrymd(d); print y; 19980615</pre>		
Source	time.src		
See also	date, datestr, datestring, time, timestr, ethsec		

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

dayinyr

dayinyr

a	Purpose	Returns day number in the year of a given date.
b	Format	<i>daynum</i> = dayinyr (<i>dt</i>);
c	Input	<i>dt</i> 3x1 or 4x1 vector, date to check. The date should be in the form returned by date .
d	Output	<i>daynum</i> scalar, the day number of that date in that year, 1-366.
e	Example	<i>x</i> = { 1998, 6, 15, 0 }; <i>y</i> = dayinyr(<i>x</i>); print <i>y</i> ; 166.00000
f	Source	time.src
g	Globals	_isleap
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

dayofweek

Purpose Returns day of week.

Format $d = \text{dayofweek}(a);$

Input a Nx1 vector, dates in DT format.

Output d Nx1 vector, integers indicating day of week of each date:

[1]	Sunday
[2]	Monday
[3]	Tuesday
[4]	Wednesday
[5]	Thursday
[6]	Friday
[7]	Saturday

Remarks The DT scalar format is a double precision representation of the date and time. In the DT scalar format, the number
20010421183207
represents 18:32:07 or 6:32:07 PM on April 21, 2001.

Source time.src

debug

debug

Purpose	Runs a program under the source level debugger.
Format	debug <i>filename</i> ;
Input	<i>filename</i> Literal or name of file to debug.
Remarks	See “Debugging” in the <i>User’s Guide</i> .

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

declare

Purpose	To initialize matrices and strings at compile time.		
Format	declare <i>[[type] symbol [[aop clist]]</i> ;		
Input	<i>type</i>	optional literal, specifying the type of the symbol. matrix string if <i>type</i> is not specified, matrix is assumed.	
	<i>symbol</i>	the name of the symbol being declared.	
	<i>aop</i>	the type of assignment to be made. = if not initialized, initialize. if already initialized, reinitialize. != if not initialized, initialize. if already initialized, reinitialize. := if not initialized, initialize. if already initialized, redefinition error. ?= if not initialized, initialize. if already initialized, leave as is. If <i>aop</i> is specified, <i>clist</i> must be also.	
	<i>clist</i>	a list of constants to assign to <i>symbol</i> . If <i>aop clist</i> is not specified, <i>symbol</i> is initialized as a scalar 0 or a null string.	
Remarks	<p>The declare syntax is similar to the let statement.</p> <p>declare generates no executable code. This is strictly for compile time initialization. The data on the right-hand side of the equal sign must be constants. No expressions or variables are allowed.</p> <p>declare statements are intended for initialization of global matrices and strings that are used by procedures in a library system.</p> <p>It is best to place declare statements in a separate file from procedure definitions. This will prevent redefinition errors when rerunning the same program without clearing your workspace.</p> <p>Complex numbers can be entered by joining the real and imaginary parts with a sign (+ or -); there should be no spaces between the numbers and</p>		

declare

the sign. Numbers with no real part can be entered by appending an ‘i’ to the number.

There should be only one declaration for any symbol in a program. Multiple declarations of the same symbol should be considered a programming error. When GAUSS is looking through the library to reconcile a reference to a matrix or a string, it will quit looking as soon as a symbol with the correct name is found. If another symbol with the same name existed in another file, it would never be found. Only the first one in the search path would be available to programs.

Here are some of the possible uses of the three forms of declaration:

!=, = Interactive programming or any situation where a global by the same name will probably be listed in the symbol table when the file containing the **declare** statement is compiled. The symbol will be reset.

This allows mixing **declare** statements with the procedure definitions that reference the global matrices and strings, or placing them in your main file.

:= Redefinition is treated as an error. This will not allow you to assign one symbol with another value already in your program. Rename one of them.

Place **declare** statements in a separate file from the rest of your program and procedure definitions.

?= Interactive programming where some global defaults were set when you started and you do not want them reset for each successive run even if the file containing the **declare**’s gets recompiled. Be careful when using.

CTRL+W controls the **declare** statement warning level. If **declare** warnings are on, you will be warned whenever a **declare** statement encounters a symbol that is already initialized. This happens when you declare a symbol that is already initialized when **declare** warnings are turned on:

declare != Reinitialize and warn.
declare := Crash with fatal error.
declare ?= Leave as is and warn.

If **declare** warnings are off, no warnings are given for the **!=** and **?=** cases.

Example `declare matrix x,y,z;`

declare

```
x = 0
y = 0
z = 0

declare string x = "This string.";
x = "This string."

declare matrix x;
x = 0

declare matrix x != { 1 2 3, 4 5 6, 7 8 9 };

    1 2 3
x =  4 5 6
    7 8 9

declare matrix x[3,3] = 1 2 3 4 5 6 7 8 9;

    1 2 3
x =  4 5 6
    7 8 9

declare matrix x[3,3] = 1;

    1 1 1
x =  1 1 1
    1 1 1

declare matrix x[3,3];

    0 0 0
x =  0 0 0
    0 0 0
```

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

declare

```
declare matrix x = 1 2 3 4 5 6 7 8 9;
```

```
1
```

```
2
```

```
3
```

```
4
```

```
x = 5
```

```
6
```

```
7
```

```
8
```

```
9
```

```
declare matrix x = dog cat;
```

```
x = DOG
```

```
CAT
```

```
declare matrix x = "dog" "cat";
```

```
x = dog
```

```
cat
```

See also **let**, **external**

delete

Purpose	Deletes global symbols from the symbol table.	
Format	delete <i>[-flags]</i> <i>[[symbol]]</i> <i>[[symbol2]]</i> <i>[[symbol3]]</i> ;	
Input	<i>flags</i>	<p>specify the type(s) of symbols to be deleted</p> <ul style="list-style-type: none"> p procedures k keywords f fn functions m matrices s strings g only symbols with global references l only symbols with all local references n no pause for confirmation
	<i>symbol</i>	literal, name of symbol to be deleted. If symbol ends in an asterisk, all symbols matching the leading characters will be deleted.
Remarks	<p>This completely and irrevocably deletes symbols from GAUSS's memory and workspace.</p> <p>Flags must be preceded by a slash (e.g., -pfk). If the n (no pause) flag is used, you will not be asked for confirmation for each symbol.</p> <p>This command is supported only from interactive level. Since the interpreter executes a compiled pseudo-code, this command would invalidate a previously compiled code image and therefore would destroy any program it was a part of. If any symbols are deleted, all procedures, keywords, and functions with global references will be deleted as well.</p>	
Example	<pre>print x; 96.000000 6.0000000 14.000000 3502965.9</pre>	

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

delete

```
delete -m x;
```

At the Delete? [Yes No Previous Quit] prompt, enter y.

```
show x;
```

x no longer exists.

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

delete (dataloop)**delete (dataloop)**

Purpose	Removes specific rows in a data loop based on a logical expression.
Format	delete <i>logical expression</i> ;
Remarks	<p>Deletes only those rows for which <i>logical expression</i> is <i>TRUE</i>. Any variables referenced must already exist, either as elements of the source data set, as externs, or as the result of a previous make, vector, or code statement.</p> <p>GAUSS expects <i>logical expression</i> to return a row vector of 1's and 0's. The relational and other operators (e.g., <) are already interpreted in terms of their dot equivalents (.<), but it is up to the user to make sure that function calls within <i>logical expression</i> result in a vector.</p>
Example	<code>delete age < 40 or sex == 'FEMALE' ;</code>
See also	select

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

delif**delif**

Purpose Deletes rows from a matrix. The rows deleted are those for which there is a 1 in the corresponding row of *e*.

Format `y = delif(x,e);`

Input

<i>x</i>	NxK data matrix.
<i>e</i>	Nx1 logical vector (vector of 0's and 1's).

Output *y* MxK data matrix consisting of the rows of *y* for which there is a 0 in the corresponding row of *e*. If no rows remain, **delif** will return a scalar missing.

Remarks The input *e* will usually be generated by a logical expression using dot operators. For instance:

```
y = delif(x, x[.,2] .> 100);
```

will delete all rows of *x* whose second element is greater than 100. The remaining rows of *x* will be assigned to *y*.

Example

```
x =      { 0 10 20,
           30 40 50,
           60 70 80};
/* logical vector */
```

```
e = (x[.,1] .gt 0) .and (x[.,3] .lt 100);
y = delif(x,e);
```

Here is the resulting matrix *y*:

```
0 10 20
```

All rows for which the elements in column 1 are greater than 0 and the elements in column 3 are less than 100 are deleted.

Source `datatran.src`

delif

See also **selif**

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

denseSubmat

denseSubmat

a	Purpose	Returns dense submatrix of sparse matrix.
b		
c	Format	$e = \text{denseSubmat}(x,r,c);$
d	Input	x MxN sparse matrix.
e		r Kx1 vector, row indices.
f		c Lx1 vector, column indices.
g	Output	e KxL dense matrix.
h	Remarks	If r or c are scalar zeros, all rows or columns will be returned.
i	Source	<code>sparse.src</code>
j	See also	sparseFd, sparseFp
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

design

Purpose	Creates a design matrix of 0's and 1's from a column vector of numbers specifying the columns in which the 1's should be placed.	
Format	<code>y = design(x);</code>	
Input	<i>x</i>	Nx1 vector.
Output	<i>y</i>	NxK matrix, where $K = \text{maxc}(x)$; each row of <i>y</i> will contain a single 1, and the rest 0's. The one in the i^{th} row will be in the <code>round(x[i,1])</code> column.
Remarks	Note that <i>x</i> does not have to contain integers: it will be rounded to nearest if necessary.	
Example	<pre>x = { 1, 1.2, 2, 3, 4.4 }; y = design(x);</pre> $y = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	
Source	design.src	
See also	<code>cumprodc</code> , <code>cumsumc</code> , <code>recserrc</code>	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

det

det

a	Purpose	Returns the determinant of a square matrix.
b	Format	$y = \text{det}(x);$
c	Input	x NxN square matrix.
d	Output	y determinant of x .
e	Remarks	x may be any valid expression that returns a square matrix (number of rows equals number of columns).
f		det computes a LU decomposition.
g		det1 can be much faster in many applications.
h	Example	$x = \begin{Bmatrix} 3 & 2 & 1, \\ 0 & 1 & -2, \\ 1 & 3 & 4 \end{Bmatrix};$
i		$y = \text{det}(x);$
j		$x = \begin{Bmatrix} 3 & 2 & 1 \\ 0 & 1 & -2 \\ 1 & 3 & 4 \end{Bmatrix}$
k		$y = 25$

See also **det1**

det1

- Purpose** Returns the determinant of the last matrix that was passed to one of the intrinsic matrix decomposition routines.
- Format** `y = det1;`
- Remarks** Whenever one of the following functions is executed, the determinant of the matrix is also computed and stored in a system variable. This function will return the value of that determinant and, because the value has been computed in a previous instruction, this will require no computation.

The following functions will set the system variable used by **det1**:

```
chol(x)
crout(x)
croutp(x)
det(x)
inv(x)
invpd(x)
solpd(y,x)  determinant of x
y/x          determinant of x when neither argument is a scalar
or
              determinant of  $x'x$  if  $x$  is not square
```

- Example** If both the inverse and the determinant of the matrix are needed, the following two commands will return both with the minimum amount of computation:

```
xi = inv(x);
xd = det1;
```

The function **det(x)** returns the determinant of a matrix using the Crout decomposition. If you only want the determinant of a positive definite matrix, the following code will be the fastest for matrices larger than 10x10:

```
call chol(x);
xd = det1;
```

det1

The Cholesky decomposition is computed and the result from that is discarded. The determinant saved during that instruction is retrieved using **det1**. This can execute up to 2.5 times faster than **det(x)** for large positive definite matrices.

See also **det**

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

dfft

Purpose	Computes a discrete Fourier transform.	
Format	$y = \text{dfft}(x);$	
Input	x	Nx1 vector.
Output	y	Nx1 vector.
Remarks	<p>The transform is divided by N.</p> <p>This uses a second-order Goertzel algorithm. It is considerably slower than fft, but it may have some advantages in some circumstances. For one thing, N does not have to be an even power of 2.</p>	
Source	dfft.src	
See also	dffti , fft , ffti	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

dfffti

dfffti

a	Purpose	Computes inverse discrete Fourier transform.
b	Format	<code>y = dfffti(x);</code>
c	Input	<code>x</code> Nx1 vector.
d	Output	<code>y</code> Nx1 vector.
e	Remarks	The transform is divided by N . This uses a second-order Goertzel algorithm. It is considerably slower than fffti , but it may have some advantages in some circumstances. For one thing, N does not have to be an even power of 2.
f	Source	<code>dfffti.src</code>
g	See also	fft , dfffti , fffti
h		
i		
j		
k		

dfree (DOS only)**dfree (DOS only)**

Purpose	Returns the amount of room left on a diskette or hard disk.
Format	<i>y</i> = dfree (<i>drive</i>);
Input	<i>drive</i> scalar, valid disk drive number.
Output	<i>y</i> number of bytes free.
Portability	All others return -1
Remarks	Valid disk drive numbers are 0 = default, 1 = A, 2 = B, etc. If an error is encountered, dfree will return -1.
See also	coreleft

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

diag

diag

a		
b		
c		
d	Purpose	Creates a column vector from the diagonal of a matrix.
e	Format	$y = \text{diag}(x);$
f	Input	x NxK matrix.
g	Output	y min(N,K)x1 vector.
h	Remarks	The matrix need not be square. diagrv reverses the procedure and puts a vector into the diagonal of a matrix.
i	Example	$x = \text{rndu}(3,3);$ $y = \text{diag}(x);$ $\begin{matrix} & 0.660818 & 0.367424 & 0.302208 \\ x = & 0.204800 & 0.077357 & 0.145755 \\ & 0.712284 & 0.353760 & 0.642567 \end{matrix}$ $\begin{matrix} y = & 0.660818 \\ & 0.077357 \\ & 0.642567 \end{matrix}$
j	See also	diagrv
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

diagrv

Purpose	Inserts a vector into the diagonal of a matrix.	
Format	<code>y = diagrv(x,v);</code>	
Input	<i>x</i>	NxK matrix.
	<i>v</i>	min(N,K) vector.
Output	<i>y</i>	NxK matrix equal to <i>x</i> with its principal diagonal elements equal to those of <i>v</i> .
Remarks	diag reverses the procedure and pulls the diagonal out of a matrix.	
Example	<code>x = rndu(3,3);</code>	
	<code>v = ones(3,1);</code>	
	<code>y = diagrv(x,v);</code>	
	<div> <div>0.660818 0.367424 0.302208</div> <div><i>x</i> = 0.204800 0.077357 0.145755</div> <div>0.712284 0.353760 0.642567</div> </div> <div> <div>1.000000</div> <div><i>v</i> = 1.000000</div> <div>1.000000</div> </div> <div> <div>1.000000 0.367424 0.302208</div> <div><i>y</i> = 0.204800 1.000000 0.145755</div> <div>0.712284 0.353760 1.000000</div> </div>	
See also	diag	

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

dlibrary

dlibrary

a		
b		
c		
d		
e		
f		
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		
	Purpose	Dynamically links and unlinks shared libraries.
	Format	dlibrary <i>lib1</i> [<i>lib2...</i>]; dlibrary -a <i>lib1</i> [<i>lib2...</i>]; dlibrary -d dlibrary
	Input	<i>lib1 lib2...</i> literal, the base name of the library or the pathed name of the library. dlibrary takes two types of arguments, “base” names and file names. Arguments without any “\” path separators are assumed to be library base names, and are expanded by adding the suffix .dll. They are searched for in the default dynamic library directory. Arguments that include “\” path separators are assumed to be file names, and are not expanded. Relatively pathed file names are assumed to be specified relative to the current working directory, not relative to the dynamic library directory. -a append flag, the DLL’s listed are added to the current set of DLL’s rather than replacing them. For search purposes, the new DLL’s follow the already active ones. Without the -a flag, any previously linked libraries are dumped. -d dump flag, ALL DLL’s are unlinked and the functions they contain are no longer available to your programs. If you use dllcall to call one of your functions after executing a dlibrary -d , your program will terminate with an error.
	Remarks	If no flags are used, the DLL’s listed are linked into GAUSS and any previously linked libraries are dumped. When you call dllcall , the DLL’s will be searched in the order listed for the specified function. The first instance of the function found will be called. dlibrary with no arguments prints out a list of the currently linked DLL’s. The order in which they are listed is the order in which they are searched for functions. dlibrary recognizes a default directory in which to look for dynamic libraries. You can specify this by setting the variable dlib_path in <code>gauss.cfg</code> . Set it to point to a single directory, not a sequence of

dlibrary

directories. A new case (case 24) has also been added to **sysstate** for getting and setting this default.

GAUSS maintains its own DLL, `gauss.dll`. `gauss.dll` is listed when you execute **dlibrary** with no arguments, and searched when you call **dllcall**. By default, `gauss.dll` is searched last, after all other DLL's but you can force it to be searched earlier by listing it explicitly in a **dlibrary** statement. `gauss.dll` is always active. It is not unlinked when you execute **dlibrary -d**. `gauss.dll` is located in the `gauss.exe` directory.

See also **dllcall**, **sysstate-case 24**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

dllcall

dllcall

a		
b		
c		
d		
e		
f		
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		
	Purpose	Calls functions located in dynamic libraries.
	Format	dllcall [-r] [-v] <i>func</i> [(<i>arg1</i> [, <i>arg2</i> ...])]; dllcall works in conjunction with dlibrary . dlibrary is used to link dynamic-link libraries (DLL's) into GAUSS; dllcall is used to access the functions contained in those DLL's. dllcall searches the DLL's (see dlibrary for an explanation of the search order) for a function named <i>func</i> , and calls the first instance it finds. The default DLL, <code>gauss.dll</code> , is searched last.
	Input	<i>func</i> the name of a function contained in a DLL (linked into GAUSS with dlibrary). If <i>func</i> is not specified or cannot be located in a DLL, dllcall will fail. <i>arg#</i> arguments to be passed to <i>func</i> ; optional. These must be elementary variable; they cannot be expressions. -r optional flag. If -r is specified, dllcall examines the value returned by <i>func</i> , and fails if it is nonzero. -v optional flag. Normally, dllcall passes parameters to <i>func</i> in a list. If -v is specified, dllcall passes them in a vector. See below for more details.
	Remarks	<i>func</i> should be written to: 1. Take 0 or more pointers to doubles as arguments. 2. Take arguments either in a list or a vector. 3. Return an integer. In C syntax, <i>func</i> should take one of the following forms: 1. int func (void); 2. int func (double *arg1 [, double *arg2 ...]); 3. int func (double *argv []); dllcall can pass a list of up to 100 arguments to <i>func</i> ; if it requires more arguments than that, you MUST write it to take a vector of arguments, and you MUST specify the -v flag when calling it. dllcall can pass up to 1000 arguments in vector format. In addition, in vector format dllcall appends a null pointer to the vector, so you can write

dllcall

func to take a variable number of arguments and just test for the null pointer.

Arguments are passed to *func* by reference. This means you can send back more than just the return value, which is usually just a success/failure code. (It also means that you need to be careful not to overwrite the contents of matrices or strings you want to preserve.) To return data from *func*, simply set up one or more of its arguments as return matrices (basically, by making them the size of what you intend to return), and inside *func* assign the results to them before returning.

See also **dlibrary, sysstate-case 24**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

do while, do until

do while, do until

a		
b	Purpose	Executes a series of statements in a loop as long as a given expression is true (or false).
c		
d	Format	do while <i>expression</i> ; or do until <i>expression</i> ;
e		
f		.
g		.
h		.
i		<i>statements in loop</i>
j		
k		.
l		.
m		.
n		endo ;
o	Remarks	<i>expression</i> is any expression that returns a scalar. It is TRUE if it is nonzero and FALSE if it is zero.
p		In a do while loop, execution of the loop will continue as long as the expression is TRUE.
q		In a do until loop, execution of the loop will continue as long as the expression is FALSE.
r		The condition is checked at the top of the loop. If execution can continue, the statements of the loop are executed until the endo is encountered. Then GAUSS returns to the top of the loop and checks the condition again.
s		The do loop does not automatically increment a counter. See the first example, following.
t		do loops may be nested.
u		
v		
w		
x y z		

do while, do until

It is often possible to avoid using loops in GAUSS by using the appropriate matrix operator or function. It is almost always preferable to avoid loops when possible, since the corresponding matrix operations can be much faster.

Example

```

format /rdn 1,0;
space = "      ";
comma = ",";
i = 1;
do while i <= 4;
    j = 1;
    do while j <= 3;
        print space i comma j;;
        j = j+1;
    endo;
    i = i+1;
    print;
endo;

1,1  1,2  1,3
2,1  2,2  2,3
3,1  3,2  3,3
4,1  4,2  4,3

```

In the example above, two nested loops are executed and the loop counter values are printed out. Note that the inner loop counter must be reset inside the outer loop before entering the inner loop. An empty **print** statement is used to print a carriage return/line feed sequence after the inner loop finishes.

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

do while, do until

The following are examples of simple loops that execute a predetermined number of times. These loops will both have the result shown.

First loop

```
format /rd 1,0;  
i = 1;  
do while i <= 10;  
print i;;  
i = i+1;  
endo;
```

produces

```
1 2 3 4 5 6 7 8 9 10
```

Second loop

```
format /rd 1,0;  
i = 1;  
do until i > 10;  
print i;;  
i = i+1;  
endo;
```

produces

```
1 2 3 4 5 6 7 8 9 10
```

See also **continue, break**

dos

Purpose	Provides access to the operating system from within GAUSS.
Format	dos [<i>s</i>];
Input	<i>s</i> literal or ^string, the OS command to be executed.
Portability	<p>UNIX</p> <p>Control and output go to the controlling terminal, if there is one. This function may be used in terminal mode.</p> <p>OS/2, Windows</p> <p>The dos function opens a new terminal.</p> <p>Running programs in the background is allowed in all three of the aforementioned platforms.</p>
Remarks	<p>This allows all operating system commands to be used from within GAUSS. It allows other programs to be run even though GAUSS is still resident in memory.</p> <p>If no operating system command (for instance, dir or copy) or program name is specified, a shell of the operating system will be entered which can be used just like the base level OS. The exit command must be given from the shell to get back into GAUSS. If a command or program name is included, the return to GAUSS is automatic after the DOS command has been executed.</p> <p>All matrices are retained in memory when the OS is accessed in this way. This command allows the use of word processing, communications, and other programs from within GAUSS.</p> <p>Do not execute programs that terminate and remain resident because they will be left resident inside GAUSS's workspace. Some examples are programs that create RAM disks or print spoolers.</p> <p>If the command is to be taken from a string variable, the ^ (caret) must precede the string.</p> <p>The shorthand ">" can be used in place of the word "DOS".</p>

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

dos

Example `comstr = "basic myprog";`

`dos ^comstr;`

This will cause the BASIC program **myprog** to be run. When that program is finished, control will automatically return to GAUSS.

`>dir *.prg;`

This will use the DOS **dir** command to print a directory listing of all files with a .prg extension. When the listing is finished, control will be returned to GAUSS.

`dos;`

This will cause a second level OS shell to be entered. The OS prompt will appear and OS commands or other programs can be executed. To return to GAUSS, type **exit**.

See also **exec**

doswin

Purpose	Opens the DOS compatibility window with default settings.
Format	doswin ;
Portability	Windows only
Remarks	Calling doswin is equivalent to: <pre>call DOSWinOpen("",error(0));</pre>
Source	gauss.src

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

DOSWinCloseall

DOSWinCloseall

a	Purpose	Closes the DOS compatibility window.
b	Format	DOSWinCloseall;
c	Portability	Windows only
d	Remarks	Calling DOSWinCloseall closes the DOS window immediately, without asking for confirmation. If a program is running, its I/O reverts to the Command window.
e	Example	<pre>let attr = 50 50 7 0 7; if not DOSWinOpen("Legacy Window", attr); errorlog "Failed to open DOS window, aborting"; stop; endif; . . DOSWinCloseall;</pre>
f		
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

DOSWinOpen

Purpose	Opens the DOS compatibility window and gives it the specified title and attributes.			
Format	ret = DOSWinOpen(<i>title</i>,<i>attr</i>) ;			
Portability	Windows 3.2 only			
Input	<i>title</i>	string, window title.		
	<i>attr</i>	5x1 vector or scalar missing, window attributes.		
	[1]	window x position		
	[2]	window y position		
	[3]	text foreground color		
	[4]	text background color		
	[5]	close action bit flags		
		bit 0 (1's bit)]	issue dialog	
Output	<i>ret</i>	scalar, success flag, 1 if successful, 0 if not.		
	Remarks	If <i>title</i> is a null string (“”), the window will be titled “ GAUSS-DOS ”.		
		Defaults are defined for the elements of <i>attr</i> . To use the default, set an element to a missing value. Set <i>attr</i> to a scalar missing to use all defaults		
		[1]	<i>varies</i>	use x position of previous DOS window
		[2]	<i>varies</i>	use y position of previous DOS window
		[3]	7	white foreground
		[4]	0	black background
		[5]	6	4+2: stop program and close window without confirming
	If the DOS window is already open, the new title and attr will be applied to it. Elements of attr that are missing are not reset to the default values, but are left as is.			
	To set the close action flags value (<i>attr</i> [5]), just sum the desired bit values. For example:			

DOSWinOpen

stop program (4) + close window (2) + confirm close (1) = 7

The close action flags are only relevant when a user attempts to interactively close the DOS window while a program is running. If GAUSS is idle, the window will be closed immediately. Likewise, if a program calls **DOSWinCloseall**, the window is closed, but the program does not get terminated.

Example

```
let attr = 50 50 7 0 7;  
if not DOSWinOpen("Legacy Window", attr);  
    errorlog "Failed to open DOS window, aborting";  
    stop;  
endif;
```

This example opens the DOS window at screen location (50,50), with white text on a black background. The close action flags are $4 + 2 + 1$ (stop program + close window + issue confirm dialog) = 7. Thus, if the user attempts to close the window while a program is running, he/she will be asked for confirmation. Upon confirmation, the window will be closed and the program terminated.

dotfeq, dotfge, dotfgt, dotfle, dotflt, dotfne

dotfeq, dotfge, dotfgt, dotfle, dotflt, dotfne

Purpose	Fuzzy comparison functions. These functions use _fcmptol to fuzz the comparison operations to allow for roundoff error.	
Format	<pre> y = dotfeq(a,b); y = dotfge(a,b); y = dotfgt(a,b); y = dotfle(a,b); y = dotflt(a,b); y = dotfne(a,b); </pre>	
Input	<i>a</i> NxK matrix, first matrix. <i>b</i> LXM matrix, second matrix, EXE compatible with <i>a</i> .	
Global Input	_fcmptol	global scalar, comparison tolerance. The default value is 1.0e-15.
Output	<i>y</i>	max(N,L) by max(K,M) matrix of 1's and 0's.
Remarks	<p>The return value is 1 if true and 0 if false.</p> <p>The statement:</p> <pre>y = dotfeq(a,b);</pre> <p>is equivalent to:</p> <pre>y = a .eq b;</pre> <p>The calling program can reset _fcmptol before calling these procedures.</p> <pre>_fcmptol = 1e-12;</pre>	

dotfeq, dotfge, dotfgt, dotfle, dotflt, dotfne

```
Example  x = rndu(2,2);
          y = rndu(2,2);
          t = dotfge(x,y);

          x = 0.85115559 0.98914218
              0.12703276 0.43365175

          y = 0.41907226 0.49648058
              0.58039125 0.98200340

          t = 1.00000000 1.00000000
              0.00000000 0.00000000
```

Source `fcompare.src`

Globals `_fcmptol`

See also **feq-fne**

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

draw

Purpose	Graphs lines, symbols, and text using the PQG global variables. This procedure does not require actual X, Y, or Z data since its main purpose is to manually build graphs using _pline , _pmsgctl , _psym , _paxes , _parrow and other globals.
Library	pgraph
Format	draw;
Remarks	draw is especially useful when used in conjunction with transparent graphic panels.
Example	<pre> library pgraph; graphset; begwind; /* make full size window for plot */ makewind(9,6.855,0,0,0); /* make small overlapping window for text*/ makewind(3,1,3,3,0); setwind(1); x = seqa(.1,.1,100); y = sin(x); /* plot data in first window*/ xy(x,y); </pre>

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

draw

```
nextwind;  
    _pbox = 15;  
    _paxes = 0;  
    _pnum = 0;  
    _ptitlht = 1;  
margin(0,0,2,0);  
title("This is a text window.");  
draw;  
  
        /* add a smaller text window */  
endwind; /* create graph*/
```

Source `pdraw.src`

See also **window, makewind**

drop (dataloop)

Purpose	Specifies columns to be dropped from the output data set in a data loop.
Format	drop variable_list;
Remarks	<p>Commas are optional in variable_list.</p> <p>Deletes the specified variables from the output data set. Any variables referenced must already exist, either as elements of the source data set, or as the result of a previous make, vector, or code statement.</p> <p>If neither keep nor drop is used, the output data set will contain all variables from the source data set, as well as any defined variables. The effects of multiple keep and drop statements are cumulative.</p>
Example	<code>drop age, pay, sex;</code>
See also	keep

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

dstat

dstat

a	Purpose	Compute descriptive statistics.
b	Format	{ <i>vnam,mean,var,std,min,max,valid,mis</i> } = dstat (<i>dataset,vars</i>);
c	Input	<i>dataset</i> string, name of data set.
d		If <i>dataset</i> contains the name of a GAUSS data set, <i>vars</i> will be interpreted as:
e		<i>vars</i> the variables.
f		If <i>dataset</i> is null or 0, <i>vars</i> will be assumed to be a matrix containing the data.
g		Kx1 character vector names of variables.
h		Kx1 numeric vector indices of columns.
i		These can be any size subset of the variables in the data set and can be in any order. If a scalar 0 is passed, all columns of the data set will be used.
j		If <i>dataset</i> is null or 0, <i>vars</i> will be interpreted as:
k		NxK matrix the data on which to compute the descriptive statistics.
l		
m		
n		Defaults are provided for the following global input variables, so they can be ignored unless you need control over the other options provided by this procedure.
o		
p	__altnam	global matrix, default 0.
q	__miss	This can be a Kx1 character vector of alternate variable names for the output.
r		global scalar, default 0.
s		0 there are no missing values (fastest).
t		1 listwise deletion, drop a row if any missings occur in it.
u		2 pairwise deletion.
v	__row	global scalar, the number of rows to read per iteration of the read loop.
w		if 0, (default) the number of rows will be calculated internally.
x y z		

dstat

	__output	global scalar, controls output, default 1. 1 print output table. 0 do not print output.
Output	<i>vnam</i>	Kx1 character vector, the names of the variables used in the statistics.
	<i>mean</i>	Kx1 vector, means.
	<i>var</i>	Kx1 vector, variance.
	<i>std</i>	Kx1 vector, standard deviation.
	<i>min</i>	Kx1 vector, minima.
	<i>max</i>	Kx1 vector, maxima.
	<i>valid</i>	Kx1 vector, the number of valid cases.
	<i>mis</i>	Kx1 vector, the number of missing cases.
Remarks		If pairwise deletion is used, the minima and maxima will be the true values for the valid data. The means and standard deviations will be computed using the correct number of valid observations for each variable.
Source		<code>dstat.src</code>
Globals		__output, _dstatd, _dstatx

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

dtdate

dtdate

a		
b		
c		
d	Purpose	Creates a matrix in DT scalar format.
e	Format	<i>dt</i> = dtdate (<i>year</i> , <i>month</i> , <i>day</i> , <i>hour</i> , <i>minute</i> , <i>second</i>);
f	Input	<i>year</i> NxK matrix of years. <i>month</i> NxK matrix of months, 1-12. <i>day</i> NxK matrix of days, 1-31. <i>hour</i> NxK matrix of hours, 0-23. <i>minute</i> NxK matrix of minutes, 0-59. <i>second</i> NxK matrix of seconds, 0-59.
g	Output	<i>dt</i> NxK matrix of DT scalar format dates.
h	Remarks	The arguments must be ExE conformable.
i	Source	time.src
j	See also	dtday , dttime , utctodt , dttostr
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

dtday

Purpose	Creates a matrix in DT scalar format containing only the year, month and day. Time of day information is zeroed out.
Format	<i>dt</i> = dtday (<i>year</i> , <i>month</i> , <i>day</i>);
Input	<i>year</i> NxK matrix of years. <i>month</i> NxK matrix of months, 1-12. <i>day</i> NxK matrix of days, 1-31.
Output	<i>dt</i> NxK matrix of DT scalar format dates.
Remarks	This amounts to 00:00:00 or midnight on the given day. The arguments must be ExE conformable.
Source	time.src
See also	dttime , dtdate , utctodt , dttostr

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

datetime

datetime

Purpose	Creates a matrix in DT scalar format containing only the hour, minute and second. The date information is zeroed out.
Format	<i>dt</i> = datetime (<i>hour</i> , <i>minute</i> , <i>second</i>);
Input	<i>hour</i> NxK matrix of hours, 0-23. <i>minute</i> NxK matrix of minutes, 0-59. <i>second</i> NxK matrix of seconds, 0-59.
Output	<i>dt</i> NxK matrix of DT scalar format times.
Remarks	The arguments must be EXE conformable.
Source	time.src
See also	dtday , dtdate , utctodt , dttostr

dttodtv

Purpose	Converts DT scalar format to DTV vector format.
Format	<code>dtv = dttodtv(dt);</code>
Input	<code>dt</code> Nx1 vector, DT scalar format.
Output	<code>dtv</code> Nx8 matrix, DTV vector format.
Remarks	<p>In DT scalar format, 11:06:47 on March 15, 2001 is 20010315110647.</p> <p>Each row of <code>dtv</code>, in DTV vector format, contains:</p> <p>[N,1] Year</p> <p>[N,2] Month in Year, 1-12</p> <p>[N,3] Day of month, 1-31</p> <p>[N,4] Hours since midnight, 0-23</p> <p>[N,5] Minutes, 0-59</p> <p>[N,6] Seconds, 0-59</p> <p>[N,7] Day of week, 0-6, 0 = Sunday</p> <p>[N,8] Days since Jan 1 of current year, 0-365</p>
Example	<pre>dt = 20010326110722; print "dt = " dt; dtv = dttodtv(dt); print "dtv = " dtv;</pre> <p>produces:</p> <pre>dt = 20010326110722 dtv = 2001 3 26 11 7 22 1 84</pre>
Source	<code>time.src</code>
See also	<code>dtvnormal</code> , <code>timeutc</code> , <code>utctodtv</code> , <code>dtvtodt</code> , <code>dttoutc</code> , <code>dtvtodt</code> , <code>strtodt</code> , <code>dttostr</code>

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

dttostr

dttostr

a		
b		
c		
d	Purpose	Converts a matrix containing dates in DT scalar format to a string array.
e	Format	<i>sa</i> = dttostr (<i>x</i> , <i>fmt</i>);
f	Input	<i>x</i> NxK matrix containing dates in DT scalar format. <i>fmt</i> string containing date/time format characters.
g	Output	<i>sa</i> NxK string array.
h	Remarks	The DT scalar format is a double precision representation of the date and time. In the DT scalar format, the number 20010421183207 represents 18:32:07 or 6:32:07 PM on April 21, 2001. dttostr converts a date in DT scalar format to a character string using the format string in <i>fmt</i> . The following formats are supported: YYYY 4 digit year YR Last two digits of year MO Number of month, 01-12 DD Day of month, 01-31 HH Hour of day, 00-23 MI Minute of hour, 00-59 SS Second of minute, 00-59
i	Example	<i>s0</i> = dttostr(utctodt(timeutc), "YYYY-MO-DD HH:MI:SS"); Print ("Date and Time are: " \$+ <i>s0</i>); produces: Date and time are: 2001-03-25 14:59:40
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

dttostr

```
print dttostr(utctodt(timeutc),  
              "Today is DD-MO-YR")
```

produces:

```
Today is 25-03-01
```

```
s = dttostr(x, "YYYY-MO-DD");
```

```
if x = 20000317060424  
        20010427031213  
        20010517020437  
        20011117161422  
        20010717120448  
        20010817043451  
        20010919052320  
        20011017032203  
        20011107071418
```

```
then s = 2000-03-17  
        2001-04-27  
        2001-05-17  
        2001-11-17  
        2001-07-17  
        2001-08-17  
        2001-09-19
```

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

dttostr

2001-10-17

2001-11-07

See also **strtodt, dttoutc, utctodt**

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

dttoutc

Purpose	Converts DT scalar format to UTC scalar format.
Format	<i>utc</i> = dttoutc (<i>dt</i>);
Input	<i>dt</i> Nx1 vector, DT scalar format.
Output	<i>utc</i> Nx1 vector, UTC scalar format.
Remarks	In DT scalar format, 11:06:47 on March 15, 2001 is 20010315110647. A UTC scalar gives the number of seconds since or before January 1, 1970 Greenwich Mean Time.
Example	<pre>dt = 20010326085118; tc = dttoutc(dt); print "utc = " utc;</pre> <p>produces:</p> <pre>tc = 985633642;</pre>
Source	time.src
See also	dtvnormal , timeutc , utctodtv , dttodtv , dtvtodt , dtvtoutc , dtvtodt , strtodt , dttostr

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

dtvnormal

dtvnormal

a		
b		
c		
d		
e		
f		
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		
	Purpose	Normalizes a date and time (DTV) vector.
	Format	<i>d</i> = dtvnormal (<i>t</i>);
	Input	<i>t</i> 1x8 date and time vector that has one or more elements outside the normal range.
	Output	<i>d</i> Normalized 1x8 date and time vector.
	Remarks	The date and time vector is a 1x8 vector whose elements consist of: Year: Year, four digit integer. Month: 1-12, Month in year. Day: 1-31, Day of month. Hour: 0-23, Hours since midnight. Min: 0-59, Minutes. Sec: 0-59, Seconds. DoW: 0-6, Day of week, 0 = Sunday. DiY: 0-365, Days since Jan 1 of year. The last two elements are ignored on input.
	Example	<pre>format /rd 10,2; x = { 1996 14 21 6 21 37 0 0 }; d = dtvnormal(x); d:97.00 2.00 21.00 6.00 21.00 37.00 2.00 51.00</pre>
	See also	date, ethsec, etstr, time, timestr, timeutc, utctodtv

dtvtodt

Purpose	Converts DT vector format to DT scalar format.
Format	<i>dt</i> = dtvtodt (<i>dtv</i>);
Input	<i>dtv</i> Nx8 matrix, DTV vector format.
Output	<i>dt</i> Nx1 vector, DT scalar format.
Remarks	<p>In DT scalar format, 11:06:47 on March 15, 2001 is 20010315110647.</p> <p>Each row of <i>dtv</i>, in DTV vector format, contains:</p> <p>[N,1] Year</p> <p>[N,2] Month in Year, 1-12</p> <p>[N,3] Day of month, 1-31</p> <p>[N,4] Hours since midnight, 0-23</p> <p>[N,5] Minutes, 0-59</p> <p>[N,6] Seconds, 0-59</p> <p>[N,7] Day of week, 0-6, 0 = Sunday</p> <p>[N,8] Days since Jan 1 of current year, 0-365</p>
Example	<pre>let dtv = { 2001 3 26 11 7 22 1 84 }; print "dtv = " dtv; dt = dtvtodt(dtv); print "dt = " dt;</pre> <p>produces:</p> <pre>dtv = 2001 3 26 11 7 22 1 84; dt = 20010326110722</pre>
Source	time.src
See also	dtvnormal , timeutc , utctodtv , dttodtv , dtvtodt , dttoutc , dtvtodt , strtodt , dttostr

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

dtvtoutc

dtvtoutc

a	Purpose	Converts DTV vector format to UTC scalar format.
b	Format	<i>utc</i> = dtvtoutc (<i>dtv</i>) ;
c	Input	<i>dtv</i> Nx8 matrix, DTV vector format.
d	Output	<i>utc</i> Nx1 vector, UTC scalar format.
e	Remarks	A UTC scalar gives the number of seconds since or before January 1, 1970 Greenwich Mean Time.
f		Each row of <i>dtv</i> , in DTV vector format, contains:
g		[N,1] Year
h		[N,2] Month in Year, 1-12
i		[N,3] Day of month, 1-31
j		[N,4] Hours since midnight, 0-23
k		[N,5] Minutes, 0-59
l		[N,6] Seconds, 0-59
m		[N,7] Day of week, 0-6, 0 = Sunday
n		[N,8] Days since Jan 1 of current year, 0-365
o	Example	<pre>dtv = utctodtv(timeutc); print "dtv = " dtv; utc = dtvtoutc(dtv); print "utc = " utc;</pre> <p>produces:</p> <pre>dtv = 2001 3 26 11 7 22 1 utc = 84985633642</pre>
p	See also	dtvnormal, timeutc, utctodt, dttodtv, dttoutc, dtvtodt, dtvtoutc, strtodt, dttostr
q		
r		
s		
t		
u		
v		
w		
x y z		

dummy

Purpose Creates a set of dummy (0/1) variables by breaking up a variable into specified categories. The highest (rightmost) category is unbounded on the right.

Format `y = dummy(x,v);`

Input

x Nx1 vector of data that is to be broken up into dummy variables.

v (K-1)x1 vector specifying the K-1 breakpoints (these must be in ascending order) that determine the K categories to be used. These categories should not overlap.

Output *y* NxK matrix containing the K dummy variables.

Remarks Missings are deleted before the dummy variables are created.

All categories are open on the left (i.e., do not contain their left boundaries) and all but the highest are closed on the right (i.e., do contain their right boundaries). The highest (rightmost) category is unbounded on the right. Thus, only K-1 breakpoints are required to specify K dummy variables.

The function **dummybr** is similar to **dummy**, but in that function the highest category is bounded on the right. The function **dummydn** is also similar to **dummy**, but in that function a specified column of dummies is dropped.

Example

```
x = { 0, 2, 4, 6 };
v = { 1, 5, 7 };
y = dummy(x,v);
```

The result *y* looks like this:

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

dummy

The vector *v* will produce 4 dummies satisfying the following conditions:

$$x \leq 1$$

$$1 < x \leq 5$$

$$5 < x \leq 7$$

$$7 < x$$

Source `datatran.src`

See also `dummybr`, `dummydn`

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

dummybr

Purpose	Creates a set of dummy (0/1) variables. The highest (rightmost) category is bounded on the right.	
Format	$y = \text{dummybr}(x, v);$	
Input	x	$N \times 1$ vector of data that is to be broken up into dummy variables.
	v	$K \times 1$ vector specifying the K breakpoints (these must be in ascending order) that determine the K categories to be used. These categories should not overlap.
Output	y	$N \times K$ matrix containing the K dummy variables. Each row will have a maximum of one 1.
Remarks	Missings are deleted before the dummy variables are created.	
	All categories are open on the left (i.e., do not contain their left boundaries) and are closed on the right (i.e., do contain their right boundaries). Thus, K breakpoints are required to specify K dummy variables.	
	The function dummy is similar to dummybr , but in that function the highest category is unbounded on the right.	
Example	$x = \begin{Bmatrix} 0, \\ 2, \\ 4, \\ 6 \end{Bmatrix};$ $v = \begin{Bmatrix} 1, \\ 5, \\ 7 \end{Bmatrix};$ $y = \text{dummybr}(x, v);$ <p>The resulting matrix y looks like this:</p>	

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

dummybr

		1 0 0
a		0 1 0
b		0 1 0
c		0 0 1
d	The vector $v = 1\ 5\ 7$ will produce 3 dummies satisfying the following conditions:	
e		$x \leq 1$
f		$1 < x \leq 5$
g		$5 < x \leq 7$
h	Source	datatran.src
i	See also	dummydn , dummy
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

dummydn

Purpose	Creates a set of dummy (0/1) variables by breaking up a variable into specified categories. The highest (rightmost) category is unbounded on the right, and a specified column of dummies is dropped.		
Format	$y = \text{dummydn}(x, v, p);$		
Input	x	Nx1 vector of data to be broken up into dummy variables.	
	v	Kx1 vector specifying the K-1 breakpoints (these must be in ascending order) that determine the K categories to be used. These categories should not overlap.	
	p	positive integer in the range [1,K], specifying which column should be dropped in the matrix of dummy variables.	
Output	y	Nx(K-1) matrix containing the K-1 dummy variables.	
Remarks	This is just like the function dummy , except that the p^{th} column of the matrix of dummies is dropped. This ensures that the columns of the matrix of dummies do not sum to 1, and so these variables will not be collinear with a vector of ones.		
	Missings are deleted before the dummy variables are created.		
	All categories are open on the left (i.e., do not contain their left boundaries) and all but the highest are closed on the right (i.e., do contain their right boundaries). The highest (rightmost) category is unbounded on the right. Thus, only K-1 breakpoints are required to specify K dummy variables.		
Example	$x = \{ 0, 2, 4, 6 \};$ $v = \{ 1, 5, 7 \};$ $p = 2;$ $y = \text{dummydn}(x, v, p);$		

dummydn

The resulting matrix y looks like this:

1 0 0
0 0 0
0 0 0
0 1 0

The vector $v = 1\ 5\ 7$ will produce 4 dummies satisfying the following conditions:

$x \leq 1$
 $1 < x \leq 5$
 $5 < x \leq 7$
 $7 < x$

Source `datatran.src`

See also **dummy**, **dummybr**

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

ed

Purpose	Accesses an alternate editor.
Format	ed <i>filename</i> ;
Input	<i>filename</i> The name of the file to be edited.
Remarks	<p>The default name of the editor is set in <code>gauss.cfg</code>. To change the name of the editor used type:</p> <pre>ed = editor_name flags ;</pre> <p>or</p> <pre>ed = "editor_name flags" ;</pre> <p>The flags are any command line flags you may want between the name of the editor and the filename when your editor is invoked. The quoted version will prevent the flags, if any, from being forced to uppercase.</p> <p>This command can be placed in the startup file so it will be set for you automatically when you start GAUSS.</p>

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

edit

edit

a	Purpose	Edits a disk file.
b	Format	edit <i>filename</i> ;
c	Remarks	The edit command does not follow the src_path to locate files. You must specify the location in the filename. The default location is the current directory.
d	Example	edit test1.e;
e	See also	run
f		
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

eig

Purpose	Computes the eigenvalues of a general matrix.
Format	<code>va = eig(x);</code>
Input	<i>x</i> NxN matrix.
Output	<i>va</i> Nx1 vector, the eigenvalues of <i>x</i> .
Remarks	<p>If the eigenvalues cannot all be determined, <i>va</i>[1] is set to an error code. Passing <i>va</i>[1] to the scalerr function will return the index of the eigenvalue that failed. The eigenvalues for indices scalerr(<i>va</i>[1])+1 to N should be correct.</p> <p>Error handling is controlled with the low bit of the trap flag.</p> <p>trap 0 set <i>va</i>[1] and terminate with message</p> <p>trap 1 set <i>va</i>[1] and continue execution</p> <p>The eigenvalues are unordered except that complex conjugate pairs of eigenvalues will appear consecutively with the eigenvalue having the positive imaginary part first.</p>
Example	<pre> x = { 4 8 1 , 9 4 2 , 5 5 7 }; va = eig(x); -4.4979246 va = 14.475702 5.0222223 </pre>
See also	eigh, eighv, eigv

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

eigcg

eigcg

a		
b	Purpose	Computes the eigenvalues of a complex, general matrix. (Included for backwards compatibility — use eig instead.)
c	Format	<code>{ var,vai } = eigcg(xr,xi);</code>
d	Input	<div><div><code>xr</code></div><div>NxN matrix, real part.</div></div> <div><div><code>xi</code></div><div>NxN matrix, imaginary part.</div></div>
e	Output	<div><div><code>var</code></div><div>Nx1 vector, real part of eigenvalues.</div></div> <div><div><code>vai</code></div><div>Nx1 vector, imaginary part of eigenvalues.</div></div> <div><div><code>_eigerr</code></div><div>global scalar, if all the eigenvalues can be determined <code>_eigerr</code> = 0, otherwise <code>_eigerr</code> is set to the index of the eigenvalue that failed. The eigenvalues for indices <code>_eigerr</code>+1 to N should be correct.</div></div>
f	Remarks	<div>Error handling is controlled with the low bit of the trap flag.</div> <div><code>trap 0</code> set <code>_eigerr</code> and terminate with message</div> <div><code>trap 1</code> set <code>_eigerr</code> and continue execution</div> <div>The eigenvalues are unordered except that complex conjugate pairs of eigenvalues will appear consecutively with the eigenvalue having the positive imaginary part first.</div>
g		
h		
i		
j		
k		
l		
m	Source	<code>eigcg.src</code>
n	Globals	<code>_eigerr</code>
o	See also	<code>eigcg2</code> , <code>eigch</code> , <code>eigrg</code> , <code>eigrs</code>
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

eigcg2

Purpose	Computes eigenvalues and eigenvectors of a complex, general matrix. (Included for backwards compatibility — use eigv instead.)	
Format	{ <i>var,vai,ver,vei</i> } = eigcg2 (<i>xr,xi</i>);	
Input	<i>xr</i>	NxN matrix, real part.
	<i>xi</i>	NxN matrix, imaginary part.
Output	<i>var</i>	Nx1 vector, real part of eigenvalues.
	<i>vai</i>	Nx1 vector, imaginary part of eigenvalues.
	<i>ver</i>	NxN matrix, real part of eigenvectors.
	<i>vei</i>	NxN matrix, imaginary part of eigenvectors.
Global Input	_eigerr	global scalar, if all the eigenvalues can be determined _eigerr = 0, otherwise _eigerr is set to the index of the eigenvalue that failed. The eigenvalues for indices _eigerr +1 to N should be correct. The eigenvectors are not computed.
Remarks	<p>Error handling is controlled with the low bit of the trap flag.</p> <p>trap 0 set _eigerr and terminate with message</p> <p>trap 1 set _eigerr and continue execution</p> <p>The eigenvalues are unordered except that complex conjugate pairs of eigenvalues will appear consecutively with the eigenvalue having the positive imaginary part first. The columns of <i>ver</i> and <i>vei</i> contain the real and imaginary eigenvectors of <i>x</i> in the same order as the eigenvalues. The eigenvectors are not normalized.</p>	
Source	eigcg.src	
Globals	_eigerr	
See also	eigcg, eigch, eigrg, eigrs	

eigch

eigch

a	Purpose	Computes the eigenvalues of a complex, hermitian matrix. (Included for backwards compatibility — use eigh instead.)
b	Format	<code>va = eigch(xr,xi);</code>
c	Input	<code>xr</code> NxN matrix, real part. <code>xi</code> NxN matrix, imaginary part.
d	Output	<code>va</code> Nx1 vector, real part of eigenvalues. <code>_eigerr</code> global scalar, if all the eigenvalues can be determined <code>_eigerr</code> = 0, otherwise <code>_eigerr</code> is set to the index of the eigenvalue that failed. The eigenvalues for indices 1 to <code>_eigerr</code> -1 should be correct.
e	Remarks	Error handling is controlled with the low bit of the trap flag. trap 0 set <code>_eigerr</code> and terminate with message trap 1 set <code>_eigerr</code> and continue execution The eigenvalues are in ascending order. The eigenvalues for a complex hermitian matrix are always real so this procedure returns only one vector.
f	Source	<code>eigch.src</code>
g	Globals	<code>_eigerr</code>
h	See also	eigch2, eigcg, eigrg, eigrs
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

eigch2

Purpose	Computes eigenvalues and eigenvectors of a complex, hermitian matrix. (Included for backwards compatibility — use eighv instead.)		
Format	{ <i>var,vai,ver,vei</i> } = eigch2 (<i>xr,xi</i>);		
Input	<i>xr</i>	NxN matrix, real part.	
	<i>xi</i>	NxN matrix, imaginary part.	
Output	<i>var</i>	Nx1 vector, real part of eigenvalues.	
	<i>vai</i>	Nx1 vector, imaginary part of eigenvalues.	
	<i>ver</i>	NxN matrix, real part of eigenvectors.	
	<i>vei</i>	NxN matrix, imaginary part of eigenvectors.	
	_eigerr	global scalar, if all the eigenvalues can be determined _eigerr = 0, otherwise _eigerr is set to the index of the eigenvalue that failed. The eigenvalues for indices 1 to _eigerr -1 should be correct. The eigenvectors are not computed.	
Remarks	Error handling is controlled with the low bit of the trap flag.		
	trap 0 set _eigerr and terminate with message		
	trap 1 set _eigerr and continue execution		
	The eigenvalues are in ascending order. The eigenvalues of a complex hermitian matrix are always real. This procedure returns a vector of zeros for the imaginary part of the eigenvalues so the syntax is consistent with other eigxx procedure calls. The columns of <i>ver</i> and <i>vei</i> contain the real and imaginary eigenvectors of <i>x</i> in the same order as the eigenvalues. The eigenvectors are orthonormal.		
Source	eigch.src		
Globals	_eigerr		
See also	eigch, eigcg, eigrg, eigrs		

eigh

eigh

Purpose	Computes the eigenvalues of a complex hermitian or real symmetric matrix.		
Format	$va = \text{eigh}(x);$		
Input	x	NxN matrix.	
Output	va	Nx1 vector, the eigenvalues of x .	
Remarks	<p>If the eigenvalues cannot all be determined, $va[1]$ is set to an error code. Passing $va[1]$ to the scalerr function will return the index of the eigenvalue that failed. The eigenvalues for indices 1 to scalerr($va[1]$)-1 should be correct.</p> <p>Error handling is controlled with the low bit of the trap flag.</p> <p>trap 0 set $va[1]$ and terminate with message</p> <p>trap 1 set $va[1]$ and continue execution</p> <p>The eigenvalues are in ascending order.</p> <p>The eigenvalues of a complex hermitian or real symmetric matrix are always real.</p>		
See also	eig, eighv, eigv		

eighv

Purpose	Computes eigenvalues and eigenvectors of a complex hermitian or real symmetric matrix.	
Format	<code>{ va,ve } = eighv(x);</code>	
Input	<i>x</i>	NxN matrix.
Output	<i>va</i>	Nx1 vector, the eigenvalues of <i>x</i> .
	<i>ve</i>	NxN matrix, the eigenvectors of <i>x</i> .
Remarks	<p>If the eigenvalues cannot all be determined, <i>va</i>[1] is set to an error code. Passing <i>va</i>[1] to the scalerr function will return the index of the eigenvalue that failed. The eigenvalues for indices 1 to scalerr(<i>va</i>[1])-1 should be correct. The eigenvectors are not computed.</p> <p>Error handling is controlled with the low bit of the trap flag.</p> <p>trap 0 set <i>va</i>[1] and terminate with message</p> <p>trap 1 set <i>va</i>[1] and continue execution</p> <p>The eigenvalues are in ascending order. The columns of <i>ve</i> contain the eigenvectors of <i>x</i> in the same order as the eigenvalues. The eigenvectors are orthonormal.</p> <p>The eigenvalues of a complex hermitian or real symmetric matrix are always real.</p>	
See also	eig , eigh , eigv	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

eigr

eigr

a		
b		
c		
d		
e		
f		
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		
	Purpose	Computes the eigenvalues of a real, general matrix. (Included for backwards compatibility — use eig instead.)
	Format	<code>{ var,vai } = eigr(x);</code>
	Input	<i>x</i> NxN matrix.
	Output	<i>var</i> Nx1 vector, real part of eigenvalues. <i>vai</i> Nx1 vector, imaginary part of eigenvalues. _eigerr global scalar, if all the eigenvalues can be determined _eigerr = 0, otherwise _eigerr is set to the index of the eigenvalue that failed. The eigenvalues for indices _eigerr +1 to N should be correct.
	Remarks	Error handling is controlled with the low bit of the trap flag. trap 0 set _eigerr and terminate with message trap 1 set _eigerr and continue execution The eigenvalues are unordered except that complex conjugate pairs of eigenvalues will appear consecutively with the eigenvalue having the positive imaginary part first.
	Example	<code>x = { 1 2i 3, 4i 5+3i 6, 7 8 9i };</code> <code>{y,n} = eigr(x);</code> -6.3836054 <code>y = 2.0816489</code> 10.301956 7.2292503 <code>n = -1.4598755</code> 6.2306252

eigr

Source `eigrg.src`

Globals `_eigerr`

See also **`eigr2`**, **`eigcg`**, **`eigch`**, **`eigrs`**

[a](#)[b](#)[c](#)[d](#)[e](#)[f](#)[g](#)[h](#)[i](#)[j](#)[k](#)[l](#)[m](#)[n](#)[o](#)[p](#)[q](#)[r](#)[s](#)[t](#)[u](#)[v](#)[w](#)[x y z](#)

eigr2

eigr2

a	Purpose	Computes eigenvalues and eigenvectors of a real, general matrix. (Included for backwards compatibility — use eigv instead.)	
b			
c	Format	{ <i>var,vai,ver,vei</i> } = eigr2 (<i>x</i>);	
d			
e	Input	<i>x</i>	NxN matrix.
f	Output	<i>var</i>	Nx1 vector, real part of eigenvalues.
g		<i>vai</i>	Nx1 vector, imaginary part of eigenvalues.
h		<i>ver</i>	NxN matrix, real part of eigenvectors.
i		<i>vei</i>	NxN matrix, imaginary part of eigenvectors.
j		_eigerr	global scalar, if all the eigenvalues can be determined
k			_eigerr = 0, otherwise _eigerr is set to the index of the eigenvalue that failed. The eigenvalues for indices _eigerr +1 to N should be correct. The eigenvectors are not computed.
l	Remarks	Error handling is controlled with the low bit of the trap flag.	
m		trap 0 set _eigerr and terminate with message	
n		trap 1 set _eigerr and continue execution	
o		The eigenvalues are unordered except that complex conjugate pairs of	
p		eigenvalues will appear consecutively with the eigenvalue having the	
q		positive imaginary part first. The columns of <i>ver</i> and <i>vei</i> contain the real and imaginary eigenvectors of <i>x</i> in the same order as the eigenvalues. The eigenvectors are not normalized.	
r	Source	eigr2.src	
s	Globals	_eigerr	
t	See also	eigr2, eigcg, eigch, eigrs	
u			
v			
w			
x y z			

eigrs

Purpose	Computes the eigenvalues of a real, symmetric matrix. (Included for backwards compatibility — use eigh instead.)	
Format	<code>va = eigrs(x);</code>	
Input	<i>x</i>	NxN matrix.
Output	<i>va</i>	Nx1 vector, eigenvalues of <i>x</i> .
	_eigerr	global scalar, if all the eigenvalues can be determined _eigerr = 0, otherwise _eigerr is set to the index of the eigenvalue that failed. The eigenvalues for indices 1 to _eigerr -1 should be correct.
Remarks	<p>Error handling is controlled with the low bit of the trap flag.</p> <p>trap 0 set _eigerr and terminate with message</p> <p>trap 1 set _eigerr and continue execution</p> <p>The eigenvalues are in ascending order. The eigenvalues for a real symmetric matrix are always real so this procedure returns only one vector.</p>	
Source	<code>eigrs.src</code>	
Globals	_eigerr	
See also	eigrs2, eigcg, eigch, eigr	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

eigrs2

eigrs2

a	Purpose	Computes eigenvalues and eigenvectors of a real, symmetric matrix. (Included for backwards compatibility — use eighv instead.)
b	Format	<code>{ va,ve } = eigrs2(x);</code>
c	Input	<i>x</i> NxN matrix.
d	Output	<i>va</i> Nx1 vector, eigenvalues of <i>x</i> . <i>ve</i> NxN matrix, eigenvectors of <i>x</i> . _eigerr global scalar, if all the eigenvalues can be determined _eigerr = 0, otherwise _eigerr is set to the index of the eigenvalue that failed. The eigenvalues and eigenvectors for indices 1 to _eigerr -1 should be correct.
e	Remarks	Error handling is controlled with the low bit of the trap flag. trap 0 set _eigerr and terminate with message trap 1 set _eigerr and continue execution The eigenvalues are in ascending order. The columns of <i>ve</i> contain the eigenvectors of <i>x</i> in the same order as the eigenvalues. The eigenvectors are orthonormal. The eigenvalues and eigenvectors for a real symmetric matrix are always real so this procedure returns only the real parts.
f	Source	<code>eigrs.src</code>
g	Globals	_eigerr
h	See also	eigrs, eigcg, eigch, eigrg
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

eigv

Purpose	Computes eigenvalues and eigenvectors of a general matrix.		
Format	{ va,ve } = eigv(x);		
Input	x	NxN matrix.	
Output	va	Nx1 vector, the eigenvalues of x.	
	ve	NxN matrix, the eigenvectors of x.	
Remarks	If the eigenvalues cannot all be determined, va[1] is set to an error code. Passing va[1] to the scalerr function will return the index of the eigenvalue that failed. The eigenvalues for indices scalerr (va[1])+1 to N should be correct. The eigenvectors are not computed.		
	Error handling is controlled with the low bit of the trap flag.		
	trap 0 set va[1] and terminate with message		
	trap 1 set va[1] and continue execution		
	The eigenvalues are unordered except that complex conjugate pairs of eigenvalues will appear consecutively with the eigenvalue having the positive imaginary part first. The columns of ve contain the eigenvectors of x in the same order as the eigenvalues. The eigenvectors are not normalized.		
Example	<pre>x = { 4 8 1, 9 4 2, 5 5 7 }; {y,n} = eigv(x); -4.4979246 y = 14.475702 5.0222223 -0.66930459 -0.64076622 -0.40145623 n = 0.71335708 -0.72488533 -0.26047487 -0.01915672 -0.91339349 1.6734214</pre>		

eigv

See also **eig, eigh, eighv**

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

elapsedTradingDays

Purpose	Compute number of trading days between two dates inclusively.	
Format	$n = \text{elapsedTradingDays}(a, b);$	
Input	a	scalar, date in DT scalar format.
	b	scalar, date in DT scalar format.
Output	n	number of trading days between dates inclusively, that is, elapsed time includes the dates a and b .
Remarks	A trading day is a weekday that is not a holiday as defined by the New York Stock Exchange from 1888 through 2004. Holidays are defined in <code>holidays.asc</code> . You may edit that file to modify or add holidays.	
Source	<code>finutils.src</code>	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

end

end

a		
b	Purpose	Terminates a program.
c	Format	end;
d	Remarks	end causes GAUSS to revert to interactive mode, and closes all open files. end also closes the auxiliary output file and turns the screen on. It is not necessary to put an end statement at the end of a program.
e		An end command can be placed above a label which begins a subroutine to make sure that a program does not enter a subroutine without a gosub .
f		stop also terminates a program but closes no files and leaves the screen setting as it is.
g		
h		
i	Example	output on; screen off; print x; end;
j		In this example, a matrix x is printed to the auxiliary output. The output to the screen is turned off to speed up the printing. The end statement is used to terminate the program so the output file will be closed and the screen will be turned back on.
k		
l		
m		
n		
o		
p	See also	new, stop, system
q		
r		
s		
t		
u		
v		
w		
x y z		

endp

Purpose Closes a procedure or keyword definition.

Format **endp;**

Remarks **endp** marks the end of a procedure definition that began with a **proc** or **keyword** statement. (For details on writing and using procedures, see “Procedures and Keywords” in the *User’s Guide*.)

Example

```
proc regress(y,x);
    retp(inv(x'x)*x'y);
endp;
```

```
x = { 1 3 2, 7 4 9, 1 1 6, 3 3 2 };
```

```
y = { 3, 5, 2, 7 };
```

```
b = regress(y,x);
```

```

      1.00000000 3.00000000 2.00000000
x =   7.00000000 4.00000000 9.00000000
      1.00000000 1.00000000 6.00000000
      3.00000000 3.00000000 2.00000000
```

```

      3.00000000
y =   5.00000000
      2.00000000
      7.00000000
```

```

      0.15456890
b =   1.50276345
     -0.12840825
```

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

endp

See also **proc, keyword, retp**

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

endwind

Purpose	Ends graphic panel manipulation; display graphs with rerun .
Library	pgraph
Format	endwind;
Remarks	This function uses rerun to display the most recently created .tkf file.
Source	pwindow.src
See also	begwind, window, makewind, setwind, nextwind, getwind

[a](#)[b](#)[c](#)[d](#)[e](#)[f](#)[g](#)[h](#)[i](#)[j](#)[k](#)[l](#)[m](#)[n](#)[o](#)[p](#)[q](#)[r](#)[s](#)[t](#)[u](#)[v](#)[w](#)[x y z](#)

envget

envget

Purpose Searches the environment table for a defined name.

Format `y = envget(s);`

Input `s` string, the name to be searched for.

Output `y` string, the string that corresponds to that name in the environment or a null string if it is not found.

Example

```
proc dopen(file);  
    local fname,fp;  
    fname = envget("DPATH");  
    if fname $== "";  
        fname = file;  
    else;  
        if strsect(fname,strlen(fname),1) $== "\\\";  
            fname = fname $+ file;  
        else;  
            fname = fname $+ "\\\" $+ file;  
        endif;  
    endif;  
    open fp = ^fname;  
    retp(fp);  
endp;
```

This is an example of a procedure which will open a data file using a path stored in an environment string called DPATH. The procedure returns the file handle and is called as follows:

```
fp = dopen("myfile");
```

See also **cdir**

eof

Purpose Tests if the end of a file has been reached.

Format `y = eof(fh);`

Input `fh` scalar, file handle.

Output `y` scalar, 1 if end of file has been reached, else 0.

Remarks This function is used with the **readr** and **fgets**_{xxx} commands to test for the end of a file.

The **seekr** function can be used to set the pointer to a specific row position in a data set; the **fseek** function can be used to set the pointer to a specific byte offset in a file opened with **fopen**.

Example

```
open f1 = dat1;
xx = 0;
do until eof(f1);
    xx = xx+moment(readr(f1,100),0);
end;
```

In this example, the data file `dat1.dat` is opened and given the handle **f1**. Then the data are read from this data set and are used to create the moment (**x'x**) matrix of the data. On each iteration of the loop, 100 additional rows of data are read in and the moment matrix for this set of rows is computed, and added to the matrix **xx**. When all the data have been read, **xx** will contain the entire moment matrix for the data set.

GAUSS will keep reading until **eof(f1)** returns the value 1, which it will when the end of the data set has been reached. On the last iteration of the loop, all remaining observations are read in if there are 100 or fewer left.

See also **open**, **readr**, **seekr**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

eqSolve

eqSolve

a	Purpose	Solves a system of nonlinear equations
b	Format	{ <i>x</i> , <i>retcode</i> } = eqSolve (& <i>F</i> , <i>start</i>);
c	Input	<i>start</i> Kx1 vector, starting values.
d		& <i>F</i> scalar, a pointer to a procedure which computes the value at <i>x</i> of the equations to be solved.
e	Global Input	The folowing are set by eqsolveset .
f		_eqs_JacobianProc pointer to a procedure which computes the analytical Jacobian. By default, eqSolve will compute the Jacobian numerically.
g		_eqs_MaxIters scalar, the maximum number of iterations. Default = 100.
h		_eqs_StepTol scalar, the step tolerance. Default = __macheps ^(2/3).
i		_eqs_TypicalF Kx1 vector of the typical F(<i>x</i>) values at a point not near a root, used for scaling. This becomes important when the magnitudes of the components of F(<i>x</i>) are expected to be very different. By default, function values are not scaled.
j		_eqs_TypicalX Kx1 vector of the typical magnitude of <i>x</i> , used for scaling. This becomes important when the magnitudes of the components of <i>x</i> are expected to be very different. By default, variable values are not scaled.
k		_eqs_IterInfo scalar, if nonzero, iteration information is printed. Default = 0.
l		The following are set by gausset .
m		__Tol scalar, the tolerance of the scalar function $f = 0.5* F(x) ^2$ required to terminate the algorithm. Default = 1e-5.
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

eqSolve

__altnam	Kx1 character vector of alternate names to be used by the printed output. By default, the names X1, X2, X3... or X01, X02, X03 (depending on how __vpad is set) will be used.
__output	scalar. If non-zero, final results are printed.
__title	string, a custom title to be printed at the top of the iterations report. By default, only a generic title will be printed.

Output	x	Kx1 vector, solution.
	<i>retcode</i>	scalar, the return code: <ol style="list-style-type: none">1 Norm of the scaled function value is less than __Tol. x given is an approximate root of $F(x)$ (unless __Tol is too large).2 The scaled distance between the last two steps is less than the step-tolerance (__eqs_StepTol). x may be an approximate root of $F(x)$, but it is also possible that the algorithm is making very slow progress and is not near a root, or the step-tolerance is too large.3 The last global step failed to decrease norm2($F(x)$) sufficiently; either x is close to a root of $F(x)$ and no more accuracy is possible, or an incorrectly coded analytic Jacobian is being used, or the secant approximation to the Jacobian is inaccurate, or the step-tolerance is too large.4 Iteration limit exceeded.5 Five consecutive steps of maximum step length have been taken; either norm2($F(x)$) asymptotes from above to a finite value in some direction or the maximum step length is too small.6 x seems to be an approximate local minimizer of norm2($F(x)$) that is not a root of $F(x)$. To find a root of $F(x)$, restart eqSolve from a different region.

Remarks	The equation procedure should return a column vector containing the result for each equation. For example: Equation 1: $x1^2 + x2^2 - 2 = 0$ Equation 2: $\exp(x1-1) + x2^3 - 2 = 0$
---------	--

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

eqSolve

```
proc f(var);  
    local x1,x2,eqns;  
  
    x1 = var[1];  
    x2 = var[2];  
    eqns[1] = x1^2 + x2^2 - 2; /* Equation 1 */  
    eqns[2] = exp(x1-1) + x2^3 - 2; /* Equation  
                                     /* 2 */  
  
    retp( eqns );  
endp;
```

Example eqSolveset;

```
proc f(x);  
    local f1,f2,f3;  
    f1 = 3*x[1]^3 + 2*x[2]^2 + 5*x[3] - 10;  
    f2 = -x[1]^3 - 3*x[2]^2 + x[3] + 5;  
    f3 = 3*x[1]^3 + 2*x[2]^2 - 4*x[3];  
    retp(f1|f2|f3);  
endp;  
  
proc fjc(x);  
    local fjc1,fjc2, fjc3;  
    fjc1 = 9*x[1]^2 ~ 4*x[2] ~ 5;  
    fjc2 = -3*x[1]^2 ~ -6*x[2] ~ 1;  
    fjc3 = 9*x[1]^2 ~ 4*x[2] ~ -4;  
    retp(fjc1|fjc2|fjc3);  
endp;
```

eqSolve

```
start = { -1, 12, -1 };
```

```
_eqs_JacobianProc = &fjc;
```

```
{ x,tcode } = eqSolve(&f,start);
```

Produces

```
=====
EqSolve Version 3.2.22          2/24/97   9:54 am
=====
||F(X)|| at final solution:      0.93699762
-----
Termination Code = 1:
Norm of the scaled function value is less than
    __Tol;
-----
```

```
-----
VARIABLE  START      ROOTS      F(ROOTS)
-----
X1         -1.00000   0.54144351   4.4175402e-06
X2         12.00000   1.4085912   -6.6263102e-06
X3         -1.00000   1.1111111   4.4175402e-06
-----
```

Source eqsolve.src

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

eqSolve

eqSolve

a	
b	
c	
d	
e	
f	
g	
h	
i	
j	
k	
l	
m	
n	
o	
p	
q	
r	
s	
t	
u	
v	
w	
x y z	

Purpose Sets global input used by **eqSolve** to default values.

Format **eqSolve**;

Global **__eqs_TypicalX** = 0
Output **__eqs_TypicalF** = 0
 __eqs_IterInfo = 0
 __eqs_JacobianProc = 0
 __eqs_MaxIters = 100
 __eqs_StepTol **__macheps**^(2/3)

erf, erfc

Purpose Computes the Gaussian error function (**erf**) and its complement (**erfc**).

Format $y = \text{erf}(x);$
 $y = \text{erfc}(x);$

Input x NxK matrix.

Output y NxK matrix.

Remarks The allowable range for x is:

$x \geq 0$

The **erf** and **erfc** functions are closely related to the Normal distribution:

$$cdfn(x) = \begin{cases} \frac{1}{2} \left(1 + erf\left(\frac{x}{\sqrt{2}}\right) \right) & x \geq 0 \\ \frac{1}{2} erfc\left(\frac{-x}{\sqrt{2}}\right) & x < 0 \end{cases}$$

Example $x = \begin{Bmatrix} .5 & .4 & .3 \\ .6 & .8 & .3 \end{Bmatrix};$
 $y = \text{erf}(x);$

$y = \begin{matrix} 0.52049988 & 0.42839236 & 0.32862676 \\ 0.60385609 & 0.74210096 & 0.32862676 \end{matrix}$

$x = \begin{Bmatrix} .5 & .4 & .3 \\ .6 & .8 & .3 \end{Bmatrix};$
 $y = \text{erfc}(x);$

$y = \begin{matrix} 0.47950012 & 0.57160764 & 0.67137324 \\ 0.39614391 & 0.25789904 & 0.67137324 \end{matrix}$

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

erf, erfc

See also **cdfn, cdfnc**

Technical **erf** and **erfc** are computed by summing the appropriate series and
Notes continued fractions. They are accurate to about 10 digits.

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

error

Purpose	Allows the user to generate a user-defined error code which can be tested quickly with the scalerr function.	
Format	<code>y = error(x);</code>	
Input	<i>x</i>	scalar, in the range 0-65535.
Output	<i>y</i>	scalar error code which can be interpreted as an integer with the scalerr function.
Remarks	<p>The user may assign any number in the range 0-65535 to denote particular error conditions. This number may be tested for as an error code by scalerr.</p> <p>The scalerr function will return the value of the error code and so is the reverse of error. These user-generated error codes work in the same way as the intrinsic GAUSS error codes which are generated automatically when trap 1 is on and certain GAUSS functions detect a numerical error such as a singular matrix.</p> <p>error(0) is equal to the missing value code.</p>	
Example	<pre>proc syminv(x); local oldtrap,y; if not x == x'; retp(error(99)); endif; oldtrap = trapchk(0xffff); trap 1; y = invpd(x); if scalerr(y); y = inv(x); endif;</pre>	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

error

```
trap oldtrap,0xffff;
```

```
    retp(y);
```

```
endp;
```

The procedure **syminv** returns error code 99 if the matrix is not symmetric. If **invpd** fails, it returns error code 20. If **inv** fails, it returns error code 50. The original trap state is restored before the procedure returns.

See also **scalerr, trap, trapchk**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

errorlog

Purpose	Prints an error message to the window and error log file.
Format	errorlog <i>str</i> ;
Input	<i>str</i> string, the error message to print.
Remarks	This function prints to the screen and the error log file.

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

etdays

etdays

a	Purpose	Computes the difference between two times, as generated by the date command, in days.
b		
c	Format	<i>days</i> = etdays (<i>tstart</i> , <i>tend</i>) ;
d	Input	<i>tstart</i> 3x1 or 4x1 vector, starting date, in the order: yr, mo, day. (Only the first 3 elements are used.)
e		<i>tend</i> 3x1 or 4x1 vector, ending date, in the order: yr, mo, day. (Only the first 3 elements are used.) MUST be later than <i>tstart</i> .
f		
g	Output	<i>days</i> scalar, elapsed time measured in days.
h		
i	Remarks	This will work correctly across leap years and centuries. The assumptions are a Gregorian calendar with leap years on the years evenly divisible by 4 and not evenly divisible by 400.
j		
k	Example	let date1 = 1986 1 2 ; let date2 = 1987 10 25 ; d = etdays(date1,date2) ; d = 661
l		
m	Source	time.src
n	See also	dayinyr
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

ethsec

Purpose	Computes the difference between two times, as generated by the date command, in hundredths of a second.		
Format	<i>hs</i> = ethsec (<i>tstart</i> , <i>tend</i>) ;		
Input	<i>tstart</i>	4x1 vector, starting date, in the order: yr, mo, day, hundredths of a second.	
	<i>tend</i>	4x1 vector, ending date, in the order: yr, mo, day, hundredths of a second. MUST be later than <i>tstart</i> .	
Output	<i>hs</i>	scalar, elapsed time measured in hundredths of a second.	
Remarks	This will work correctly across leap years and centuries. The assumptions are a Gregorian calendar with leap years on the years evenly divisible by 4 and not evenly divisible by 400.		
Example	<pre>let date1 = 1986 1 2 0; let date2 = 1987 10 25 0; t = ethsec(date1,date2); t = 5711040000</pre>		
Source	time.src		
See also	dayinyr		

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

etstr

etstr

a	Purpose	Formats an elapsed time, measured in hundredths of a second, to a string.
b	Format	<i>str</i> = etstr (<i>tothsecs</i>);
c	Input	<i>tothsecs</i> scalar, an elapsed time measured in hundredths of a second, as given, for instance, by the ethsec function.
d	Output	<i>str</i> string containing the elapsed time in the form: # days # hours # minutes #.## seconds
e	Example	d1 = { 86, 1, 2, 0 }; d2 = { 86, 2, 5, 815642 }; t = ethsec(d1,d2); str = etstr(t); <i>t = 2.9457564e + 08</i> <i>str = 34 days 2 hours 15 minutes 56.42 seconds</i>
f	Source	time.src
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

EuropeanBinomCall

Purpose European binomial method call.

Format `c = EuropeanBinomCall(S0,K,r,div,tau,sigma,N);`

Input

<i>S0</i>	scalar, current price
<i>K</i>	Mx1 vector, strike prices
<i>r</i>	scalar, risk free rate
<i>div</i>	continuous dividend yield
<i>tau</i>	scalar, elapsed time to exercise in annualized days of trading
<i>sigma</i>	scalar, volatility
<i>N</i>	number of time segments

Output *c* Mx1 vector, call premiums

Example

```

S0 = 718.46;
K = { 720, 725, 730 };
b = .0498;
r = .0498;
sigma = .2493;
t0 = dtday (2001, 1, 30);
t1 = dtday (2001, 2, 16);
tau = elapsedTradingDays(t0,t1) /
      annualTradingDays(2001);
c = EuropeanBinomCall(S0,K,r,div,tau,sigma,N);
print c;

17.1071
15.0067

```

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

EuropeanBinomCall

12.9064

Source `finprocs.src`

Technical Notes The binomial method of Cox, Ross, and Rubinstein ("Option pricing: a simplified approach", Journal of Financial Economics, 7:229:264) as described in Options, Futures, and other Derivatives by John C. Hull is the basis of this procedure.

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

EuropeanBinomCall_Greeks

Purpose European binomial method call Delta, Gamma, Theta, Vega and Rho.

Format `{ d,g,t,v,rh } = EuropeanBinomCall_Greeks(S0,K,r,div,tau,sigma,N);`

Input

<i>S0</i>	scalar, current price
<i>K</i>	Mx1 vector, strike price
<i>r</i>	scalar, risk free rate
<i>div</i>	continuous dividend yield
<i>tau</i>	scalar, elapsed time to exercise in annualized days of trading
<i>sigma</i>	scalar, volatility
<i>N</i>	number of time segments

Output

<i>d</i>	Mx1 vector, delta
<i>g</i>	Mx1 vector, gamma
<i>t</i>	Mx1 vector, theta
<i>v</i>	Mx1 vector, vega
<i>rh</i>	Mx1 vector, rho

Example

```

S0 = 305;
K = 300;
r = .08;
sigma = .25;
tau = .33;
N = 30;
{ d,g,t,v,rh } = EuropeanBinomCall_Greeks
    (S0,K,r,div,tau,sigma,N);
print d;
print g;
print t;

```

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

EuropeanBinomCall_Greeks

		<pre>print v; print rh;</pre>
a		
b		0.6738
c		0.0008
d		-44.7874
e		69.0880
f		96.9225
g	Source	finprocs.src
h	Globals	_fin_thetaType scalar, if 1, one day look ahead, else, infinitesimal. Default = 0.
i		_fin_epsilon scalar, finite difference stepsize. Default = 1e-8.
j		
k	Technical Notes	The binomial method of Cox, Ross, and Rubinstein ("Option pricing: a simplified approach", Journal of Financial Economics, 7:229:264) as described in Options, Futures, and other Derivatives by John C. Hull is the basis of this procedure.
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

EuropeanBinomCall_ImpVol

Purpose Implied volatilities for European binomial method calls.

Format *sigma* =
EuropeanBinomCall_ImpVol(*c*,*S0*,*K*,*r*,*div*,*tau*,*N*);

Input

<i>c</i>	Mx1 vector, call premiums
<i>S0</i>	scalar, current price
<i>K</i>	Mx1 vector, strike prices
<i>r</i>	scalar, risk free rate
<i>div</i>	continuous dividend yield
<i>tau</i>	scalar, elapsed time to exercise in annualized days of trading
<i>N</i>	number of time segments

Output *sigma* Mx1 vector, volatility

Example

```

c = { 13.70, 11.90, 9.10 };
S0 = 718.46;
K = { 720, 725, 730 };
r = .0498;
t0 = dtday (2001, 1, 30);
t1 = dtday (2001, 2, 16);
tau = elapsedTradingDays(t0,t1) / annualTrading-
      Days(2001);
N = 30;
sigma = EuropeanBinomCall_ImpVol
        (c,S0,K,r,div,tau,N);
print sigma;

0.1982

```

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

EuropeanBinomCall_ImpVol

0.1716

0.1301

Source `finprocs.src`

Technical Notes The binomial method of Cox, Ross, and Rubinstein ("Option pricing: a simplified approach", Journal of Financial Economics, 7:229:264) as described in Options, Futures, and other Derivatives by John C. Hull is the basis of this procedure.

EuropeanBinomPut

Purpose European binomial method Put.

Format `c = EuropeanBinomPut(S0,K,r,div,tau,sigma,N);`

Input

<i>S0</i>	scalar, current price
<i>K</i>	Mx1 vector, strike prices
<i>r</i>	scalar, risk free rate
<i>div</i>	continuous dividend yield
<i>div</i>	continuous dividend yield
<i>tau</i>	scalar, elapsed time to exercise in annualized days of trading
<i>sigma</i>	scalar, volatility
<i>N</i>	number of time segments

Output *c* Mx1 vector, put premiums

Example:

```

S0 = 718.46;
K = { 720, 725, 730 };
r = .0498;
sigma = .2493;
t0 = dtday (2001, 1, 30);
t1 = dtday (2001, 2, 16);
tau = elapsedTradingDays(t0,t1) /
      annualTradingDays(2001);
N = 30;
c = EuropeanBinomPut
    (S0,K,r,div,tau,sigma,N);
print c;

16.6927

```

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

EuropeanBinomPut

19.5266

22.3604

Source finprocs.src

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

EuropeanBinomPut_Greeks

Purpose European binomial method put Delta, Gamma, Theta, Vega, and Rho.

Format $\{ d, g, t, v, rh \} =$
EuropeanBinomPut_Greeks(*S0*,*K*,*r*,*div*,*tau*,*sigma*,*N*);

Input

<i>S0</i>	scalar, current price
<i>K</i>	Mx1 vector, strike price
<i>r</i>	scalar, risk free rate
<i>div</i>	continuous dividend yield
<i>tau</i>	scalar, elapsed time to exercise in annualized days of trading
<i>sigma</i>	scalar, volatility
<i>N</i>	number of time segments

Output

<i>d</i>	Mx1 vector, delta
<i>g</i>	Mx1 vector, gamma
<i>t</i>	Mx1 vector, theta
<i>v</i>	Mx1 vector, vega
<i>rh</i>	Mx1 vector, rho

Example

```

S0 = 305;
K = 300;
r = .08;
sigma = .25;
tau = .33;
N = 30;

{ d,g,t,v,rh } = EuropeanBinomPut_Greeks
    (S0,K,r,div,tau,sigma,N);

print d;
print g;
print t;
```

EuropeanBinomPut_Greeks

		<pre>print v; print rh;</pre>
a		
b		-0.3804
c		0.0038
d		-17.9838
e		69.0880
f		-33.7666
g	Globals	_fin_thetaType scalar, if 1, one day look ahead, else, infinitesimal. Default = 0.
h		_fin_epsilon scalar, finite difference stepsize. Default = 1e-8.
i		
j	Source	finprocs.src
k	Technical Notes	The binomial method of Cox, Ross, and Rubinstein ("Option pricing: a simplified approach", Journal of Financial Economics, 7:229:264) as described in Options, Futures, and other Derivatives by John C. Hull is the basis of this procedure.
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

EuropeanBinomPut_ImpVol

Purpose Implied volatilities for European binomial method puts.

Format *sigma* = **EuropeanBinomPut_ImpVol**(*c*,*S0*,*K*,*r*,*div*,*tau*,*N*);

Input

<i>c</i>	Mx1 vector, put premiums
<i>S0</i>	scalar, current price
<i>K</i>	Mx1 vector, strike prices
<i>r</i>	scalar, risk free rate
<i>div</i>	continuous dividend yield
<i>tau</i>	scalar, elapsed time to exercise in annualized days of trading
<i>N</i>	number of time segments

Output *sigma* Mx1 vector, volatility

Example:

```
p = { 14.60, 17.10, 20.10 };
S0 = 718.46;
K = { 720, 725, 730 };
r = .0498;
t0 = dtday (2001, 1, 30);
t1 = dtday (2001, 2, 16);
tau = elapsedTradingDays(t0,t1) /
      annualTradingDays(2001);
N = 30;
sigma = EuropeanBinomPut_ImpVol
        (p,S0,K,r,div,tau,N);
print sigma;

0.1307
```

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

EuropeanBinomPut_ImpVol

0.1714

0.2165

Source `finprocs.src`

Technical Notes The binomial method of Cox, Ross, and Rubinstein ("Option pricing: a simplified approach", Journal of Financial Economics, 7:229:264) as described in Options, Futures, and other Derivatives by John C. Hull is the basis of this procedure.

EuropeanBSCall

Purpose European Black and Scholes Call.

Format `c = EuropeanBSCall(S0,K,r,div,tau,sigma);`

Input

<i>S0</i>	scalar, current price
<i>K</i>	Mx1 vector, strike prices
<i>r</i>	scalar, risk free rate
<i>div</i>	continuous dividend yield
<i>tau</i>	scalar, elapsed time to exercise in annualized days of trading
<i>sigma</i>	scalar, volatility

Output *c* Mx1 vector, call premiums

Example

```

S0 = 718.46;
K = { 720, 725, 730 };
b = .0498;
r = .0498;
div = 0;
sigma = .2493;
t0 = dtday (2001, 1, 30);
t1 = dtday (2001, 2, 16);
tau = elapsedTradingDays(t0,t1) /
      annualTradingDays(2001);
c = EuropeanBSCall(S0,K,r,div,tau,sigma);
print c;

17.0975
14.7583

```

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

EuropeanBSCall

12.6496

Source `finprocs.src`

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

EuropeanBSCall_Greeks

Purpose	European Black and Scholes call Delta, Gamma, Omega, Theta, and Vega.		
Format	$\{ d, g, t, v, rh \} =$ EuropeanBSCall_Greeks (<i>S0</i> , <i>K</i> , <i>r</i> , <i>div</i> , <i>tau</i> , <i>sigma</i>);		
Input	<i>S0</i>	scalar, current price	
	<i>K</i>	Mx1 vector, strike price	
	<i>r</i>	scalar, risk free rate	
	<i>div</i>	continuous dividend yield	
	<i>tau</i>	scalar, elapsed time to exercise in annualized days of trading	
	<i>sigma</i>	scalar, volatility	
Output	<i>d</i>	Mx1 vector, delta	
	<i>g</i>	Mx1 vector, gamma	
	<i>t</i>	Mx1 vector, theta	
	<i>v</i>	Mx1 vector, vega	
	<i>rh</i>	Mx1 vector, rho	
Example	<pre> S0 = 305; K = 300; r = .08; sigma = .25; tau = .33; { d,g,t,v,rh } = EuropeanBSCall_Greeks (S0,K,r,div,tau,sigma); print d; print g; print t; print v; </pre>		

EuropeanBSCall_Greeks

		print rh;
		0.6446
a		0.0085
b		-38.5054
c		65.2563
d		56.8720
e	Source	finprocs.src
f	Globals	_fin_thetaType scalar, if 1, one day look ahead, else, infinitesimal. Default = 0.
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

EuropeanBSCall_ImpVol

Purpose	Implied volatilities for European Black and Scholes calls.		
Format	<i>sigma</i> = EuropeanBSCall_ImpVol (<i>c</i> , <i>S0</i> , <i>K</i> , <i>r</i> , <i>div</i> , <i>tau</i>);		
Input	<i>c</i>	Mx1 vector, call premiums	
	<i>S0</i>	scalar, current price	
	<i>K</i>	Mx1 vector, strike prices	
	<i>r</i>	scalar, risk free rate	
	<i>div</i>	continuous dividend yield	
	<i>tau</i>	scalar, elapsed time to exercise in annualized days of trading	
Output	<i>sigma</i>	Mx1 vector, volatility	
Example	<i>c</i> = { 13.70, 11.90, 9.10 };		
	<i>S0</i> = 718.46;		
	<i>K</i> = { 720, 725, 730 };		
	<i>r</i> = .0498;		
	<i>div</i> = 0;		
	<i>t0</i> = dtday (2001, 1, 30);		
	<i>t1</i> = dtday (2001, 2, 16);		
	<i>tau</i> = elapsedTradingDays(<i>t0</i> , <i>t1</i>) /		
	annualTradingDays(2001);		
	<i>sigma</i> = EuropeanBSCall_ImpVol		
	(<i>c</i> , <i>S0</i> , <i>K</i> , <i>r</i> , <i>div</i> , <i>tau</i>);		
	print <i>sigma</i> ;		
	0.1991		
	0.1725		

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

EuropeanBSCall_ImpVol

0.1310

Source finprocs.src

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

EuropeanBSPut

Purpose European Black and Scholes Put.

Format `c = EuropeanBSPut(S0,K,r,div,tau,sigma);`

Input

- `S0` scalar, current price
- `K` Mx1 vector, strike prices
- `r` scalar, risk free rate
- `div` continuous dividend yield
- `tau` scalar, elapsed time to exercise in annualized days of trading
- `sigma` scalar, volatility

Output `c` Mx1 vector, put premiums

Example

```
S0 = 718.46;
K = { 720, 725, 730 };
r = .0498;
sigma = .2493;
div = 0;
t0 = dtday (2001, 1, 30);
t1 = dtday (2001, 2, 16);
tau = elapsedTradingDays(t0,t1) /
      annualTradingDays(2001);
c = EuropeanBSPut(S0,K,r,div,tau,sigma);
print c;

16.6403
19.2872
22.1647
```

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

EuropeanBSPut

Source `finprocs.src`

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

EuropeanBSPut_Greeks

Purpose European Black and Scholes put Delta, Gamma, Omega, Theta, and Vega.

Format $\{ d, g, t, v, rh \} =$
EuropeanBSPut_Greeks(*S0*,*K*,*r*,*div*,*tau*,*sigma*);

Input

<i>S0</i>	scalar, current price
<i>K</i>	Mx1 vector, strike price
<i>r</i>	scalar, risk free rate
<i>div</i>	continuous dividend yield
<i>tau</i>	scalar, elapsed time to exercise in annualized days of trading
<i>sigma</i>	scalar, volatility

Output

<i>d</i>	Mx1 vector, delta
<i>g</i>	Mx1 vector, gamma
<i>t</i>	Mx1 vector, theta
<i>v</i>	Mx1 vector, vega
<i>rh</i>	Mx1 vector, rho

Example

```

S0 = 305;
K = 300;
r = .08;
sigma = .25;
tau = .33;
{ d,g,t,v,rh } = EuropeanBSPut_Greeks
    (S0,K,r,div,tau,sigma);
print d;
print g;
print t;
print v;
```

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

EuropeanBSPut_Greeks

		print rh;
		-0.3554
a		0.0085
b		-15.1307
c		65.2563
d		-39.5486
e	Source	finprocs.src
f	Globals	_fin_thetaType scalar, if 1, one day look ahead, else, infinitesimal. Default = 0.
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

EuropeanBSPut_ImpVol

Purpose	Implied volatilities for European Black and Scholes puts.		
Format	<i>sigma</i> = EuropeanBSPut_ImpVol (<i>c</i> , <i>S0</i> , <i>K</i> , <i>r</i> , <i>div</i> , <i>tau</i>);		
Input	<i>c</i>	Mx1 vector, put premiums	
	<i>S0</i>	scalar, current price	
	<i>K</i>	Mx1 vector, strike prices	
	<i>r</i>	scalar, risk free rate	
	<i>div</i>	continuous dividend yield	
	<i>tau</i>	scalar, elapsed time to exercise in annualized days of trading	
Output	<i>sigma</i>	Mx1 vector, volatility	
Example	<pre>p = { 14.60, 17.10, 20.10 };</pre>		
	<pre>S0 = 718.46;</pre>		
	<pre>K = { 720, 725, 730 };</pre>		
	<pre>r = .0498;</pre>		
	<pre>t0 = dtday (2001, 1, 30);</pre>		
	<pre>t1 = dtday (2001, 2, 16);</pre>		
	<pre>tau = elapsedTradingDays(t0,t1) /</pre>		
	<pre> annualTradingDays(2001);</pre>		
	<pre>sigma = EuropeanBSPut_ImpVol(p,S0,K,r,div,tau);</pre>		
	<pre>print sigma;</pre>		
	<pre>0.2123</pre>		
	<pre>0.2493</pre>		
	<pre>0.2937</pre>		
	Source	finprocs.src	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

exctsmpl

exctsmpl

a	<div><div>Purpose</div><div>Computes a random subsample of a data set.</div><div>Format</div><div>$n = \text{exctsmpl}(\text{infile}, \text{outfile}, \text{percent});$</div><div>Input</div><div><div><div><div><i>infile</i></div><div>string, the name of the original data set.</div></div><div><div><i>outfile</i></div><div>string, the name of the data set to be created.</div></div><div><div><i>percent</i></div><div>scalar, the percentage random sample to take. This must be in the range 0-100.</div></div></div></div><div>Output</div><div><div><div><div><i>n</i></div><div>scalar, number of rows in output data set.</div></div><div>Error returns are controlled by the low bit of the trap flag.</div><div>trap 0 terminate with error message</div><div>trap 1 return scalar negative integer</div><div><div>-1</div><div>can't open input file</div></div><div><div>-2</div><div>can't open output file</div></div><div><div>-3</div><div>disk full</div></div></div></div><div>Remarks</div><div>Random sampling is done with replacement. Thus, an observation may be in the resulting sample more than once. If <i>percent</i> is 100, the resulting sample will not be identical to the original sample, though it will be the same size.</div><div>Example</div><div><div>$n = \text{exctsmpl}(\text{"freq.dat"}, \text{"rout"}, 30);$</div><div>$n = 30$</div><div>freq.dat is an example data set provided with GAUSS. Switching to the GAUSS examples directory will make it possible to do the above example as shown. Otherwise substitute data set names will need to be used.</div></div><div>Source</div><div>exctsmpl.src</div></div>
b	
c	
d	
e	
f	
g	
h	
i	
j	
k	
l	
m	
n	
o	
p	
q	
r	
s	
t	
u	
v	
w	
x y z	

exec

Purpose	Executes an executable program and returns the exit code to GAUSS.		
Format	<code>y = exec(<i>program</i>,<i>comline</i>);</code>		
Input	<i>program</i>	string, the name of the program to be executed.	
	<i>comline</i>	string, the arguments to be placed on the command line of the program being executed.	
Output	<i>y</i>	the exit code returned by <i>program</i> .	
		If exec cannot execute <i>program</i> , the error returns will be negative.	
		-1	file not found
		-2	the file is not an executable file
		-3	not enough memory
		-4	command line too long
Remarks	Under Windows, the file can be given an extension or a "." meaning no extension. Otherwise the file will be searched for with the following extensions in their given order: .com, .exe, .bat, .cmd.		
Example	<pre> y = exec("atog", "comd1.cmd"); if y; errorlog "atog failed"; end; endif; </pre>		
	<p>In this example, the ATOG ASCII conversion utility is executed under the exec function. The name of the command file to be used, <code>comd1.cmd</code>, is passed to ATOG on its command line. The exit code y returned by exec is tested to see if ATOG was successful; if not, the program will be terminated after printing an error message. See "Utilities" in the <i>User Guide</i>.</p>		

exp

exp

a	<div><div>Purpose</div><div>Calculates the exponential function.</div><div>Format</div><div>$y = \text{exp}(x);$</div><div>Input</div><div>x NxK matrix.</div><div>Output</div><div>y NxK matrix containing e, the base of natural logs, raised to the powers given by the elements of x.</div><div>Example</div><div>$x = \text{eye}(3);$ $y = \text{exp}(x);$ $x = \begin{bmatrix} 1.000000 & 0.000000 & 0.000000 \\ 0.000000 & 1.000000 & 0.000000 \\ 0.000000 & 0.000000 & 1.000000 \end{bmatrix}$ $y = \begin{bmatrix} 2.718282 & 1.000000 & 1.000000 \\ 1.000000 & 2.718282 & 1.000000 \\ 1.000000 & 1.000000 & 2.718282 \end{bmatrix}$ This example creates a 3x3 identity matrix and computes the exponential function for each one of its elements. Note that exp(1) returns e, the base of natural logs.</div><div>See also</div><div>ln</div></div>
b	
c	
d	
e	
f	
g	
h	
i	
j	
k	
l	
m	
n	
o	
p	
q	
r	
s	
t	
u	
v	
w	
x y z	

export

Purpose	Exports a GAUSS matrix to a spreadsheet, database or ASCII file.
Format	<code>ret = export(x,fname,namel);</code>
Input	<p><i>x</i> MxK matrix, data to export.</p> <p><i>fname</i> string, name of file to export with optional path.</p> <p><i>namel</i> Kx1 vector, column names, or scalar 0 for default column names.</p>
Output	<p><i>ret</i> scalar, success flag, 1 if successful, 0 if not.</p> <p><i>fname</i> file in the selected format.</p>
Portability	Windows only
Remarks	Setting <i>namel</i> to 0 results in default column names of Var:1 to Var:K.
Example	<pre>fname = "c:\\temp\\testdata.xls"; matx = rndn(7,4); let names = gnp invest pop export; call export(matx, fname, names);</pre> <p>This will export the matrix matx to the file c:\\temp\\testdata.xls in Excel format.</p> <p>If the first line had been</p> <pre>fname = "c:\\temp\\testdata.dbf";</pre> <p>the file would have been exported in dBase format.</p>
Globals	<code>_dxaschdr, _dxftype, _dxmiss, _dxprcn, _dxprint, _dxtxdlim, _dxtype, _dxwidth, _dxwkshd</code>
See also	<code>exportf, import, importf</code>

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

exportf**exportf**

Purpose	Exports a GAUSS data set to a spreadsheet, database, or ASCII file.
Format	<i>ret</i> = exportf (<i>dname</i> , <i>fname</i> , <i>namel</i>);
Input	<i>dname</i> string, name of GAUSS data set with optional path <i>fname</i> string, name of file to export with optional path. <i>namel</i> Kx1 vector, column names, or 0 for all columns.
Output	<i>ret</i> scalar, success flag, 1 if successful, 0 if not. <i>fname</i> file in the specified format.
Portability	Windows only
Remarks	The columns of the data set listed in <i>namel</i> will be exported. If <i>namel</i> is 0, the entire data set will be exported.
Example	<pre>dname = "c:\\gauss\\data\\mydata.dat"; fname = "c:\\temp\\testdata2.wk1"; call exportf(dname, fname, 0);</pre> <p>This will export the data set dname to the file c:\\temp\\testdata2.wk1 in Lotus format.</p>
Globals	_dxaschdr, _dxftype, _dxmiss, _dxprcn, _dxprint, _dxtxdlim, _dxtype, _dxwidth, _dxwkshdr
See also	export, import, importf

extern (dataloop)

Purpose	Allows access to matrices or strings in memory from inside a data loop.
Format	extern <i>variable_list</i> ;
Remarks	<p>Commas in <i>variable_list</i> are optional.</p> <p>extern tells the translator not to generate local code for the listed variables, and not to assume they are elements of the input data set.</p> <p>extern statements should be placed before any reference to the symbols listed. The specified names should not exist in the input data set, or be used in a make statement.</p>
Example	<p>This example shows how to assign the contents of an external vector to a new variable in the data set, by iteratively assigning a range of elements to the variable. The reserved variable x_x contains the data read from the input data set on each iteration. The external vector must have at least as many rows as the data set.</p> <pre> base = 1; /* used to index a range of elements /* from exvec */ dataloop olddata newdata; extern base, exvec; make ndvar = exvec[seqa(base,1,rows(x_x))]; /* execute command literally */ # base = base + rows(x_x); enddata; </pre>

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

external

external

a		
b		
c		
d		
e		
f		
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		
	Purpose	Lets the compiler know about symbols that are referenced above or in a separate file from their definitions.
	Format	external proc <i>dog,cat</i> ; external keyword <i>dog</i> ; external fn <i>dog</i> ; external matrix <i>x,y,z</i> ; external string <i>mstr,cstr</i> ;
	Remarks	See “Procedures and Keywords” in the <i>User’s Guide</i> . You may have several procedures in different files that reference the same global variable. By placing an external statement at the top of each file, you can let the compiler know if the symbol is a matrix, string, or procedure. If the symbol is listed and strongly typed in an active library, no external statement is needed. If a matrix or string appears in an external statement it needs to appear once in a declare statement. If no declaration is found, an Undefined symbol error will result.
	Example	The general eigenvalue procedure, eigrg , sets a global variable _eigerr if it cannot compute all of the eigenvalues. external matrix _eigerr; x = rndn(4,4); xi = inv(x); xev = eigrg(x); if _eigerr; print "Eigenvalues not computed"; end; endif; Without the external statement, the compiler would assume that _eigerr was a procedure and incorrectly compile this program. The file

external

containing the **eigr** procedure also contains an external statement that defines **_eigerr** as a matrix, but this would not be encountered until the **if** statement containing the reference to **_eigerr** in the main program file had already been incorrectly compiled.

See also **declare**

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

eye

eye

Purpose	Creates an identity matrix.		
Format	$y = \mathbf{eye}(n);$		
Input	n	scalar, size of identity matrix to be created.	
Output	y	NxN identity matrix.	
Remarks	If n is not an integer, it will be truncated to an integer. The matrix created will contain 1's down the diagonal and 0's everywhere else.		
Example	$x = \mathbf{eye}(3);$ $x = \begin{bmatrix} 1.000000 & 0.000000 & 0.000000 \\ 0.000000 & 1.000000 & 0.000000 \\ 0.000000 & 0.000000 & 1.000000 \end{bmatrix}$		
See also	zeros, ones		

fcheckerr

Purpose	Gets the error status of a file.	
Format	<i>err</i> = fcheckerr (<i>f</i>);	
Input	<i>f</i>	scalar, file handle of a file opened with fopen .
Output	<i>err</i>	scalar, error status.
Remarks	<p>If there has been a read or write error on a file, fcheckerr returns 1, otherwise 0.</p> <p>If you pass fcheckerr the handle of a file opened with open (i.e., a data set or matrix file), your program will terminate with a fatal error.</p>	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

fclearerr

fclearerr

a	Purpose	Gets the error status of a file, then clears it.
b	Format	<i>err</i> = fclearerr (<i>f</i>);
c	Input	<i>f</i> scalar, file handle of a file opened with fopen .
d	Output	<i>err</i> scalar, error status.
e	Remarks	Each file has an error flag that is set when there is an I/O error on the file. Typically, once this flag is set, you can no longer do I/O on the file, even if the error is a recoverable one. fclearerr clears the file's error flag, and you can attempt to continue using it.
f		If there has been a read or write error on a file, fclearerr returns 1, otherwise 0.
g		If you pass fclearerr the handle of a file opened with open (i.e., a data set or matrix file), your program will terminate with a fatal error.
h		The flag accessed by fclearerr is not the same as that accessed by fstreerror .
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

feq, fge, fgt, fle, flt, fne**feq, fge, fgt, fle, flt, fne**

Purpose	Fuzzy comparison functions. These functions use _fcmp_{tol} to fuzz the comparison operations to allow for roundoff error.	
Format	<pre> y = feq(a,b); y = fge(a,b); y = fgt(a,b); y = fle(a,b); y = flt(a,b); y = fne(a,b); </pre>	
Input	<i>a</i> NxK matrix, first matrix. <i>b</i> LxM matrix, second matrix, ExE compatible with <i>a</i> .	
Global Input	_fcmp_{tol}	global scalar, comparison tolerance. The default value is 1.0e-15.
Output	<i>y</i>	scalar, 1 (true) or 0 (false).
Remarks	<p>The return value is true if every comparison is true.</p> <p>The statement:</p> <pre>y = feq(a,b);</pre> <p>is equivalent to:</p> <pre>y = a eq b;</pre> <p>The calling program can reset _fcmp_{tol} before calling these procedures.</p> <pre>_fcmp_{tol} = 1e-12;</pre>	
Example	<pre> x = rndu(2,2); y = rndu(2,2); t = fge(x,y); </pre>	

feq, fge, fgt, fle, flt, fne

$x =$ 0.0382895 0.07253527
 0.01471395 0.96863611

$y =$ 0.25622293 0.70636474
 0.00361912 0.35913385

$t =$ 0.00000000

Source fcompare.src

Globals **_fcmptol**

See also **dotfeq-dotfne**

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

fflush

Purpose	Flushes a file's output buffer.
Format	<i>ret</i> = fflush (<i>f</i>);
Input	<i>f</i> scalar, file handle of a file opened with fopen .
Output	<i>ret</i> scalar, 0 if successful, -1 if not.
Remarks	<p>If fflush fails, you can call fstrerror to find out why.</p> <p>If you pass fflush the handle of a file opened with open (i.e., a data set or matrix file), your program will terminate with a fatal error.</p>

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

fft

fft

a	Purpose	Computes a 1- or 2-D Fast Fourier transform.
b	Format	$y = \text{fft}(x);$
c	Input	x NxK matrix.
d	Output	y LxM matrix, where L and M are the smallest powers of 2 greater than or equal to N and K, respectively.
e	Remarks	This computes the FFT of x , scaled by 1/N. This uses a Temperton Fast Fourier algorithm. If N or K is not a power of 2, x will be padded out with zeros before computing the transform.
f	Example	$x = \begin{Bmatrix} 22 & 24 \\ 23 & 25 \end{Bmatrix};$ $y = \text{fft}(x);$ $y = \begin{bmatrix} 23.500000 & -1.000000 \\ -0.500000 & 0.000000 \end{bmatrix}$
g	See also	ffti , rfft , rffti

ffti

Purpose	Computes an inverse 1- or 2-D Fast Fourier transform.		
Format	$y = \text{ffti}(x);$		
Input	x	NxK matrix.	
Output	y	LxM matrix, where L and M are the smallest prime factor products greater than or equal to N and K, respectively.	
Remarks	Computes the inverse FFT of x , scaled by 1/N. This uses a Temperton prime factor Fast Fourier algorithm.		
Example	$x = \begin{Bmatrix} 22 & 24 \\ 23 & 25 \end{Bmatrix};$ $y = \text{fft}(x);$ $y = \begin{bmatrix} 23.500000 & -1.000000 \\ -0.500000 & 0.000000 \end{bmatrix}$ $fi = \text{ffti}(y);$ $fi = \begin{bmatrix} 22.000000 & 24.000000 \\ 23.000000 & 25.000000 \end{bmatrix}$		
See also	fft, rfft, rffti		

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

fftm

fftm

Purpose	Computes a multi-dimensional FFT.
Format	$y = \text{fftm}(x, dim);$
Input	x Mx1 vector, data. dim Kx1 vector, size of each dimension.
Output	y Lx1 vector, FFT of x .
Remarks	The multi-dimensional data are laid out in a recursive or heirarchical fashion in the vector x . That is to say, the elements of any given dimension are stored in sequence left to right within the vector, with each element containing a sequence of elements of the next smaller dimension. In abstract terms, a 4-dimensional 2x2x2x2 hypercubic x would consist of two cubes in sequence, each cube containing two matrices in sequence, each matrix containing two rows in sequence, and each row containing two columns in sequence. Visually, x would look something like this:

$$\begin{aligned} X_{hyper} &= X_{cube1} \mid X_{cube2} \\ X_{cube1} &= X_{mat1} \mid X_{mat2} \\ X_{mat1} &= X_{row1} \mid X_{row2} \\ X_{row1} &= X_{col1} \mid X_{col2} \end{aligned}$$

Or, in an extended GAUSS notation, x would be:

$$\begin{aligned} X_{hyper} &= x[1, ., ., ., .] \mid x[2, ., ., ., .]; \\ X_{cube1} &= x[1, 1, ., ., .] \mid x[1, 2, ., ., .]; \\ X_{mat1} &= x[1, 1, 1, ., .] \mid x[1, 1, 2, ., .]; \\ X_{row1} &= x[1, 1, 1, 1, .] \mid x[1, 1, 1, 2, .]; \end{aligned}$$

To be explicit, x would be laid out like this:

$$\begin{aligned} &x[1, 1, 1, 1, 1] \ x[1, 1, 1, 1, 2] \ x[1, 1, 2, 1, 1] \ x[1, 1, 2, 1, 2] \\ &x[1, 2, 1, 1, 1] \ x[1, 2, 1, 1, 2] \ x[1, 2, 2, 1, 1] \ x[1, 2, 2, 1, 2] \\ &x[2, 1, 1, 1, 1] \ x[2, 1, 1, 1, 2] \ x[2, 1, 2, 1, 1] \ x[2, 1, 2, 1, 2] \\ &x[2, 2, 1, 1, 1] \ x[2, 2, 1, 1, 2] \ x[2, 2, 2, 1, 1] \ x[2, 2, 2, 1, 2] \end{aligned}$$

If you look at the last diagram for the layout of x you'll notice that each line actually constitutes the elements of an ordinary matrix in normal row-major order. This is easy to achieve with **vecr**. Further, each pair of lines or "matrices" constitutes one of the desired cubes, again with all the elements in the correct order. And finally, the two cubes combine to form the hypercube. So, the process of construction is simply a sequence of concatenations of column vectors, with a **vecr** step if necessary to get started.

Here's an example, this time working with a 2x3x2x3 hypercube.

```
let dim = 2 3 2 3;

let x1[2,3] = 1 2 3 4 5 6;
let x2[2,3] = 6 5 4 3 2 1;
let x3[2,3] = 1 2 3 5 7 11;
xc1 = vecr(x1)|vecr(x2)|vecr(x3); /* cube 1 */

let x1 = 1 1 2 3 5 8;
let x2 = 1 2 6 24 120 720;
let x3 = 13 17 19 23 29 31;
xc2 = x1|x2|x3; /* cube 2 */

xh = xc1|xc2; /* hypercube */
xhfft = fftm(xh,dim);

let dimi = 2 4 2 4;
xhffti = fftmi(xhfft,dimi);
```

We left out the **vecr** step for the 2nd cube. It's not really necessary when you're constructing the matrices with **let** statements.

dim contains the dimensions of x , beginning with the highest dimension. The last element of **dim** is the number of columns, the next to the last element of **dim** is the number of rows, and so on. Thus

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

fftm

```
dim = { 2, 3, 3 };
```

indicates that the data in x is a 2x3x3 three-dimensional array, i.e., two 3x3 matrices of data. Suppose that **x1** is the first 3x3 matrix and **x2** the second 3x3 matrix, then **x = vecr(x1) | vecr(x2)**.

The size of **dim** tells you how many dimensions x has.

The arrays have to be padded in each dimension to the nearest power of two. Thus the output array can be larger than the input array. In the 2x3x2x3 hypercube example, x would be padded from 2x3x2x3 out to 2x4x2x4. The input vector would contain 36 elements, while the output vector would contain 64 elements. You may have noticed that we used a **dimi** with padded values at the end of the example to check our answer.

Source `fftm.src`

See also **fftm_i, fft, ffti, fftn**

ffftmi

Purpose Computes a multi-dimensional inverse FFT.

Format $y = \text{ffftmi}(x, \text{dim});$

Input

x	Mx1 vector, data.
dim	Kx1 vector, size of each dimension.

Output y Lx1 vector, inverse FFT of x .

Remarks The multi-dimensional data are laid out in a recursive or heirarchical fashion in the vector x . That is to say, the elements of any given dimension are stored in sequence left to right within the vector, with each element containing a sequence of elements of the next smaller dimension. In abstract terms, a 4-dimensional 2x2x2x2 hypercubic x would consist of two cubes in sequence, each cube containing two matrices in sequence, each matrix containing two rows in sequence, and each row containing two columns in sequence. Visually, x would look something like this:

$$\begin{aligned} X_{\text{hyper}} &= X_{\text{cube1}} \mid X_{\text{cube2}} \\ X_{\text{cube1}} &= X_{\text{mat1}} \mid X_{\text{mat2}} \\ X_{\text{mat1}} &= X_{\text{row1}} \mid X_{\text{row2}} \\ X_{\text{row1}} &= X_{\text{col1}} \mid X_{\text{col2}} \end{aligned}$$

Or, in an extended GAUSS notation, x would be:

$$\begin{aligned} X_{\text{hyper}} &= x[1, ., ., ., .] \mid x[2, ., ., ., .]; \\ X_{\text{cube1}} &= x[1, 1, ., ., .] \mid x[1, 2, ., ., .]; \\ X_{\text{mat1}} &= x[1, 1, 1, ., .] \mid x[1, 1, 2, ., .]; \\ X_{\text{row1}} &= x[1, 1, 1, 1, .] \mid x[1, 1, 1, 2, .]; \end{aligned}$$

To be explicit, x would be laid out like this:

$$\begin{aligned} &x[1, 1, 1, 1] \ x[1, 1, 1, 2] \ x[1, 1, 2, 1] \ x[1, 1, 2, 2] \\ &x[1, 2, 1, 1] \ x[1, 2, 1, 2] \ x[1, 2, 2, 1] \ x[1, 2, 2, 2] \\ &x[2, 1, 1, 1] \ x[2, 1, 1, 2] \ x[2, 1, 2, 1] \ x[2, 1, 2, 2] \\ &x[2, 2, 1, 1] \ x[2, 2, 1, 2] \ x[2, 2, 2, 1] \ x[2, 2, 2, 2] \end{aligned}$$

fftmi

If you look at the last diagram for the layout of x you'll notice that each line actually constitutes the elements of an ordinary matrix in normal row-major order. This is easy to achieve with **vecr**. Further, each pair of lines or "matrices" constitutes one of the desired cubes, again with all the elements in the correct order. And finally, the two cubes combine to form the hypercube. So, the process of construction is simply a sequence of concatenations of column vectors, with a **vecr** step if necessary to get started.

Here's an example, this time working with a 2x3x2x3 hypercube.

```
let dim = 2 3 2 3;

let x1[2,3] = 1 2 3 4 5 6;
let x2[2,3] = 6 5 4 3 2 1;
let x3[2,3] = 1 2 3 5 7 11;
xc1 = vecr(x1)|vecr(x2)|vecr(x3); /* cube 1 */

let x1 = 1 1 2 3 5 8;
let x2 = 1 2 6 24 120 720;
let x3 = 13 17 19 23 29 31;
xc2 = x1|x2|x3; /* cube 2 */

xh = xc1|xc2; /* hypercube */
xhfftmi = fftmi(xh,dim);
```

We left out the **vecr** step for the 2nd cube. It's not really necessary when you're constructing the matrices with **let** statements.

dim contains the dimensions of x , beginning with the highest dimension. The last element of **dim** is the number of columns, the next to the last element of **dim** is the number of rows, and so on. Thus

```
dim = { 2, 3, 3 };
```

indicates that the data in x is a 2x3x3 three-dimensional array, i.e., two 3x3 matrices of data. Suppose that **x1** is the first 3x3 matrix and **x2** the second 3x3 matrix, then **x = vecr(x1) | vecr(x2)**.

The size of **dim** tells you how many dimensions x has.

The arrays have to be padded in each dimension to the nearest power of two. Thus the output array can be larger than the input array. In the 2x3x2x3 hypercube example, x would be padded from 2x3x2x3 out to 2x4x2x4. The input vector would contain 36 elements, while the output vector would contain 64 elements.

Source `fftm.src`

See also **fftm**, **fft**, **ffti**, **fftn**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

fftn

fftn

a	Purpose	Computes a complex 1- or 2-D FFT.
b	Format	$y = \text{fftn}(x);$
c	Input	x NxK matrix.
d	Output	y LxM matrix, where L and M are the smallest prime factor products greater than or equal to N and K, respectively.
e	Remarks	<p>fftn uses the Temperton prime factor FFT algorithm. This algorithm can compute the FFT of any vector or matrix whose dimensions can be expressed as the product of selected prime number factors. GAUSS implements the Temperton algorithm for any power of 2, 3, and 5, and one factor of 7. Thus, fftn can handle any matrix whose dimensions can be expressed as</p> $2^p \times 3^q \times 5^r \times 7^s, \quad \begin{array}{l} p,q,r \text{ nonnegative integers} \\ s = 0 \text{ or } 1 \end{array}$ <p>If a dimension of x does not meet this requirement, it will be padded with zeros to the next allowable size before the FFT is computed.</p> <p>fftn pads matrices to the next allowable dimensions; however, it generally runs faster for matrices whose dimensions are highly composite numbers, i.e., products of several factors (to various powers), rather than powers of a single factor. For example, even though it is bigger, a 33600x1 vector can compute as much as 20% faster than a 32768x1 vector, because 33600 is a highly composite number, $2^6 \times 3 \times 5^2 \times 7$, whereas 32768 is a simple power of 2, 2^{15}. For this reason, you may want to hand-pad matrices to optimum dimensions before passing them to fftn. The Run-Time Library includes a routine, optn, for determining optimum dimensions.</p> <p>The Run-Time Library also includes the nextn routine, for determining allowable dimensions for a matrix. (You can use this to see the dimensions to which fftn would pad a matrix.)</p> <p>fftn scales the computed FFT by $1/(L*M)$.</p>
f	See also	fft , ffti , fftm , fftm_i , rfft , rffti , rfftip , rfftn , rfftnp , rfftp
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

fgets

Purpose	Reads a line of text from a file.		
Format	<i>str</i> = fgets (<i>f</i> , <i>maxsize</i>) ;		
Input	<i>f</i>	scalar, file handle of a file opened with fopen .	
	<i>maxsize</i>	scalar, maximum size of string to read in, including the terminating null byte.	
Output	<i>str</i>	string.	
Remarks	fgets reads text from a file into a string. It reads up to a newline, the end of the file, or <i>maxsize</i> -1 characters. The result is placed in <i>str</i> , which is then terminated with a null byte. The newline, if present, is retained.		
	If the file is already at end-of-file when you call fgets , your program will terminate with an error. Use eof in conjunction with fgets to avoid this.		
	If the file was opened for update (see fopen) and you are switching from writing to reading, don't forget to call fseek or fflush first, to flush the file's buffer.		
	If you pass fgets the handle of a file opened with open (i.e., a data set or matrix file), your program will terminate with a fatal error.		

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

fgetsa

fgetsa

a	Purpose	Reads lines of text from a file into a string array.
b	Format	<i>sa</i> = fgetsa (<i>f</i> , <i>numl</i>) ;
c	Input	<i>f</i> scalar, file handle of a file opened with fopen . <i>numl</i> scalar, number of lines to read.
d	Output	<i>sa</i> Nx1 string array, $N \leq numl$.
e	Remarks	<p>fgetsa reads up to <i>numl</i> lines of text. If fgetsa reaches the end of the file before reading <i>numl</i> lines, <i>sa</i> will be shortened. Lines are read in the same manner as fgets, except that no limit is placed on the size of a line. Thus, fgetsa always returns complete lines of text. Newlines are retained. If <i>numl</i> is 1, fgetsa returns a string. (This is one way to read a line from a file without placing a limit on the length of the line.)</p> <p>If the file is already at end-of-file when you call fgetsa, your program will terminate with an error. Use eof in conjunction with fgetsa to avoid this. If the file was opened for update (see fopen) and you are switching from writing to reading, don't forget to call fseek or fflush first, to flush the file's buffer.</p> <p>If you pass fgetsa the handle of a file opened with open (i.e., a data set or matrix file), your program will terminate with a fatal error.</p>
f		
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

fgetsat

Purpose	Reads lines of text from a file into a string array.	
Format	<code>sa = fgetsat(f,numl);</code>	
Input	<i>f</i>	scalar, file handle of a file opened with fopen .
	<i>numl</i>	scalar, number of lines to read.
Output	<i>sa</i>	Nx1 string array, $N \leq numl$.
Remarks	<p>fgetsat operates identically to fgetsa, except that newlines are not retained as text is read into <i>sa</i>.</p> <p>In general, you don't want to use fgetsat on files opened in binary mode (see fopen). fgetsat drops the newlines, but it does NOT drop the carriage returns that precede them on some platforms. Printing out such a string array can produce unexpected results.</p>	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

fgetst

fgetst

a		
b		
c		
d		
e		
f	Purpose	Reads a line of text from a file.
g	Format	<i>str</i> = fgetst (<i>f</i> , <i>maxsize</i>) ;
h	Input	<i>f</i> scalar, file handle of a file opened with fopen . <i>maxsize</i> scalar, maximum size of string to read in, including the null terminating byte.
i	Output	<i>str</i> string.
j	Remarks	fgetst operates identically to fgets , except that the newline is not retained in the string. In general, you don't want to use fgetst on files opened in binary mode (see fopen). fgetst drops the newline, but it does NOT drop the preceding carriage return used on some platforms. Printing out such a string can produce unexpected results.
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

fileinfo

Purpose Returns names and information for files that match a specification.

Format `{ fnames,finfo } = fileinfo(fspec);`

Input *fspec* string, file specification. Can include path. Wildcards are allowed in the filename.

Output *fnames* Nx1 string array of all filenames that match, null string if none are found.

finfo Nx13 matrix, information about matching files.

UNIX

[N, 1]	filesystem ID
[N, 2]	inode number
[N, 3]	mode bit mask
[N, 4]	number of links
[N, 5]	user ID
[N, 6]	group ID
[N, 7]	device ID (char/block special files only)
[N, 8]	size in bytes
[N, 9]	last access time
[N, 10]	last data modification time
[N, 11]	last file status change time
[N, 12]	preferred I/O block size
[N, 13]	number of 512-byte blocks allocated

OS/2, Windows

[N, 1]	drive number (A = 0, B = 1, etc.)
[N, 2]	n/a, 0
[N, 3]	mode bit mask
[N, 4]	number of links, always 1
[N, 5]	n/a, 0
[N, 6]	n/a, 0
[N, 7]	n/a, 0
[N, 8]	size in bytes

fileinfo

	[N, 9]	last access time
	[N,10]	last data modification time
a	[N,11]	creation time
b	[N,12]	n/a, 0
c	[N,13]	n/a, 0

DOS

d	[N, 1]	drive number (A = 0, B = 1, etc.)
e	[N, 2]	n/a, 0
f	[N, 3]	mode bit mask
g	[N, 4]	number of links, always 1
h	[N, 5]	n/a, 0
i	[N, 6]	n/a, 0
j	[N, 7]	n/a, 0
k	[N, 8]	size in bytes
l	[N, 9]	n/a, 0
m	[N,10]	last data modification time
n	[N,11]	n/a, 0
o	[N,12]	n/a, 0
p	[N,13]	n/a, 0

finfo will be a scalar zero if no matches are found.

Remarks *fnames* will contain file *names* only; any path information that was passed is dropped.

 The time stamp fields (*finfo*[N,9]-[N,11]) are expressed as the number of seconds since midnight, Jan. 1, 1970, Coordinated Universal Time (UTC).

See also **files, filesa**

filesa

Purpose	Returns a string array of file names.	
Format	<code>y = filesa(n);</code>	
Input	<i>n</i>	string, file specification to search for. Can include path. Wildcards are allowed in the filename.
Output	<i>y</i>	Nx1 string array of all filenames that match, or null string if none are found.
Remarks	<i>y</i> will contain file <i>names</i> only; any path information that was passed is dropped.	
Example	<pre>y = filesa("ch*");</pre> <p>In this example all files listed in the current directory that begin with “ch” will be returned.</p> <pre>proc exist(filename); retp(not filesa(filename) \$== ""); endp;</pre> <p>This procedure will return 1 if the file exists or 0 if not.</p>	
See also	fileinfo, files, shell	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

floor

floor

a	Purpose	Rounds down toward $-\infty$.
b	Format	$y = \text{floor}(x);$
c	Input	x NxK matrix.
d	Output	y NxK matrix containing the elements of x rounded down.
e	Remarks	This rounds every element in the matrix x down to the nearest integer.
f	Example	$x = 100*\text{rndn}(2,2);$ $x = \begin{matrix} 77.68 & -14.10 \\ 4.73 & -158.88 \end{matrix}$ $f = \text{floor}(x);$ $f = \begin{matrix} 77.00 & -15.00 \\ 4.00 & -159.00 \end{matrix}$
g	See also	ceil, round, trunc

fmod

Purpose	Computes the floating-point remainder of x/y .		
Format	$r = \text{fmod}(x,y);$		
Input	x	NxK matrix.	
	y	LxM matrix, ExE conformable with x .	
Output	r	max(N,L) by max(K,M) matrix.	
Remarks	Returns the floating-point remainder r of x/y such that $x = iy + r$, where i is an integer, r has the same sign as x , and $ r < y $.		
	Compare this with <code>%</code> , the modulo division operator. (See “Operators” in the <i>User’s Guide</i> .)		
Example	$x = \text{seqa}(1.7, 2.3, 5)';$		
	$y = 2;$		
	$r = \text{fmod}(x,y);$		
	$x = \begin{bmatrix} 1.7 & 4 & 6.3 & 8.6 & 10.9 \end{bmatrix}$		
	$r = \begin{bmatrix} 1.7 & 0 & 0.3 & 0.6 & 0.9 \end{bmatrix}$		

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

fn

fn

Purpose	Allows user to create one-line functions.
Format	fn <i>fn_name(args)</i> = <i>code_for_function</i> ;
Remarks	Functions can be called in the same way as other procedures.
Example	<pre>fn area(r) = pi*r*r; a = area(4); a = 50.265482</pre>

fonts

Purpose	Loads fonts to be used in the graph.		
Library	pgraph		
Format	fonts (<i>str</i>);		
Input	<i>str</i>	string or character vector containing the names of fonts to be used in the plot.	
		Simplex	standard sans serif font.
		Simgrma	Simplex Greek, math.
		Microb	bold and boxy.
		Complex	standard font with serif.
		The first font specified will be used for the axes numbers.	
		If <i>str</i> is a null string, or fonts is not called, Simplex is loaded by default.	
Remarks	For information on how to select fonts within a text string, see “Publication Quality Graphics in the <i>User’s Guide</i> .”		
Source	pgraph.src		
See also	title , xlabel , ylabel , zlabel		

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

fopen

fopen

a	Purpose	Opens a file.
b	Format	<i>f</i> = fopen (<i>filename</i> , <i>omode</i>) ;
c	Input	<i>filename</i> string, name of file to open. <i>omode</i> string, file I/O mode. (See Remarks, below.)
d	Output	<i>f</i> scalar, file handle.
e	Portability	UNIX Carriage return-linefeed conversion for files opened in text mode is unnecessary, because in UNIX a newline is simply a linefeed.
f	Remarks	<i>filename</i> can contain a path specification. <i>omode</i> is a sequence of characters that specify the mode in which to open the file. The first character must be one of: r Open an existing file for reading. If the file does not exist, fopen fails. w Open or create a file for writing. If the file already exists, its current contents will be destroyed. a Open or create a file for appending. All output is appended to the end of the file. To this can be appended a + and/or a b . The + indicates the file is to be opened for reading and writing, or update, as follows: r+ Open an existing file for update. You can read from or write to any location in the file. If the file does not exist, fopen fails. w+ Open or create a file for update. You can read from or write to any location in the file. If the file already exists, its current contents will be destroyed. a+ Open or create a file for update. You can read from any location in the file, but all output will be appended to the end of the file. Finally, the b indicates whether the file is to be opened in text or binary mode. If the file is opened in binary mode, the contents of the file are read verbatim; likewise, anything output to the file is written verbatim. In text mode (the default), carriage return-linefeed sequences are converted on
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

fopen

input to linefeeds, or newlines. Likewise on output, newlines are converted to carriage return-linefeeds. Also in text mode, if a CTRL-Z (char 26) is encountered during a read, it is interpreted as an end-of-file character, and reading ceases. In binary mode, CTRL-Z is read in uninterpreted.

The order of **+** and **b** is not significant; **rb+** and **r+b** mean the same thing.

You can both read from and write to a file opened for update. However, before switching from one to the other, you must make an **fseek** or **fflush** call, to flush the file's buffer.

If **fopen** fails, it returns a 0.

Use **close** and **closeall** to close files opened with **fopen**.

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

for

for

a		
b	Purpose	Begins a for loop.
c	Format	for <i>i</i> (<i>start</i> , <i>stop</i> , <i>step</i>); . . . endfor ;
d		
e		
f		
g	Input	<i>i</i> literal, the name of the counter variable. <i>start</i> scalar expression, the initial value of the counter. <i>stop</i> scalar expression, the final value of the counter. <i>step</i> scalar expression, the increment value.
h		
i		
j	Remarks	The counter is strictly local to the loop. The expressions <i>start</i> , <i>stop</i> , and <i>step</i> are evaluated only once when the loop initializes. They are converted to integers and stored local to the loop. The for loop is optimized for speed and is much faster than a do loop. The commands break and continue are supported. The continue command steps the counter and jumps to the top of the loop. The break command terminates the current loop. The loop terminates when the value of <i>i</i> exceeds <i>stop</i> . If break is used to terminate the loop and you want the final value of the counter, you need to assign it to a variable before the break statement (see the third example, following).
k		
l		
m		
n		
o		
p		
q		
r	Example	Example 1 x = zeros(10, 5); for i (1, rows(x), 1); for j (1, cols(x), 1); x[i,j] = i*j; endfor; endfor;
s		
t		
u		
v		
w		
x y z		

Example 2

```
x = rndn(3,3);
y = rndn(3,3);
for i (1, rows(x), 1);
    for j (1, cols(x), 1);
        if x[i,j] ≥ y[i,j];
            continue;
        endif;
        temp = x[i,j];
        x[i,j] = y[i,j];
        y[i,j] = temp;
    endfor;
endfor;
```

Example 3

```
li = 0;
x = rndn(100,1);
y = rndn(100,1);
for i (1, rows(x), 1);
    if x[i] /= y[i];
        li = i;
        break;
    endif;
endfor;
if li;
    print "Compare failed on row " li;
endif;
```

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

format

format

Purpose	Controls the format of matrices and numbers printed out with print or lprint statements.		
Format	format <i>[/typ]</i> <i>[/fnted]</i> <i>[/mf]</i> <i>[/jnt]</i> <i>[f,p];</i>		
Input	<i>/typ</i>	literal, symbol type flag(s). Indicate which symbol types you are setting the output format for. <i>/mat, /sa, /str</i>	Formatting parameters are maintained separately for matrices (<i>/mat</i>), string arrays (<i>/sa</i>), and strings (<i>/str</i>). You can specify more than one <i>/typ</i> flag; the format will be set for all types indicated. If no <i>/typ</i> flag is listed, format assumes <i>/mat</i> .
	<i>/fnted</i>	literal, enable formatting flag. <i>/on, /off</i>	Enable/disable formatting. When formatting is disabled, the contents of a variable are dumped to the window in a “raw” format. <i>/off</i> is currently supported only for strings. Raw format for strings means that the entire string is printed, starting at the current cursor position. When formatting is enabled for strings, they are handled the same as string arrays. This shouldn’t be too surprising, since a string is actually a 1x1 string array.
	<i>/mf</i>	literal, matrix row format flag. <i>/m0</i>	no delimiters before or after rows when printing out matrices.
		<i>/m1 or /mb1</i>	print 1 carriage return/line feed pair before each row of a matrix with more than 1 row.
		<i>/m2 or /mb2</i>	print 2 carriage return/line feed pairs before each row of a matrix with more than 1 row.
		<i>/m3 or /mb3</i>	print “Row 1”, “Row 2”... before each row of a matrix with more than one row.

format

/ma1	print 1 carriage return/line feed pair after each row of a matrix with more than 1 row.
/ma2	print 2 carriage return/line feed pairs after each row of a matrix with more than 1 row.
/a1	print 1 carriage return/line feed pair after each row of a matrix.
/a2	print 2 carriage return/line feed pairs after each row of a matrix.
/b1	print 1 carriage return/line feed pair before each row of a matrix.
/b2	print 2 carriage return/line feed pairs before each row of a matrix.
/b3	print “Row 1”, “Row 2”... before each row of a matrix.
/jnt	literal, matrix element format flag controls justification, notation and trailing character.
<u>Right-Justified</u>	
/rd	Signed decimal number in the form <code>[[-]] #####.#####</code> , where ##### is one or more decimal digits. The number of digits before the decimal point depends on the magnitude of the number, and the number of digits after the decimal point depends on the precision. If the precision is 0, no decimal point will be printed.
/re	Signed number in the form <code>[[-]] #####.#####</code> , where # is one decimal digit, ## is one or more decimal digits depending on the precision, and ### is three decimal digits. If precision is 0, the form will be <code>[[-]] #E±###</code> with no decimal point printed.

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

format

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

/ro This will give a format like **/rd** or **/re** depending on which is most compact for the number being printed. A format like **/re** will be used only if the exponent value is less than -4 or greater than the precision. If a **/re** format is used, a decimal point will always appear. The precision signifies the number of significant digits displayed.

/rz This will give a format like **/rd** or **/re** depending on which is most compact for the number being printed. A format like **/re** will be used only if the exponent value is less than -4 or greater than the precision. If a **/re** format is used, trailing zeros will be suppressed and a decimal point will appear only if one or more digits follow it. The precision signifies the number of significant digits displayed.

Left-Justified

/ld Signed decimal number in the form `[[-]] #####.####`, where `####` is one or more decimal digits. The number of digits before the decimal point depends on the magnitude of the number, and the number of digits after the decimal point depends on the precision. If the precision is 0, no decimal point will be printed. If the number is positive, a space character will replace the leading minus sign.

/le Signed number in the form `[[-]] #.##E±###`, where `#` is one decimal digit, `##` is one or more decimal digits depending on the precision, and `###` is three decimal digits. If precision is 0, the form will be `[[-]] #E±###` with no decimal point printed. If the number is positive, a space character will replace the leading minus sign.

format

/lo This will give a format like **/ld** or **/le** depending on which is most compact for the number being printed. A format like **/le** will be used only if the exponent value is less than -4 or greater than the precision. If a **/le** format is used, a decimal point will always appear. If the number is positive, a space character will replace the leading minus sign. The precision specifies the number of significant digits displayed.

/lz This will give a format like **/ld** or **/le** depending on which is most compact for the number being printed. A format like **/le** will be used only if the exponent value is less than -4 or greater than the precision. If a **/le** format is used, trailing zeros will be suppressed and a decimal point will appear only if one or more digits follow it. If the number is positive, a space character will replace the leading minus sign. The precision specifies the number of significant digits displayed.

Trailing Character

The following characters can be added to the **/jnt** parameters above to control the trailing character if any:

`format /rdn 1,3;`

- s** The number will be followed immediately by a space character. This is the default.
- c** The number will be followed immediately by a comma.
- t** The number will be followed immediately by a tab character.
- n** No trailing character.

f scalar expression, controls the field width.
p scalar expression, controls the precision.

Remarks If character elements are to be printed, the precision should be at least 8 or the elements will be truncated. This does not affect the string data type.

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

format

For numeric values in matrices, p sets the number of significant digits to be printed. For string arrays, strings, and character elements in matrices, p sets the number of characters to be printed. If a string is shorter than the specified precision, the entire string is printed. For string arrays and strings, $p = -1$ means print the entire string, regardless of its length. $p = -1$ is illegal for matrices; setting $p \geq 8$ means the same thing for character elements.

The `/xxx` slash parameters are optional. Field and precision are optional also but if one is included, then both must be included.

Slash parameters, if present, must precede the field and precision parameters.

A **format** statement stays in effect until it is overridden by a new **format** statement. The slash parameters may be used in a **print** statement to override the current default.

f and p may be any legal expressions that return scalars. Nonintegers will be truncated to integers.

The total width of field will be overridden if the number is too big to fit into the space allotted. For instance, **format /rds 1,0** can be used to print integers with a single space between them, regardless of the magnitudes of the integers.

Complex numbers are printed with the sign of the imaginary half separating them and an “i” appended to the imaginary half. Also, the field parameter refers to the width of field for each half of the number, so a complex number printed with a field of 8 will actually take (at least) 20 spaces to print. The character printed after the imaginary part can be changed (for example, to a “j”) with the **sysstate** function, case 9.

The default when GAUSS is first started is:

```
format /mb1 /ros 16,8;
```

Example This code:

```
x = rndn(3,3);
```

```
format /m1 /rd 16,8;
```

```
print x;
```

produces:

```
-1.63533465    1.61350700    -1.06295179
 0.26171282    0.27972294    -1.38937242
 0.58891114    0.46812202     1.08805960
```

This code:

```
format /m1 /rzs 1,10;
```

```
print x;
```

produces:

```
-1.6353346    1.613507    -1.0629518
 0.26171282    0.27972294    -1.3893724
 0.58891114    0.46812202     1.0880596
```

This code:

```
format /m3 /rdn 16,4;
```

```
print x;
```

produces:

```
Row 1
      -1.6353  1.6135  -1.0630
Row 2
      0.2617  0.2797  -1.3894
Row 3
      0.5889  0.4681   1.0881
```

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

format

This code:

```
format /ml /ldn 16,4;
```

```
print x;
```

produces:

-1.6353	1.6135	-1.0630
0.2617	0.2797	-1.3894
0.5889	0.4681	1.0881

This code:

```
format /ml /res 12,4;
```

```
print x;
```

produces:

-1.6353E+000	1.6135E+000	-1.0630E+000
2.6171E-001	2.7972E-001	-1.3894E+000
5.8891E-001	4.6812E-001	1.0881E+000

See also **formatcv, formatnv, print, lprint, output**

formatcv

Purpose	Sets the character data format used by printfmt .
Format	<i>oldfmt</i> = formatcv (<i>newfmt</i>);
Input	<i>newfmt</i> 1x3 vector, the new format specification.
Output	<i>oldfmt</i> 1x3 vector, the old format specification.
Remarks	See printfm for details on the format vector.
Example	<p>This example saves the old format, sets the format desired for printing <i>x</i>, prints <i>x</i>, then restores the old format. This code:</p> <pre> x = { A 1, B 2, C 3 }; oldfmt = formatcv("*. *s" ~ 3 ~ 3); call printfmt(x,0~1); call formatcv(oldfmt); </pre> <p>produces:</p> <pre> A 1 B 2 C 3 </pre>
Source	gauss.src
Globals	__fmtcv
See also	formatnv , printfm , printfmt

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

formatnv

formatnv

a	Purpose	Sets the numeric data format used by printfmt .
b	Format	<i>oldfmt</i> = formatnv (<i>newfmt</i>);
c	Input	<i>newfmt</i> 1x3 vector, the new format specification.
d	Output	<i>oldfmt</i> 1x3 vector, the old format specification.
e	Remarks	See printfm for details on the format vector.
f	Example	<p>This example saves the old format, sets the format desired for printing <i>x</i>, prints <i>x</i>, then restores the old format. This code:</p> <pre>x = { A 1, B 2, C 3 }; oldfmt = formatnv("*. *lf" ~ 8 ~ 4); call printfmt(x,0~1); call formatnv(oldfmt);</pre> <p>produces:</p> <pre>A 1 B 2 C 3</pre>
g	Source	gauss.src
h	Globals	__fmtnv
i	See also	formatcv , printfm , printfmt
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

fputs

Purpose	Writes strings to a file.	
Format	<i>numl</i> = fputs (<i>f</i> , <i>sa</i>);	
Input	<i>f</i>	scalar, file handle of a file opened with fopen .
	<i>sa</i>	string or string array.
Output	<i>numl</i>	scalar, the number of lines written to the file.
Portability	UNIX Carriage return-linefeed conversion for files opened in text mode is unnecessary, because in UNIX a newline is simply a linefeed.	
Remarks	fputs writes the contents of each string in <i>sa</i> , minus the null terminating byte, to the file specified. If the file was opened in text mode (see fopen), any newlines present in the strings are converted to carriage return-linefeed sequences on output. If <i>numl</i> is not equal to the number of elements in <i>sa</i> , there may have been an I/O error while writing the file. You can use fcheckerr or fclearerr to check this. If there was an error, you can call fstrerror to find out what it was. If the file was opened for update (see fopen) and you are switching from reading to writing, don't forget to call fseek or fflush first, to flush the file's buffer. If you pass fputs the handle of a file opened with open (i.e., a data set or matrix file), your program will terminate with a fatal error.	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

fputst

fputst

a		
b		
c		
d		
e		
f	Purpose	Writes strings to a file.
g	Format	<i>numl</i> = fputst (<i>f</i> , <i>sa</i>);
h	Input	<i>f</i> scalar, file handle of a file opened with fopen . <i>sa</i> string or string array.
i	Output	<i>numl</i> scalar, the number of lines written to the file.
j	Portability	UNIX Carriage return-linefeed conversion for files opened in text mode is unnecessary, because in UNIX a newline is simply a linefeed.
k	Remarks	fputst works identically to fputs , except that a newline is appended to each string that is written to the file. If the file was opened in text mode (see fopen), these newlines are also converted to carriage return-linefeed sequences on output.
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

fseek

Purpose	Positions the file pointer in a file.
Format	<code>ret = fseek(<i>f</i>,<i>offs</i>,<i>base</i>);</code>
Input	<p><i>f</i> scalar, file handle of a file opened with fopen.</p> <p><i>offs</i> scalar, offset (in bytes).</p> <p><i>base</i> scalar, base position.</p> <p>0 beginning of file.</p> <p>1 current position of file pointer.</p> <p>2 end of file.</p>
Output	<i>ret</i> scalar, 0 if successful, 1 if not.
Portability	<p>UNIX</p> <p>Carriage return-linefeed conversion for files opened in text mode is unnecessary, because in UNIX a newline is simply a linefeed.</p>
Remarks	<p>fseek moves the file pointer <i>offs</i> bytes from the specified <i>base</i> position. <i>offs</i> can be positive or negative. The call may fail if the file buffer needs to be flushed (see fflush).</p> <p>If fseek fails, you can call fstrerror to find out why.</p> <p>For files opened for update (see fopen), the next operation can be a read or a write.</p> <p>fseek is not reliable when used on files opened in text mode (see fopen). This has to do with the conversion of carriage return-linefeed sequences to newlines. In particular, an fseek that follows one of the fgets<i>xx</i> or fputs<i>xx</i> commands may not produce the expected result. For example:</p> <pre>p = ftell(f); s = fgets(a,f,7); call fseek(f,p,0);</pre> <p>is not reliable. The best results are obtained by fseek'ing to the beginning of the file and then fseek'ing to the desired location, as in</p>

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

fseek

```
p = ftell(f);  
s = fgetsa(f,7);  
call fseek(f,0,0);  
call fseek(f,p,0);
```

If you pass **fseek** the handle of a file opened with **open** (i.e., a data set or matrix file), your program will terminate with a fatal error.

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

fstrerror

Purpose	Returns an error message explaining the cause of the most recent file I/O error.
Format	<code>s = fstrerror;</code>
Output	<code>s</code> string, error message.
Remarks	<p>Any time an I/O error occurs on a file opened with fopen, an internal error flag is updated. (This flag, unlike those accessed by fcheckerr and fclearerr, is not specific to a given file; rather, it is system-wide.) fstrerror returns an error message based on the value of this flag, clearing it in the process. If no error has occurred, a null string is returned.</p> <p>Since fstrerror clears the error flag, if you call it twice in a row, it will always return a null string the second time.</p>

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

ftell

ftell

a	Purpose	Gets the position of the file pointer in a file.
b	Format	<i>pos</i> = ftell (<i>f</i>);
c	Input	<i>f</i> scalar, file handle of a file opened with fopen .
d	Output	<i>pos</i> scalar, current position of the file pointer in a file.
e	Remarks	ftell returns the position of the file pointer in terms of bytes from the beginning of the file. The call may fail if the file buffer needs to be flushed (see fflush). If an error occurs, ftell returns -1. You can call fstrerror to find out what the error was. If you pass ftell the handle of a file opened with open (i.e., a data set or matrix file), your program will terminate with a fatal error.
f		
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

ftocv

Purpose	Converts a matrix containing floating point numbers into a matrix containing the decimal character representation of each element.	
Format	$y = \text{ftocv}(x, \text{field}, \text{prec});$	
Input	x	NxK matrix containing numeric data to be converted.
	field	scalar, minimum field width.
	prec	scalar, the numbers created will have prec places after the decimal point.
Output	y	NxK matrix containing the decimal character equivalent of the corresponding elements in x in the format defined by field and prec .
Remarks	If a number is narrower than field , it will be padded on the left with zeros. If $\text{prec} = 0$, the decimal point will be suppressed.	
Example	$y = \text{seqa}(6, 1, 5);$ $x = 0 \ \$+ \ \text{"cat"} \ \$+ \ \text{ftocv}(y, 2, 0);$	
	cat06 cat07 $x = \text{cat08}$ cat09 cat10 Notice that the ($0 \ \$+ \$) above was necessary to force the type of the result to matrix because the string constant cat would be of type string. The left operand in an expression containing a $\$+$ operator controls the type of the result.	
See also	ftos	

ftos

ftos

Purpose	Converts a scalar into a string containing the decimal character representation of that number.		
Format	$y = \text{ftos}(x, \text{fmat}, \text{field}, \text{prec}) ;$		
Input	x	scalar, the number to be converted.	
	fmat	string, the format string to control the conversion.	
	field	scalar or 2x1 vector, the minimum field width. If field is 2x1, it specifies separate field widths for the real and imaginary parts of x .	
	prec	scalar or 2x1 vector, the number of places following the decimal point. If prec is 2x1, it specifies separate precisions for the real and imaginary parts of x .	
Output	y	string containing the decimal character equivalent of x in the format specified.	
Remarks	The format string corresponds to the format /jnt (justification, notation, trailing character) slash parameter as follows: /rdn "%*.*1f" /ren "%*.*1E" /ron "%#*.*1G" /rzn "%*.*1G" /ldn "%- *.*1f" /len "%- *.*1E" /lon "%-# *.*1G" /lzn "%- *.*1G" If x is complex, you can specify separate formats for the real and imaginary parts by putting two format specifications in the format string. You can also specify separate fields and precisions. You can position the sign of the imaginary part by placing a "+" between the two format specifications. If you use two formats, no "i" is appended to the imaginary		

part. This is so you can use an alternate format if you prefer, for example, prefacing the imaginary part with a “j”.

The format string can be a maximum of 80 characters.

If you want special characters to be printed after *x*, include them as the last characters of the format string. For example:

`“%*. *lf,”` right-justified decimal followed by a comma.

`“%-*. *s”` left-justified string followed by a space.

`“%*. *lf”` right-justified decimal followed by nothing.

You can embed the format specification in the middle of other text.

`“Time: %*. *lf seconds.”`

If you want the beginning of the field padded with zeros, then put a “0” before the first “*” in the format string:

`“%0*. *lf”` right-justified decimal.

If *prec* = 0, the decimal point will be suppressed.

Example You can create custom formats for complex numbers with **ftos**. For example,

```
let c = 24.56124+6.3224e-2i;
```

```
field = 1;
```

```
prec = 3|5;
```

```
fmat = “%lf + j%le is a complex number.”;
```

```
cc = ftos(c,fmat,field,prec);
```

results in

```
cc = “24.561 + j6.32240e-02 is a complex
number.”
```

Some other things you can do with **ftos**:

```
let x = 929.857435324123;
```

```
let y = 5.46;
```

```
let z = 5;
```

ftos

```
field = 1;
prec = 0;
fmat = "%*.*lf";
zz = ftos(z,fmat,field,prec);

field = 1;
prec = 10;
fmat = "%*.*lE";
xx = ftos(x,fmat,field,prec);

field = 7;
prec = 2;
fmat = "%*.*lf seconds";
s1 = ftos(x,fmat,field,prec);
s2 = ftos(y,fmat,field,prec);

field = 1;
prec = 2;
fmat = "The maximum resistance is %*.*lf
      ohms.";
om = ftos(x,fmat,field,prec);
```

The results:

```
zz = "5"
xx = "9.2985743532E+02"
s1 = " 929.86 seconds"
s2 = "    5.46 seconds"
om = "The maximum resistance is 929.86 ohms."
```


See also **ftocv**, **stof**, **format**

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

ftostrC

ftostrC

a	Purpose	Converts a matrix to a string array using a C language format specification.
b		
c	Format	<i>sa</i> = ftostrC (<i>x</i> , <i>fmt</i>);
d	Input	<i>x</i> NxK matrix, real or complex. <i>fmt</i> Kx1, 1xK or 1x1 string array containing format information.
e		
f	Output	<i>sa</i> NxK string array.
g		
h	Remarks	If <i>fmt</i> has K elements, each column of <i>sa</i> can be formatted separately. If <i>x</i> is complex, there must be two format specifications in each element of <i>fmt</i> .
i		
j	Example	declare string fmtr = { "%6.3lf", "%11.8lf" }; declare string fmtc = { "(%6.3lf, %6.3lf)", "(%11.8lf, %11.8lf)" }; xr = rndn(4, 2); xc = sqrt(xr')'; sar = ftostrC(xr, fmtr); sac = ftostrC(xc, fmtc);
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

```
print sar;
```

```
print sac;
```

produces:

```
-0.166      1.05565441
```

```
-1.590      -0.79283296
```

```
 0.130      -1.84886957
```

```
 0.789      0.86089687
```

```
( 0.000, -0.407)      ( 1.02745044,  0.00000000)
```

```
( 0.000, -1.261)      ( 0.00000000, -0.89041168)
```

```
( 0.361,  0.000)      ( 0.00000000, -1.35973143)
```

```
( 0.888,  0.000)      ( 0.92784529,  0.00000000)
```

See also **strtof**, **strtofcp1x**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

gamma

gamma

a	Purpose	Returns the value of the gamma function.
b	Format	$y = \text{gamma}(x);$
c	Input	x NxK matrix.
d	Output	y NxK matrix.
e	Remarks	For each element of x , this function returns the integral
f		$\int_0^{\infty} t^{(x-1)} e^{-t} dt$
g		All elements of x must be positive and less than or equal to 169. Values of x greater than 169 will cause an overflow.
h		The natural log of gamma is often what is required and it can be computed without the overflow problems of gamma . lnfact can be used to compute log gamma.
i	Example	$y = \text{gamma}(2.5);$ $y = 1.32934$
j	See also	cdfchic, cdfbeta, cdffc, cdfn, cdfnc, cdftc, erf, erfc
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

gammaii

Purpose	Computes the inverse incomplete gamma function.		
Format	$x = \text{gammaii}(a,p);$		
Input	a	MxN matrix, exponents.	
	p	KxL matrix, EXE conformable with a , incomplete gamma values.	
Output	x	max(M,K) by max(N,L) matrix, abscissae.	
Source	cdfchii.src		
Globals	_ginvinc, __macheps		

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

gausset

a	
b	
c	
d	
e	
f	
g	
h	
i	
j	
k	
l	
m	
n	
o	
p	
q	
r	
s	
t	
u	
v	
w	
x y z	

Purpose	Resets the global control variables declared in <code>gauss.dec</code> .
Format	<code>gausset;</code>
Source	<code>gauss.src</code>
Globals	<code>__altnam, __con, __ff, __fmtcv, __fmtnv,</code> <code>__header, __miss, __output, __row, __rowfac,</code> <code>__sort, __title, __tol, __vpad, __vtype, __weight</code>

getf

Purpose	Loads an ASCII or binary file into a string.
Format	<code>y = getf(filename,mode);</code>
Input	<p><i>filename</i> string, any valid file name.</p> <p><i>mode</i> scalar 1 or 0 which determines if the file is to be loaded in ASCII mode (0) or binary mode (1).</p>
Output	<i>y</i> string containing the file.
Remarks	<p>If the file is loaded in ASCII mode, it will be tested to see if it contains any end of file characters. These are ^Z (ASCII 26). The file will be truncated before the first ^Z and there will be no ^Z's in the string. This is the correct way to load most text files because the ^Z's can cause problems when trying to print the string to a printer.</p> <p>If the file is loaded in binary mode, it will be loaded just like it is with no changes.</p>
Example	<p>Create a file <code>examp.e</code> containing the following program.</p> <pre>library pgraph; graphset; x = seqa(0,0.1,100); y = sin(x); xy(x,y);</pre> <p>Then execute the following.</p> <pre>y = getf("examp.e",0); print y;</pre>

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

getf

This produces:

```
library pgraph;  
graphset;  
x = seqa(0,0.1,100);  
y = sin(x);  
xy(x,y);
```

See also **load, save, let, con**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

getname

Purpose	Returns a column vector containing the names of the variables in a GAUSS data set.	
Format	<code>y = getname(dset);</code>	
Input	<i>dset</i>	string specifying the name of the data set from which the function will obtain the variable names.
Output	<i>y</i>	Nx1 vector containing the names of all of the variables in the specified data set.
Remarks	The output, <i>y</i> , will have as many rows as there are variables in the data set.	
Example	<pre> y = getname("olsdat"); format 8,8; print \$y; produces: TIME DIST TEMP FRICT The above example assumes the data set olsdat contained the variables <i>TIME</i>, <i>DIST</i>, <i>TEMP</i>, <i>FRICT</i>. Note that the extension is not included in the filename passed to the getname function. </pre>	
See also	getnamef , indcv	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

getnamef

getnamef

Purpose	Returns a string array containing the names of the variables in a GAUSS data set.
Format	<code>y = getnamef(f);</code>
Input	<i>f</i> scalar, file handle of an open data set.
Output	<i>y</i> Nx1 string array containing the names of all of the variables in the specified data set.
Remarks	The output, <i>y</i> , will have as many rows as there are variables in the data set.
Example	<pre>open f = olsdat for read; y = getnamef(f); t = vartypef(f); print y;</pre> <p>produces:</p> <pre>time dist temp frict</pre> <p>The above example assumes the data set olsdat contained the variables <i>time</i>, <i>dist</i>, <i>temp</i>, <i>frict</i>.</p> <p>Note the use of vartypef to determine the types of these variables.</p>
See also	getname , indcv , vartypef

getNextTradingDay

Purpose	Returns the next trading day.
Format	$n = \text{getNextTradingDay}(a)$
Input	a scalar, date in DT scalar format.
Output	n next trading day in DT scalar format
Remarks	A trading day is a weekday that is not a holiday as defined by the New York Stock Exchange from 1888 through 2004. Holidays are defined in <code>holidays.asc</code> . You may edit that file to modify or add holidays.
Source	<code>finutils.src</code>

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

getNextWeekDay

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

Purpose	Returns the next day that is not on a weekend.
Format	$n = \text{getNextWeekDay}(a)$
Input	a scalar, date in DT scalar format.
Output	n next week day in DT scalar format
Source	finutils.src

getnr

Purpose	Computes number of rows to read per iteration for a program that reads data from a disk file in a loop.	
Format	<code>nr = getnr(nsets,ncols);</code>	
Input	<i>nsets</i>	scalar, estimate of the maximum number of duplicate copies of the data matrix read by readr to be kept in memory during each iteration of the loop.
	<i>ncols</i>	scalar, columns in the data file.
Output	<i>nr</i>	scalar, number of rows readr should read per iteration of the read loop.
Remarks	If __row is greater than 0, <i>nr</i> will be set to __row .	
	If an insufficient memory error is encountered, change __rowfac to a number less than 1.0 (e.g., 0.75). The number of rows read will be reduced in size by this factor.	
Source	gauss.src	
Globals	__row , __rowfac	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

getpath

getpath

Purpose	Returns an expanded filename including the drive and path.
Format	<i>fname</i> = getpath (<i>pfname</i>) ;
Input	<i>pfname</i> string, partial filename with only partial or missing path information.
Output	<i>fname</i> string, filename with full drive and path.
Remarks	This function handles relative path references.
Example	<pre>y = getpath("temp.e"); print y; produces: /gauss/temp.e</pre>
Source	getpath.src

getPreviousTradingDay

Purpose Returns the previous trading day.

Format $n = \text{getPreviousTradingDay}(a)$

Input a scalar, date in DT scalar format.

Output n Previous trading day in DT scalar format

Remarks A trading day is a weekday that is not a holiday as defined by the New York Stock Exchange from 1888 through 2004. Holidays are defined in `holidays.asc`. You may edit that file to modify or add holidays.

Source `finutils.src`

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

getPreviousWeekDay

getPreviousWeekDay

Purpose	Returns the previous day that is not on a weekend.
Format	$n = \text{getPreviousWeekDay}(a)$
Input	a scalar, date in DT scalar format.
Output	n previous week day in DT scalar format
Source	finutils.src

getwind

Purpose	Retrieves the current graphic panel number.
Library	pgraph
Format	<i>n</i> = getwind ;
Output	<i>n</i> scalar, graphic panel number of current graphic panel.
Remarks	The current graphic panel is the graphic panel in which the next graph will be drawn.
Source	pwindow.src
See also	endwind , begwind , window , setwind , nextwind

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

gosub

gosub

a		
b	Purpose	Causes a branch to a subroutine.
c	Format	gosub <i>label</i> ;
d		.
e		.
f		.
g		<i>label</i> :
h		
i		.
j		.
k		.
l		
m		return ;
n	Remarks	For multi-line recursive user-defined functions, see “Procedures and Keywords” in the <i>User’s Guide</i> .
o		When a gosub statement is encountered, the program will branch to the label and begin executing from there. When a return statement is encountered, the program will resume executing at the statement following the gosub statement. Labels are 1-32 characters long and are followed by a colon. The characters can be A-Z or 0-9 and they must begin with an alphabetic character. Uppercase or lowercase is allowed.
p		It is possible to pass parameters to subroutines and receive parameters from them when they return. See the second example, following.
q		The only legal way to enter a subroutine is with a gosub statement.
r		If your subroutines are at the end of your program, you should have an end statement before the first one to prevent the program from running into a subroutine without using a gosub . This will result in a “return without gosub” error message.
s		
t		
u		
v		
w		
x y z		

gosub

The variables used in subroutines are not local to the subroutine and can be accessed from other places in your program. (See “Procedures and Keywords” in the *User’s Guide*.)

Example

In the program below, the name **mysub** is a label. When the **gosub** statement is executed, the program will jump to the label **mysub** and continue executing from there. When the **return** statement is executed, the program will resume executing at the statement following the **gosub**.

```
x = rndn(3,3); z = 0;
```

```
gosub mysub;
```

```
print z;
```

```
end;
```

```
/* ----- Subroutines Follow ----- */
```

```
mysub:
```

```
    z = inv(x);
```

```
    return;
```

Parameters can be passed to subroutines in the following way (line numbers are added for clarity):

```
1. gosub mysub(x,y);
```

```
2. pop j; /* b will be in j */
```

```
3. pop k; /* a will be in k */
```

```
4. t = j*k;
```

```
5. print t;
```

```
6. end;
```

```
7.
```

```
8. /* ---- Subroutines Follow ---- */
```

```
9.
```

```
10. mysub:
```

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

gosub

```
11.    pop b; /* y will be in b */
12.    pop a; /* x will be in a */
13.
14.    a = inv(b)*b+a;
15.    b = a'b;
16.    return(a,b);
```

In the previous example, when the **gosub** statement is executed, the following sequence of events results:

1. **x** and **y** are pushed on the stack and the program branches to the label **mysub** in line 10.
11. the second argument that was pushed, **y**, is **pop**'ped into **b**.
12. the first argument that was pushed, **x**, is **pop**'ped into **a**.
14. **inv(b)*b+a** is assigned to **a**.
15. **a/b** is assigned to **b**.
16. **a** and **b** are pushed on the stack and the program branches to the statement following the **gosub**, which is line 2.
2. the second argument that was pushed, **b**, is **pop**'ped into **j**.
3. the first argument that was pushed, **a**, is **pop**'ped into **k**.
4. **j*k** is assigned to **t**.
5. **t** is printed.
6. the program is terminated with the **end** statement.

Matrices are pushed on a last-in/first-out stack in the **gosub()** and **return()** statements. They must be popped off in the reverse order. No intervening statements are allowed between the label and the **pop** or the **gosub** and the **pop**. Only one matrix may be popped per **pop** statement.

See also **goto, proc, pop, return**

goto

Purpose	Causes a branch to a label.
Format	goto <i>label</i> ; . . . <i>label</i> :
Remarks	<p>Label names can be any legal GAUSS names up to 32 alphanumeric characters, beginning with an alphabetic character or an underscore, not a reserved word.</p> <p>Labels are always followed immediately by a colon.</p> <p>Labels do not have to be declared before they are used. GAUSS knows they are labels by the fact that they are followed immediately by a colon.</p> <p>When GAUSS encounters a goto statement, it jumps to the specified label and continues execution of the program from there.</p> <p>Parameters can be passed in a goto statement the same way as they can with a gosub.</p>
Example	<pre>x = seqa(.1,.1,5); n = { 1 2 3 }; goto fip; print x; end; fip: print n; produces: 1.0000000 2.0000000 3.0000000</pre>

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

goto

See also **gosub, if**

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

gradp

Purpose	Computes the gradient vector or matrix (Jacobian) of a vector-valued function that has been defined in a procedure. Single-sided (forward difference) gradients are computed.		
Format	$g = \text{gradp}(\&f, x0);$		
Input	$\&f$	a pointer to a vector-valued function ($f: K \times 1 \rightarrow N \times 1$) defined as a procedure. It is acceptable for $f(x)$ to have been defined in terms of global arguments in addition to x , and thus f can return an $N \times 1$ vector: <pre>proc f(x); retp(exp(x.*b)); endp;</pre>	
	$x0$	$K \times 1$ vector of points at which to compute gradient.	
Output	g	$N \times K$ matrix containing the gradients of f with respect to the variable x at $x0$.	
Remarks	gradp will return a row for every row that is returned by f . For instance, if f returns a scalar result, then gradp will return a $1 \times K$ row vector. This allows the same function to be used regardless of N , where N is the number of rows in the result returned by f . Thus, for instance, gradp can be used to compute the Jacobian matrix of a set of equations.		
Example	<pre>proc myfunc(x); retp(x.*2 .* exp(x.*x./3)); endp; x0 = 2.5 3.0 3.5; y = gradp(&myfunc,x0); 82.98901842 0.00000000 0.00000000 y = 0.00000000 281.19752975 0.00000000 0.00000000 0.00000000 1087.95414117</pre>		

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

gradp

It is a 3X3 matrix because we are passing it 3 arguments and **myfunc** returns 3 results when we do that; the off-diagonals are zeros because the cross-derivatives of 3 arguments are 0.

Source `gradp.src`

See also **hessp**

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

graphprt

Purpose	Controls automatic printer hardcopy and conversion file output.
Library	pgraph
Format	graphprt (<i>str</i>) ;
Input	<i>str</i> string, control string.
Portability	UNIX Not supported, use WinPrintPQG instead.
Remarks	<p>graphprt is used to create hardcopy output automatically without user intervention. The input string <i>str</i> can have any of the following items, separated by spaces. If <i>str</i> is a null string, the interactive mode is entered. This is the default.</p> <ul style="list-style-type: none"> -P print graph. -PO=<i>c</i> set print orientation. <ul style="list-style-type: none"> L landscape. P portrait. -C=<i>n</i> convert to another file format. <ul style="list-style-type: none"> 1 Encapsulated PostScript file. 3 HPGL Plotter file. 5 BMP (Windows only) 8 WMF (Windows only) -CF=<i>name</i> set converted output file name. -I Minimize (iconize) the graphics window. -Q Close window after processing. -W=<i>n</i> display graph, wait <i>n</i> seconds, then continue. <p>If you are not using graphic panels, you can call graphprt anytime before the call to the graphics routine. If you are using graphic panels, call graphprt just before the endwind statement.</p> <p>The print option default values are set from the viewer application. Any parameters passed through graphprt will override the default values. (See “Publication Quality Graphics” in the <i>User’s Guide</i>.)</p> <p>Under DOS, this uses a utility called vwr.exe by default.</p>

graphprt

Example Automatic print using a single graphics call.

```
library pgraph;
graphset;
load x,y;
graphprt("-p"); /* tell "xy" to print */
xy(x,y);        /* create graph and print */
```

Automatic print using multiple graphics graphic panels. Note **graphprt** is called once just before the **endwind** call.

```
library pgraph;
graphset;
load x,y;
begwind;
window(1,2,0); /* create two windows */

setwind(1);
xy(x,y); /* first graphics call */

nextwind;
xy(x,y); /* second graphics call */

graphprt("-p");
endwind; /* print page containing all graphs */
```

The next example shows how to build a string to be used with **graphprt**.

```
library pgraph;
graphset;
load x,y;
```

graphprt

```
cvtnam = "mycvt.eps"; /* name of output file

/* concatenate options into one string */
cmdstr = "-c=1" $+ " -cf=" $+ cvtnam;
cmdstr = cmdstr $+ "n-q";

graphprt(cmdstr); /* tell "xy" to convert and
                  close*/
xy(x,y);          /* create graph and print */
```

The above string *cmdstr* will produce:

"-c = 1_n - cf = mycvt.eps_n - q"

Source pgraph.src

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

graphset

graphset

a	Purpose	Resets graphics globals to default values.
b	Library	pgraph
c	Format	graphset ;
d	Remarks	This procedure is used to reset the defaults between graphs. graphset may be called between each graphic panel to be displayed. To change the default values of the global control variables, make the appropriate changes in the file <code>pgraph.dec</code> and to the procedure graphset .
e	Source	<code>pgraph.src</code>
f		
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

hasimag

Purpose Tests whether the imaginary part of a complex matrix is negligible.

Format `y = hasimag(x);`

Input `x` N×K matrix.

Output `y` scalar, 1 if the imaginary part of `x` has any nonzero elements, 0 if it consists entirely of 0's.

The function **iscplx** tests whether `x` is a complex matrix or not, but it does not test the contents of the imaginary part of `x`. **hasimag** tests the contents of the imaginary part of `x` to see if it is zero.

hasimag actually tests the imaginary part of `x` against a tolerance to determine if it is negligible. The tolerance used is the imaginary tolerance set with the **sysstate** command, case 21.

Some functions are not defined for complex matrices. **iscplx** can be used to determine whether a matrix has no imaginary part and so can pass through those functions. **hasimag** can be used to determine whether a complex matrix has a negligible imaginary part and could thus be converted to a real matrix to pass through those functions.

iscplx is useful as a preliminary check because for large matrices it is much faster than **hasimag**.

Example

```
x = { 1 2 3i,
      4-i 5 6i,
      7 8i 9 };
y = hasimag(x);
y = 1.0000000
```

See also **iscplx**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

header

header

a			
b	Purpose	Prints a header for a report.	
c	Format	header (<i>prcnm</i> , <i>dataset</i> , <i>ver</i>) ;	
d	Input	<i>prcnm</i>	string, name of procedure that calls header .
e		<i>dataset</i>	string, name of data set.
f		<i>ver</i>	2x1 numeric vector, the first element is the major version
g			number of the program, the second element is the revision
h			number. Normally this argument will be the version/revision
i			global (__??_ver) associated with the module within
j			which header is called. This argument will be ignored if set to
k			0.
l	Global Input	__header	string, containing the letters:
m		t	title is to be printed
n		l	lines are to bracket the title
o		d	a date and time is to be printed
p		v	version number of program is printed
q		f	file name being analyzed is printed
r		__title	string, title for header.
s	Source	gauss.src	
t	Globals	__header , __title	
u			
v			
w			
x y z			

hess

Purpose Computes the Hessenberg form of a square matrix.

Format $\{ h, z \} = \text{hess}(x);$

Input x KxK matrix.

Output h KxK matrix, Hessenberg form.
 z KxK matrix, transformation matrix.

Remarks **hess** computes the Hessenberg form of a square matrix. The Hessenberg form is an intermediate step in computing eigenvalues. It also is useful for solving certain matrix equations that occur in control theory (see Van Loan, Charles F. "Using the Hessenberg Decomposition in Control Theory," *Algorithms and Theory in Filtering and Control*. Sorenson, D.C., and R.J. Wets, eds., Mathematical Programming Study No. 18, No. Holland, Amsterdam, 1982, 102-11).

z is an orthogonal matrix that transforms x into h and vice versa. Thus:

$$h = z' x z$$

and since z is orthogonal,

$$x = z h z'$$

x is reduced to upper Hessenberg form using orthogonal similarity transformations. This preserves the Frobenius norm of the matrix and the condition numbers of the eigenvalues.

hess uses the ORTRAN and ORTHES functions from EISPACK.

Example `let x[3,3] = 1 2 3
 4 5 6
 7 8 9;
 { h,z } = hess(x);`

```

           1.00000000  -3.59700730  -0.24806947
h =  -8.06225775  14.04615385   2.83076923
           0.00000000   0.83076923  -0.04615385
```

hess

$$z = \begin{bmatrix} 1.00000000 & 0.00000000 & 0.00000000 \\ 0.00000000 & -0.49613894 & -0.86824314 \\ 0.00000000 & -0.86824314 & 0.49613894 \end{bmatrix}$$

See also `schur`

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

hessp

Purpose	Computes the matrix of second partial derivatives (Hessian matrix) of a function defined as a procedure.	
Format	$h = \text{hessp}(\&f, x0);$	
Input	$\&f$	pointer to a single-valued function $f(x)$, defined as a procedure, taking a single $K \times 1$ vector argument ($f: K \times 1 \rightarrow 1 \times 1$); $f(x)$ may be defined in terms of global arguments in addition to x .
	$x0$	$K \times 1$ vector specifying the point at which the Hessian of $f(x)$ is to be computed.
Output	h	$K \times K$ matrix of second derivatives of f with respect to x at $x0$; this matrix will be symmetric.
Remarks	This procedure requires $K*(K+1)/2$ function evaluations. Thus if K is large, it may take a long time to compute the Hessian matrix.	
	No more than 3-4 digit accuracy should be expected from this function, though it is possible for greater accuracy to be achieved with some functions.	
	It is important that the function be properly scaled, in order to obtain greatest possible accuracy. Specifically, scale it so that the first derivatives are approximately the same size. If these derivatives differ by more than a factor of 100 or so, the results can be meaningless.	
Example	<pre> x = { 1, 2, 3 }; proc g(b); retp(exp(x'b)); endp; b0 = { 3, 2, 1 }; h = hessp(&g,b0); </pre>	

hessp

The resulting matrix of second partial derivatives of **g(b)** evaluated at **b=b0** is:

22027.12898372	44054.87238165	66083.36762901
44054.87238165	88111.11102645	132168.66742899
66083.36762901	132168.66742899	198256.04087836

Source `hessp.src`

See also **gradp**

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

hist

Purpose	Computes and graphs a frequency histogram for a vector. The actual frequencies are plotted for each category.		
Library	pgraph		
Format	{ <i>b,m,freq</i> } = hist(<i>x,v</i>);		
Input	<i>x</i>	Mx1 vector of data.	
	<i>v</i>	Nx1 vector, the breakpoints to be used to compute the frequencies, or scalar, the number of categories.	
Output	<i>b</i>	Px1 vector, the breakpoints used for each category.	
	<i>m</i>	Px1 vector, the midpoints of each category.	
	<i>freq</i>	Px1 vector of computed frequency counts.	
Remarks	If a vector of breakpoints is specified, a final breakpoint equal to the maximum value of <i>x</i> will be added if the maximum breakpoint value is smaller.		
	If a number of categories is specified, the data will be divided into <i>v</i> evenly spaced categories.		
	Each time an element falls into one of the categories specified in <i>b</i> , the corresponding element of <i>freq</i> will be incremented by one. The categories are interpreted as follows:		
	$\begin{array}{rclclcl} freq[1] & = & & x & \leq & b[1] \\ freq[2] & = & b[1] & < & x & \leq & b[2] \\ freq[3] & = & b[2] & < & x & \leq & b[3] \\ & & & & & & \\ & & & & & & \\ freq[P] & = & b[P-1] & < & x & \leq & b[P] \end{array}$		

hist

Example `library pgraph;`
 `x = rndn(5000,1);`
 `{ b,m,f } = hist(x,20);`

Source `phist.src`

See also **histp, histf, bar**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

histf

Purpose	Graphs a histogram given a vector of frequency counts.	
Library	pgraph	
Format	histf (<i>f</i> , <i>c</i>);	
Input	<i>f</i>	Nx1 vector, frequencies to be graphed.
	<i>c</i>	Nx1 vector, numeric labels for categories. If this is a scalar 0, a sequence from 1 to rows (<i>f</i>) will be created.
Remarks	The axes are not automatically labeled. Use xlabel for the category axis and ylabel for the frequency axis.	
Source	phist.src	
See also	hist , bar , xlabel , ylabel	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

histp

See also `hist`, `histf`, `bar`

hsec

Purpose	Returns the number of hundredths of a second since midnight.
Format	<code>y = hsec;</code>
Remarks	The number of hundredths of a second since midnight can also be accessed as the [4,1] element of the vector returned by the date function.
Example	<pre> x = rndu(100,100); ts = hsec; y = x*x; et = hsec-ts; </pre> <p>In this example, hsec is used to time a 100x100 multiplication in GAUSS. A 100x100 matrix, x, is created, and the current time, in hundredths of a second since midnight, is stored in the variable ts. Then the multiplication is carried out. Finally, ts is subtracted from hsec to give the time difference which is assigned to et.</p>
See also	date, time, timestr, ethsec, etstr

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

if

if

a		
b		
c		
d		
e		
f		
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

Purpose Controls program flow with conditional branching.

Format **if** *scalar_expression*;
list of statements;
elseif *scalar_expression*;
list of statements;
elseif *scalar_expression*;
list of statements;
else;
list of statements;
endif;

Remarks *scalar_expression* is any expression that returns a scalar. It is *TRUE* if it is not zero, and *FALSE* if it is zero.

A list of statements is any set of GAUSS statements.

GAUSS will test the expression after the **if** statement. If it is *TRUE* (nonzero), then the first list of statements is executed. If it is *FALSE* (zero), then GAUSS will move to the expression after the first **elseif** statement if there is one and test it. It will keep testing expressions and will execute the first list of statements that corresponds to a *TRUE* expression. If no expression is *TRUE*, then the list of statements following the **else** statement is executed. After the appropriate list of statements is executed, the program will go to the statement following the **endif** and continue on.

if statements can be nested.

One **endif** is required per **if** statement. If an **else** statement is used, there may be only one per **if** statement. There may be as many **elseif**'s as are required. There need not be any **elseif**'s or any **else** statement within an **if** statement.

Note the semicolon after the **else** statement.

Example `if x < 0;`
 `y = -1;`
 `elseif x > 0;`
 `y = 1;`
 `else;`
 `y = 0;`
 `endif;`

See also **do**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

imag

imag

Purpose	Returns the imaginary part of a matrix.
Format	$zi = \mathbf{imag}(x);$
Input	x NxK matrix.
Output	zi NxK matrix, the imaginary part of x .
Remarks	If x is real, zi will be an NxK matrix of zeros.
Example	$x = \begin{Bmatrix} 4i & 9 & 3, \\ 2 & 5-6i & 7i \end{Bmatrix};$ $y = \mathbf{imag}(x);$ $y = \begin{bmatrix} 4.0000000 & 0.0000000 & 0.0000000 \\ 0.0000000 & -6.0000000 & 7.0000000 \end{bmatrix}$
See also	complex, real

import

Purpose	Imports a spreadsheet, database, or ASCII file into a matrix.
Format	<code>{ <i>x</i>,<i>namel</i> } = import(<i>fname</i>,<i>range</i>,<i>sheet</i>);</code>
Input	<p><i>fname</i> string, file name with path.</p> <p><i>range</i> string, range of cells (spreadsheet only), field descriptor (ASCII files), or 0.</p> <p><i>sheet</i> scalar, sheet number of multi-sheet spreadsheet or 0.</p>
Output	<p><i>x</i> NxK matrix, imported data.</p> <p><i>namel</i> Kx1 vector, column headers.</p>
Portability	Windows only
Remarks	<p>If <i>sheet</i> is 0, import will import the first (possibly only) sheet of a spreadsheet.</p> <p>For spreadsheets, <i>range</i> indicates the columns to be imported, and can be expressed in the form B7:K243 or B7..K243. A <i>range</i> of 0 imports all data.</p> <p>For packed ASCII files, <i>range</i> is a descriptor that defines field name, type, width, and optionally precision. It is a single string of the form</p> <p style="text-align: center;"><i>name</i> <code>[[type]] fldspec</code> <code>[[name</code> <code>[[type]] fldspec . . .]]</code></p> <p>where</p> <p><i>name</i> is the column name for the field.</p> <p><i>type</i> is \$ for a character field, blank for a numeric field.</p> <p><i>fldspec</i> is either a column range (e.g., 5-8), a start column and field width (e.g., 5,4 or 5,4.2), or a field width only (e.g., 4 or 4.2). If only a field width is given, the start column is input from the previous field. If the field width is specified with a decimal (e.g., 4.2), then a decimal point will be inserted that many places in from the right edge of the field.</p> <p>A shorthand for a set of columns is available (e.g., x01-x09); the subsequent <i>type</i> and <i>fldspec</i> are applied to all columns in the set.</p>

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

import

Example `fname = "c:\\temp\\testdata.xls";`
 `{mmx, names} = import(fname, "A3:D12", 3);`

This will import the range of cells **A3** to **D12** of the third sheet of `testdata.xls` into the GAUSS matrix **mmx** with the column headings defined by **names**. Substituting a 0 or 1 for the **3** would have imported the first (possibly only) sheet.

```
fname = "c:\\temp\\tstdta.asc";  
range = "var1 $ 6,2 var2 8-15 var3 18,4.1 gnp 7  
cons 6.2 xs1-xs20 2";  
{x,names} = import(fname,range,0);
```

Here, a GAUSS matrix **x** is created from a packed ASCII data set, using the descriptor shown.

Globals `_dxaschdr, _dxftype, _dxmiss, _dxprcn, _dxprint,`
 `_dxtxdlim, _dxtype, _dxwidth, _dxwkshdr`

See also `export, exportf, importf`

importf

Purpose	Imports a spreadsheet, database, or ASCII file to a GAUSS data set.
Format	<i>ret</i> = importf (<i>fname</i> , <i>dset</i> , <i>range</i> , <i>sheet</i>);
Input	<i>fname</i> string, file name with optional path. <i>dset</i> string, name of data set with optional path. <i>range</i> string, range of cells (spreadsheet only), field descriptor (ASCII files), or 0. <i>sheet</i> scalar, sheet number of multi-sheet spreadsheet or 0.
Output	<i>ret</i> scalar, success flag, 1 if successful, 0 if not. <i>dset</i> GAUSS data set.
Portability	Windows only
Remarks	See import for details on the <i>range</i> and <i>sheet</i> parameters.
Example	<pre>fname = "c:\\temp\\testdata2.dbf"; dname = "c:\\gauss\\data\\mydata.dat"; call importf(dname, fname, 0, 0);</pre> <p>This will create a GAUSS data set <code>c:\gauss\data\mydata.dat</code> from the data in the dBase III file <code>c:\temp\testdata2.dbf</code>.</p>
Globals	_dxaschdr, _dxftype, _dxmiss, _dxprcn, _dxprint, _dxtxdlim, _dxtype, _dxwidth, _dxwkshdr
See also	export, exportf, import

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

#include

#include

a		
b	Purpose	Inserts code from another file into a GAUSS program.
c	Format	#include <i>filename</i> ; #include " <i>filename</i> ";
d	Remarks	<i>filename</i> can be any legitimate file name.
e		This command makes it possible to write a section of general-purpose code, and insert it into other programs.
f		The code from the #include 'd file is inserted literally as if it were merged into that place in the program with a text editor.
g		If a path is specified for the file, then no additional searching will be attempted if the file is not found.
h		If a path is not specified, the current directory will be searched first, then each directory listed in src_path . src_path is defined in <i>gauss.cfg</i> .
i		#include /gauss/myprog.prc; No additional search will be made if the file is not found.
j		#include myprog.prc; The directories listed in src_path will be searched for <i>myprog.prc</i> if the file is not found in the current directory.
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

#include

Example `#include "/gauss/inc/cond.inc";`

The command will cause the code in the file `cond.inc` to be merged into the current program at the point at which this statement appears.

See also **`run, #lineson`**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

indcv

indcv

a		
b		
c		
d		
e		
f		
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		
	Purpose	Checks one character vector against another and returns the indices of the elements of the first vector in the second vector.
	Format	<code>z = indcv(what,where);</code>
	Input	<i>what</i> Nx1 character vector which contains the elements to be found in vector <i>where</i> . <i>where</i> Mx1 character vector to be searched for matches to the elements of <i>what</i> .
	Output	<i>z</i> Nx1 vector of integers containing the indices of the corresponding element of <i>what</i> in <i>where</i> .
	Remarks	If no matches are found for any of the elements in <i>what</i> , then the corresponding elements in the returned vector are set to the GAUSS missing value code. Both arguments will be forced to uppercase before the comparison. If there are duplicate elements in <i>where</i> , the index of the first match will be returned.
	Example	<code>let what = AGE PAY SEX;</code> <code>let where = AGE SEX JOB "date" PAY;</code> <code>z = indcv(what,where);</code> <div><div><i>what</i> =</div><div>AGE PAY SEX</div></div> <div><div><i>where</i> =</div><div>AGE SEX JOB date PAY</div></div>

$$Z = \begin{matrix} 1 \\ 5 \\ 2 \end{matrix}$$

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

indexcat

indexcat

a			
b	Purpose	Returns the indices of the elements of a vector which fall into a specified category.	
c	Format	$y = \text{indexcat}(x,v);$	
d	Input	x	Nx1 vector.
e		v	scalar or 2x1 vector.
f			If scalar, the function returns the indices of all elements of x equal to v .
g			If 2x1, then the function returns the indices of all elements of x that fall into the range:
h			$v[1] < x \leq v[2]$.
i			If v is scalar, it can contain a single missing to specify the missing value as the category.
j	Output	y	Lx1 vector, containing the indices of the elements of x which fall into the category defined by v . It will contain error code 13 if there are no elements in this category.
k			
l			
m	Remarks	Use a loop to pull out indices of multiple categories.	
n	Example	let $x = 1.0 \ 4.0 \ 3.3 \ 4.2 \ 6.0 \ 5.7 \ 8.1 \ 5.5;$	
o		let $v = 4 \ 6;$	
p		$y = \text{indexcat}(x,v);$	
q			
r		1.0	
s		4.0	
t		3.3	
u		$x =$	4.2
v			6.0
w			5.7
x y z			8.1
			5.5

indexcat

$v = \begin{matrix} 4 \\ 6 \end{matrix}$

$y = \begin{matrix} 4 \\ 5 \\ 6 \\ 8 \end{matrix}$

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

indices

indices

a		
b		
c		
d		
e		
f		
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

Purpose	Processes a set of variable names or indices and returns a vector of variable names and a vector of indices.	
Format	{ <i>name</i> , <i>indx</i> } = indices (<i>dataset</i> , <i>vars</i>);	
Input	<i>dataset</i>	string, the name of the data set.
	<i>vars</i>	Nx1 vector, a character vector of names or a numeric vector of column indices. If scalar 0, all variables in the data set will be selected.
Output	<i>name</i>	Nx1 character vector, the names associated with <i>vars</i> .
	<i>indx</i>	Nx1 numeric vector, the column indices associated with <i>vars</i> .
Remarks	If an error occurs, indices will either return a scalar error code or terminate the program with an error message, depending on the trap state. If the low order bit of the trap flag is 0, indices will terminate with an error message. If the low order bit of the trap flag is 1, indices will return an error code. The value of the trap flag can be tested with trapchk ; the return from indices can be tested with scalerr . You only need to check one argument; they will both be the same. The following error codes are possible: <div><div>1</div>Can't open dataset.</div> <div><div>2</div>Index of variable out of range, or undefined data set variables.</div>	
Source	indices.src	

indices2

Purpose	Processes two sets of variable names or indices from a single file. The first is a single variable and the second is a set of variables. The first must not occur in the second set and all must be in the file.		
Format	{ <i>name1</i> , <i>indx1</i> , <i>name2</i> , <i>indx2</i> } = indices2 (<i>dataset</i> , <i>var1</i> , <i>var2</i>);		
Input	<i>dataset</i>	string, the name of the data set.	
	<i>var1</i>	string or scalar, variable name or index. This can be either the name of the variable, or the column index of the variable. If null or 0, the last variable in the data set will be used.	
	<i>var2</i>	Nx1 vector, a character vector of names or a numeric vector of column indices. If scalar 0, all variables in the data set except the one associated with <i>var1</i> will be selected.	
Output	<i>name1</i>	scalar character matrix containing the name of the variable associated with <i>var1</i> .	
	<i>indx1</i>	scalar, the column index of <i>var1</i> .	
	<i>name2</i>	Nx1 character vector, the names associated with <i>var2</i> .	
	<i>indx2</i>	Nx1 numeric vector, the column indices of <i>var2</i> .	
Remarks	<p>If an error occurs, indices2 will either return a scalar error code or terminate the program with an error message, depending on the trap state. If the low order bit of the trap flag is 0, indices2 will terminate with an error message. If the low order bit of the trap flag is 1, indices2 will return an error code. The value of the trap flag can be tested with trapchk; the return from indices2 can be tested with scalerr. You only need to check one argument; they will all be the same. The following error codes are possible:</p> <ol style="list-style-type: none">1 Can't open dataset.2 Index of variable out of range, or undefined data set variables.3 First variable must be a single name or index.4 First variable contained in second set.		

indices2

Source indices2.src

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

indnv

Purpose	Checks one numeric vector against another and returns the indices of the elements of the first vector in the second vector.	
Format	$z = \text{indnv}(\text{what}, \text{where});$	
Input	<i>what</i>	Nx1 numeric vector which contains the values to be found in vector <i>where</i> .
	<i>where</i>	Mx1 numeric vector to be searched for matches to the values in <i>what</i> .
Output	<i>z</i>	Nx1 vector of integers, the indices of the corresponding elements of <i>what</i> in <i>where</i> .
Remarks	If no matches are found for any of the elements in <i>what</i> , then those elements in the returned vector are set to the GAUSS missing value code. If there are duplicate elements in <i>where</i> , the index of the first match will be returned.	
Example	<pre>let what = 8 7 3; let where = 2 7 8 4 3; z = indnv(what,where); 8 what = 7 3 2 7 where = 8 4 3</pre>	

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

indrv

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

3

$z =$ 2

5

intgrat2

Purpose Integrates the following double integral, using user-defined functions f , g_1 and g_2 , and scalars a and b :

$$\int_a^b \int_{g_2(x)}^{g_1(x)} f(x, y) dy dx$$

Format `y = intgrat2(&f,xl,gl);`

Input

&f	scalar, pointer to the procedure containing the function to be integrated.
xl	2x1 or 2xN matrix, the limits of x . These must be scalar limits.
gl	2x1 or 2xN matrix of function pointers, the limits of y . For xl and gl , the first row is the upper limit and the second row is the lower limit. N integrations are computed.

Global Input

_intord	scalar, the order of the integration. The larger _intord , the more precise the final result will be. _intord may be set to 2, 3, 4, 6, 8, 12, 16, 20, 24, 32, 40. Default = 12.
_intrec	scalar. This variable is used to keep track of the level of recursion of intgrat2 and may start out with a different value if your program terminated inside of the integration function on a previous run. Always set _intrec explicitly to 0 before any call to intgrat2 .

Output

y	Nx1 vector of the estimated integral(s) of $f(x,y)$ evaluated between the limits given by xl and gl .
----------	---

Remarks The user-defined functions specified by f and gl must either

Return a scalar constant, OR

Return a vector of function values. **intgrat2** will pass to user-defined functions a vector or matrix for x and y and expect a vector or matrix to be returned. Use **.*** and **./** instead of ***** and **/**.

intgrat2

Example `proc f(x,y);`

`retp(cos(x) + 1).*(sin(y) + 1));`

`endp;`

`proc g1(x);`

`retp(sqrt(1-x^2));`

`endp;`

`proc g2(x);`

`retp(0);`

`endp;`

`x1 = 1|-1;`

`g0 = &g1|&g2;`

`_intord = 40;`

`_intrec = 0;`

`y = intgrat2(&f,x1,g0);`

This will integrate the function $f(x,y) = (\cos(x)+1)(\sin(y)+1)$ over the upper half of the unit circle. Note the use of the `.*` operator instead of just `*` in the definition of $f(x,y)$. This allows f to return a vector or matrix of function values.

Source `intgrat.src`

Globals `_intord, _intq12, _intq16, _intq2, _intq20,`
 `_intq24, _intq3, _intq32, _intq4, _intq40,`
 `_intq6, _intq8, _intrec`

See also `intgrat3, intquad1, intquad2, intquad3, intsimp`

intgrat3

Purpose Integrates the following triple integral, using user-defined functions and scalars for bounds:

$$\int_a^b \int_{g_2(x)}^{g_1(x)} \int_{h_2(x,y)}^{h_1(x,y)} f(x, y, z) dz dy dx$$

Format `y = intgrat3(&f,xl,gl,hl);`

Input **&f** scalar, pointer to the procedure containing the function to be integrated. *F* is a function of (x,y,z).

xl 2x1 or 2xN matrix, the limits of *x*. These must be scalar limits.

gl 2x1 or 2xN matrix of function pointers. These procedures are functions of *x*.

hl 2x1 or 2xN matrix of function pointers. These procedures are functions of *x* and *y*.

For *xl*, *gl*, and *hl*, the first row is the upper limit and the second row is the lower limit. N integrations are computed.

Global Input **_intord** scalar, the order of the integration. The larger **_intord**, the more precise the final result will be. **_intord** may be set to 2, 3, 4, 6, 8, 12, 16, 20, 24, 32, 40. Default = 12.

_intrec scalar. This variable is used to keep track of the level of recursion of **intgrat3** and may start out with a different value if your program terminated inside of the integration function on a previous run. Always set **_intrec** explicitly to 0 before any call to **intgrat3**.

Output **y** Nx1 vector of the estimated integral(s) of *f*(x,y,z) evaluated between the limits given by *xl*, *gl*, and *hl*.

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

integrat3

Remarks User-defined functions f , and those used in gl and hl , must either:

 Return a scalar constant, OR

 Return a vector of function values. **integrat3** will pass to user-defined functions a vector or matrix for x and y and expect a vector or matrix to be returned. Use **.*** and **./** operators instead of just ***** or **/**.

Example `proc f(x,y,z);`
 `retp(2);`
 `endp;`

`proc g1(x);`
 `retp(sqrt(25-x^2));`
 `endp;`

`proc g2(x);`
 `retp(-g1(x));`
 `endp;`

`proc h1(x,y);`
 `retp(sqrt(25 - x^2 - y^2));`
 `endp;`

`proc h2(x,y);`
 `retp(-h1(x,y));`
 `endp;`

intgrat3

```

x1 = 5|-5;
g0 = &g1|&g2;
h0 = &h1|&h2;
_intrec = 0;
_intord = 40;
y = intgrat3(&f,x1,g1,h1);

```

This will integrate the function $f(x,y,z) = 2$ over the sphere of radius 5. The result will be approximately twice the volume of a sphere of radius 5.

Source `intgrat.src`

Globals `_intord, _intq12, _intq16, _intq2, _intq20,`
`_intq24, _intq3, _intq32, _intq4, _intq40,`
`_intq6, _intq8, _intrec`

See also `intgrat2, intquad1, intquad2, intquad3, intsimp`

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

intquad1

intquad1

a			
b	Purpose	Integrates a specified function using Gauss-Legendre quadrature. A suite of upper and lower bounds may be calculated in one procedure call.	
c	Format	<code>y = intquad1(&f,xl);</code>	
d	Input	<code>&f</code>	scalar, pointer to the procedure containing the function to be integrated. This must be a function of <i>x</i> .
e		<code>xl</code>	2xN matrix, the limits of <i>x</i> . The first row is the upper limit and the second row is the lower limit. N integrations are computed.
f		<code>_intord</code>	scalar, the order of the integration. The larger <code>_intord</code> , the more precise the final result will be. <code>_intord</code> may be set to 2, 3, 4, 6, 8, 12, 16, 20, 24, 32, 40.
g			Default = 12.
h	Global Input	<code>_intord</code>	scalar, the order of the integration. The larger <code>_intord</code> , the more precise the final result will be. <code>_intord</code> may be set to 2, 3, 4, 6, 8, 12, 16, 20, 24, 32, 40.
i			Default = 12.
j	Output	<code>y</code>	Nx1 vector of the estimated integral(s) of <i>f(x)</i> evaluated between the limits given by <i>xl</i> .
k			
l	Remarks	The user-defined function <i>f</i> must return a vector of function values. <code>intquad1</code> will pass to the user-defined function a vector or matrix for <i>x</i> and expect a vector or matrix to be returned. Use the <code>.*</code> and <code>./</code> instead of <code>*</code> and <code>/</code> .	
m	Example	<code>proc f(x);</code>	
n		<code> retp(x.*sin(x));</code>	
o		<code>endp;</code>	
p			
q		<code>xl = 1 0;</code>	
r		<code>y = intquad1(&f,xl);</code>	
s			
t			
u			
v			
w			
x y z			

intquad1

This will integrate the function $f(x) = x\sin(x)$ between 0 and 1. Note the use of the `.*` instead of `*`.

Source `integral.src`

Globals `_intord, _intq12, _intq16, _intq2, _intq20,`
`_intq24, _intq3, _intq32, _intq4, _intq40,`
`_intq6, _intq8`

See also `intsimp, intquad2, intquad3, intgrat2, intgrat3`

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

intquad2

intquad2

Purpose	Integrates a specified function using Gauss-Legendre quadrature. A suite of upper and lower bounds may be calculated in one procedure call.		
Format	$y = \text{intquad2}(\&f,xl,yl);$		
Input	$\&f$	scalar, pointer to the procedure containing the function to be integrated.	
	xl	2x1 or 2xN matrix, the limits of x .	
	yl	2x1 or 2xN matrix, the limits of y .	
	For xl and yl , the first row is the upper limit and the second row is the lower limit. N integrations are computed.		
Global Input	$_intord$	global scalar, the order of the integration. The larger $_intord$, the more precise the final result will be. $_intord$ may be set to 2, 3, 4, 6, 8, 12, 16, 20, 24, 32, 40. Default = 12.	
	$_intrec$	global scalar. This variable is used to keep track of the level of recursion of intquad2 and may start out with a different value if your program terminated inside of the integration function on a previous run. Always set $_intrec$ explicitly to 0 before any calls to intquad2 .	
Output	y	Nx1 vector of the estimated integral(s) of $f(x,y)$ evaluated between the limits given by xl and yl .	
Remarks	<p>The user-defined function f must return a vector of function values. intquad2 will pass to user-defined functions a vector or matrix for x and y and expect a vector or matrix to be returned. Use $_*$ and $_/$ instead of $*$ and $/$.</p> <p>intquad2 will expand scalars to the appropriate size. This means that functions can be defined to return a scalar constant. If users write their functions incorrectly (using $*$ instead of $_*$, for example), intquad2 may not compute the expected integral, but the integral of a constant function.</p> <p>To integrate over a region which is bounded by functions, rather than just scalars, use intgrat2 or intgrat3.</p>		

intquad2

Example `proc f(x,y);`
 `retp(x.*sin(x+y));`
 `endp;`

`x1 = 1|0;`
`y1 = 1|0;`

`_intrec = 0;`
`y = intquad2(&f,x1,y1);`

This will integrate the function **`x.*sin(x+y)`** between **`x`** = 0 and 1, and between **`y`** = 0 and 1.

Source `integral.src`

Globals `_intord, _intq12, _intq16, _intq2, _intq20,`
 `_intq24, _intq3, _intq32, _intq4, _intq40,`
 `_intq6, _intq8, _intrec`

See also `intquad1, intquad3, intsimp, intgrat2, intgrat3`

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

intquad3

intquad3

Purpose	Integrates a specified function using Gauss-Legendre quadrature. A suite of upper and lower bounds may be calculated in one procedure call.		
Format	$y = \text{intquad3}(\&f,xl,yl,zl);$		
Input	$\&f$	scalar, pointer to the procedure containing the function to be integrated. f is a function of (x,y,z) .	
	xl	2x1 or 2xN matrix, the limits of x .	
	yl	2x1 or 2xN matrix, the limits of y .	
	zl	2x1 or 2xN matrix, the limits of z .	
	For xl , yl , and zl , the first row is the upper limit and the second row is the lower limit. N integrations are computed.		
Global Input	$_intord$	global scalar, the order of the integration. The larger $_intord$, the more precise the final result will be. $_intord$ may be set to 2, 3, 4, 6, 8, 12, 16, 20, 24, 32, 40. Default = 12.	
	$_intrec$	global scalar. This variable is used to keep track of the level of recursion of intquad3 and may start out with a different value if your program terminated inside of the integration function on a previous run. Always set $_intrec$ explicitly to 0 before any calls to intquad3 .	
Output	y	Nx1 vector of the estimated integral(s) of $f(x,y,z)$ evaluated between the limits given by xl , yl , and zl .	
Remarks	The user-defined function f must return a vector of function values. intquad3 will pass to the user-defined function a vector or matrix for x , y , and z and expect a vector or matrix to be returned. Use $_*$ and $_/$ instead of $*$ and $/$.		
	intquad3 will expand scalars to the appropriate size. This means that functions can be defined to return a scalar constant. If users write their functions incorrectly (using $*$ instead of $_*$, for example), intquad3 may not compute the expected integral, but the integral of a constant function.		

intquad3

To integrate over a region which is bounded by functions, rather than just scalars, use **intgrat2** or **intgrat3**.

Example

```
proc f(x,y,z);
    retp(x.*y.*z);
endp;
```

```
x1 = 1|0;
y1 = 1|0;
z1 = { 1 2 3, 0 0 0 };
```

```
_intrec = 0;
y = intquad3(&f,x1,y1,z1);
```

This will integrate the function $f(\mathbf{x}) = \mathbf{x} \cdot \mathbf{y} \cdot \mathbf{z}$ over 3 sets of limits, since **z1** is defined to be a 2x3 matrix.

Source

integral.src

Globals

```
_intord, _intq12, _intq16, _intq2, _intq20,
_intq24, _intq3, _intq32, _intq4, _intq40,
_intq6, _intq8, _intrec
```

See also

intquad1, **intquad2**, **intsimp**, **intgrat2**, **intgrat3**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

intrleav

intrleav

a		
b		
c		
d		
e		
f		
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

Purpose	Interleaves the rows of two files that have been sorted on a common variable, to give a single file sorted on that variable.	
Format	intrleav (<i>infile1</i> , <i>infile2</i> , <i>outfile</i> , <i>keyvar</i> , <i>keytyp</i>) ;	
Input	<i>infile1</i>	string, name of input file 1.
	<i>infile2</i>	string, name of input file 2.
	<i>outfile</i>	string, name of output file.
	<i>keyvar</i>	string, name of key variable, this is the column the files are sorted on.
	<i>keytyp</i>	scalar, data type of key variable.
	1	numeric key, ascending order
	2	character key, ascending order
	-1	numeric key, descending order
	-2	character key, descending order
Remarks	The two files MUST have exactly the same variables, i.e., the same number of columns AND the same variable names. They must both already be sorted on the key column. This procedure will combine them into one large file, sorted by the key variable.	
	If the inputs are null or 0, the procedure will ask for them.	
Example	intrleav("freq.dat","freqata.dat", "intfile","AGE",1);	
Source	sortd.src	

intrsect

Purpose	Returns the intersection of two vectors, with duplicates removed.
Format	<code>y = intrsect(v1,v2,flag);</code>
Input	<p><i>v1</i> Nx1 vector.</p> <p><i>v2</i> Mx1 vector.</p> <p><i>flag</i> scalar; if 1, <i>v1</i>, and <i>v2</i> are numeric, if 0, character.</p>
Output	<i>y</i> Lx1 vector containing all unique values that are in both <i>v1</i> and <i>v2</i> , sorted in ascending order.
Remarks	<p>Place smaller vector first for fastest operation.</p> <p>If there are a lot of duplicates within a vector, it is faster to remove them with unique before calling intrsect.</p>
Source	<code>intrsect.src</code>
Example	<pre>let v1 = mary jane linda dawn; let v2 = mary sally lisa ruth linda; y = intrsect(v1,v2,0);</pre> <p><i>y</i> = <i>LINDA</i> <i>MARY</i></p>

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

intsimp

intsimp

Purpose	Integrates a specified function using Simpson's method with end correction. A single integral is computed in one function call.		
Format	<code>y = intsimp(&f,xl,tol);</code>		
Input	<code>&f</code>	pointer to the procedure containing the function to be integrated.	
	<code>xl</code>	2x1 vector, the limits of x . The first element is the upper limit and the second element is the lower limit.	
	<code>tol</code>	The tolerance to be used in testing for convergence.	
Output	<code>y</code>	The estimated integral of $f(x)$ between $xl[1]$ and $xl[2]$.	
Example	<pre>proc f(x); retp(sin(x)); endp; let xl = { 1, 0 }; y = intsimp(&f,xl,1E-8); y = 0.45969769 This will integrate the function between 0 and 1.</pre>		
Source	<code>intsimp.src</code>		
See also	<code>intquad1, intquad2, intquad3, intgrat2, intgrat3</code>		

inv, invpd

Purpose	inv returns the inverse of an invertible matrix. invpd returns the inverse of a symmetric, positive definite matrix.													
Format	$y = \text{inv}(x);$ $y = \text{invpd}(x);$													
Input	x	NxN matrix.												
Output	y	NxN matrix containing the inverse of x .												
Remarks	x can be any legitimate matrix expression that returns a matrix that is legal for the function. For inv , x must be square and invertible. For invpd , x must be symmetric and positive definite. If the input matrix is not invertible by these functions, they will either terminate the program with an error message or return an error code which can be tested for with the scalerr function. This depends on the trap state as follows: <table><tr><td colspan="2">trap 1, return error code</td></tr><tr><td>inv</td><td>invpd</td></tr><tr><td>50</td><td>20</td></tr></table> <table><tr><td colspan="2">trap 0, terminate with error message</td></tr><tr><td>inv</td><td>invpd</td></tr><tr><td>Matrix singular</td><td>Matrix not positive definite</td></tr></table> If the input to invpd is not symmetric, it is possible that the function will (erroneously) appear to operate successfully. Positive definite matrices can be inverted by inv . However, for symmetric, positive definite matrices (such as moment matrices), invpd is about twice as fast as inv .		trap 1 , return error code		inv	invpd	50	20	trap 0 , terminate with error message		inv	invpd	Matrix singular	Matrix not positive definite
trap 1 , return error code														
inv	invpd													
50	20													
trap 0 , terminate with error message														
inv	invpd													
Matrix singular	Matrix not positive definite													

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

inv, invpd

Example `n = 4000;`
 `x1 = rndn(n,1);`
 `x = ones(n,1)~x1;`
 `btrue = { 1, 0.5 };`
 `y = x*btrue + rndn(n,1);`
 `bols = invpd(x'x)*x'y;`
 `bols = 1.017201 0.484244`

This example simulates some data and computes the **ols** coefficient estimator using the **invpd** function. First, the number of observations is specified. Second, a vector **x1** of standard Normal random variables is generated and is concatenated with a vector of 1's (to create a constant term). The true coefficients are specified, and the dependent variable **y** is created. Then the **ols** coefficient estimates are computed.

See also **scalerr, trap, prcsn**

Technical For complex matrices, **inv** uses the ZGECO, ZGEDI path in the
Notes LINPACK routines. For real matrices, it uses the **croutp** function.

The **inv** function uses the Crout decomposition. The advantage of this routine is that on some platforms it allows most of the intermediate results to be computed in extended precision.

The **invpd** function uses the Cholesky decomposition and is based upon the LINPACK routines for positive definite matrices. On OS/2 and DOS, if **prcsn** 80 is in effect, all intermediate calculations and intermediate results will be in the 80-bit extended precision of the 80x87 temporary real format. The final results will be rounded to 64-bit double precision.

The tolerance used to determine singularity is 1.0e-14. This can be changed. See "Singularity Tolerance" in the *User's Guide*.

invswp

Purpose	Computes a generalized sweep inverse.		
Format	$y = \text{invswp}(x);$		
Input	x	NxN matrix.	
Output	y	NxN matrix, the generalized inverse of x .	
Remarks	<p>This will invert any general matrix. That is, even matrices which will not invert using inv because they are singular will invert using invswp.</p> <p>x and y will satisfy the four Moore-Penrose conditions:</p> <ol style="list-style-type: none">1. $xyx = x$2. $yxy = y$3. xy is symmetric4. yx is symmetric <p>The tolerance used to determine if a pivot element is zero is taken from the crout singularity tolerance. The corresponding row and column are zeroed out. See “Appendix C” in the <i>User’s Guide</i>.</p>		
Example	<pre>let x[3,3] = 1 2 3 4 5 6 7 8 9; y = invswp(x);</pre> $y = \begin{bmatrix} -1.6666667 & 0.6666667 & 0.0000000 \\ 1.3333333 & -0.3333333 & 0.0000000 \\ 0.0000000 & 0.0000000 & 0.0000000 \end{bmatrix}$		

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

iscplx

iscplx

a		
b		
c		
d		
e		
f		
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		
	Purpose	Returns whether a matrix is complex or real.
	Format	$y = \text{iscplx}(x);$
	Input	x NxK matrix.
	Output	y scalar, 1 if x is complex, 0 if it is real.
	Example	$x = \{ 1, 2i, 3 \};$ $y = \text{iscplx}(x);$
		$y = 1$
	See also	hasimag, iscplx

iscplxf

Purpose	Returns whether a data set is complex or real.
Format	<code>y = iscplxf(fh);</code>
Input	<i>fh</i> scalar, file handle of an open file.
Output	<i>y</i> scalar, 1 if the data set is complex, 0 if it is real.
See also	hasimag , iscplx

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

isinfnanmiss

isinfnanmiss

Purpose	Returns true if the argument contains an infinity, NaN, or missing value.		
Format	<code>y = isinfnanmiss(x);</code>		
Input	<i>x</i>	NxK matrix.	
Output	<i>y</i>	scalar, 1 if <i>x</i> contains any infinities, NaNs, or missing values, else 0.	
See Also	<code>scalinfnanmiss, ismiss, scalmiss</code>		

issmiss

Purpose	Returns a 1 if its matrix argument contains any missing values, otherwise returns a 0.
Format	<code>y = issmiss(x);</code>
Input	<i>x</i> NxK matrix.
Output	<i>y</i> scalar, 1 or 0.
Remarks	<p><i>y</i> will be a scalar 1 if the matrix <i>x</i> contains any missing values, otherwise it will be a 0.</p> <p>An element of <i>x</i> is considered to be a missing if and only if it contains a missing value in the real part. Thus, for $x = 1 + .i$, issmiss(<i>x</i>) will return a 0.</p>
Example	<pre>x = { 1 6 3 4 }; y = issmiss(x); y = 0</pre>
See also	scalmiss, miss, missrv

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

isSparse

isSparse

Purpose	Tests whether a matrix is a sparse matrix.		
Format	$r = \text{isSparse}(x);$		
Input	x	MxN sparse or dense matrix.	
Output	r	scalar, 1 if x is sparse, 0 otherwise.	
Source	sparse.src		

keep (dataloop)

Purpose	Specifies columns (variables) to be saved to the output data set in a data loop.
Format	keep <i>variable_list</i> ;
Remarks	<p>Commas are optional in variable_list.</p> <p>Retains only the specified variables in the output data set. Any variables referenced must already exist, either as elements of the source data set, or as the result of a previous make, vector, or code statement.</p> <p>If neither keep nor drop is used, the output data set will contain all variables from the source data set, as well as any newly defined variables. The effects of multiple keep and drop statements are cumulative.</p>
Example	<code>keep age, pay, sex;</code>
See also	drop

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

key

key

a	Purpose	Returns the ASCII value of the next key available in the keyboard buffer.
b	Format	y = key ;
c	Remarks	If you are working in terminal mode, key does not “see” any keystrokes until ENTER is pressed. The value returned will be zero if no key is available in the buffer or it will equal the ASCII value of the key if one is available. The key is taken from the buffer at this time and the next call to key will return the next key.
d		Here are the values returned if the key pressed is not a standard ASCII character in the range of 1-255.
e		1015 SHIFT+TAB
f		1016-1025 ALT+Q, W, E, R, T, Y, U, I, O, P
g		1030-1038 ALT+A, S, D, F, G, H, J, K, L
h		1044-1050 ALT+Z, X, C, V, B, N, M
i		1059-1068 F1-F10
j		1071 HOME
k		1072 CURSOR UP
l		1073 PAGE UP
m		1075 LEFT ARROW
n		1077 RIGHT ARROW
o		1079 END
p		1080 DOWN ARROW
q		1081 PAGE DOWN
r		1082 INSERT
s		1083 DELETE
t		1084-1093 SHIFT+F1-F10
u		1094-1103 CTRL+F1-F10
v		1104-1113 ALT+F1-F10
w		1114 CTRL+PRINT SCREEN
x		1115 CTRL+LEFT ARROW
y		1116 CTRL+RIGHT ARROW
z		1117 CTRL+END
		1118 CTRL+PAGE DOWN

1119	CTRL+HOME
1120-1131	ALT+1,2,3,4,5,6,7,8,9,0,HYPHEN, EQUAL SIGN
1132	CTRL+PAGE UP

Example

```
format /rds 1,0;
kk = 0;
do until kk == 27;
    kk = key;
    if kk == 0;
        continue;
    elseif kk == vals(" ");
        print "space \\" kk;
    elseif kk == vals("\r");
        print "carriage return \\" kk;
    elseif kk ≥ vals("0") and kk ≤ vals("9");
        print "digit \\" kk chrs(kk);
    elseif vals(upper(chrs(kk))) ≥ vals("A") and
        vals(upper(chrs(kk))) ≤ vals("Z");
        print "alpha \\" kk chrs(kk);
    else;
        print "\\\" kk;
    endif;
endo;
```

This is an example of a loop that processes keyboard input. This loop will continue until the ESC key (ASCII 27) is pressed.

See also **vals, chrs, upper, lower, con, cons**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

keyw

keyw

a	Purpose	Waits for and gets a key.
b	Format	$k = \text{keyw};$
c	Output	k scalar, ASCII value of the key pressed.
d	Remarks	If you are working in terminal mode, GAUSS will not see any input until you press the ENTER key. keyw gets the next key from the keyboard buffer. If the keyboard buffer is empty, keyw waits for a keystroke. For normal keys, keyw returns the ASCII value of the key. See key for a table of return values for extended and function keys.
e	See also	key
f		
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

keyword

Purpose Begins the definition of a keyword procedure. Keywords are user-defined functions with local or global variables.

Format **keyword** *name*(*str*) ;

Input

name literal, name of the keyword. This name will be a global symbol.

str string, a name to be used inside the keyword to refer to the argument that is passed to the keyword when the keyword is called. This will always be local to the keyword, and cannot be accessed from outside the keyword or from other keywords or procedures.

Remarks A keyword definition begins with the **keyword** statement and ends with the **endp** statement. See “Procedures and Keywords” in *User’s Guide*.

Keywords always have 1 string argument and 0 returns. GAUSS will take everything past *name*, excluding leading spaces, and pass it as a string argument to the keyword. Inside the keyword, the argument is a local string. The user is responsible to manipulate or parse the string.

An example of a keyword definition is:

```
keyword add(str);
    local tok,sum;
    sum = 0;
    do until str $== "";
        { tok, str } = token(str);
        sum = sum + stof(tok);
    endo;
    print "Sum is: " sum;
endp;
```

To use this keyword, type:

```
add 1 2 3 4 5;
```

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

keyword

This keyword will respond by printing:

```
Sum is: 15
```

See also **proc, local, endp**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

lag (dataloop)

Purpose Lags variables a specified number of periods.

Format **lag** *nv1* = *var1:p1* [*nv2 = var2:p2...*];

Input *var* name of the variable to lag.
p scalar constant, number of periods to lag.

Output *nv* name of the new lagged variable.

Remarks You can specify any number of variables to lag. Each variable can be lagged a different number of periods. Both positive and negative lags are allowed.

Lagging is executed before any other transformations. If the new variable name is different from that of the variable to lag, the new variable is first created and appended to a temporary data set. This temporary data set becomes the input data set for the data loop, and is then automatically deleted.

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

lag1

lag1

a	Purpose	Lags a matrix by one time period for time series analysis.
b	Format	$y = \text{lag1}(x);$
c	Input	x NxK matrix.
d	Output	y NxK matrix, x lagged 1 period.
e	Remarks	lag1 lags x by one time period, so the first observations of y are missing.
f	Source	lag.src
g	See also	lagn
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

lagn

Purpose	Lags a matrix a specified number of time periods for time series analysis.	
Format	$y = \mathbf{lagn}(x, t);$	
Input	x	NxK matrix.
	t	scalar, number of time periods.
Output	y	NxK matrix, x lagged t periods.
Remarks	If t is positive, lagn lags x back t time periods, so the first t observations of y are missing. If t is negative, lagn lags x forward t time periods, so the last t observations of y are missing.	
Source	lag.src	
See also	lag1	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

let

let

a		
b		
c		
d		
e		
f		
g		
h		
i		
j		
k		
l		
m	Purpose	Creates a matrix from a list of numeric or character values. The result is always of type matrix , string or string array .
n	Format	let <i>x</i> = <i>constant_list</i> ;
o	Remarks	Expressions and matrix names are not allowed in the let command. Expressions such as this: let x[2,1] = 3*a b are illegal. To define matrices by combining matrices and expressions, use an expression containing the concatenation operators: ~ and . Numbers can be entered in scientific notation. The syntax is <i>dE±n</i> , where <i>d</i> is a number and <i>n</i> is an integer (denoting the power of 10). let x = 1e+10 1.1e-4 4.019e+2; Complex numbers can be entered by joining the real and imaginary parts with a sign (+ or -); there should be no spaces between the numbers and the sign. Numbers with no real part can be entered by appending an “i” to the number. let x = 1.2+23 8.56i 3-2.1i -4.2e+6i 1.2e-4-4.5e+3i; If curly braces are used, the let is optional. You will need the let for statements that you want to protect from the beautifier using the -l flag on the beautifier command line. let x = { 1 2 3, 4 5 6, 7 8 9 }; x = { 1 2 3, 4 5 6, 7 8 9 }; If indices are given, a matrix of that size will be created: let x[2,2] = 1 2 3 4; x = 1 2 3 4
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

If indices are not given, a column vector will be created:

```
let x = 1 2 3 4;
```

```

      1
x =   2
      3
      4

```

You can create matrices with no elements, i.e., “empty matrices”. Just use a set of empty curly braces.

```
x = { };
```

Empty matrices are chiefly used as the starting point for building up a matrix, for example in a **do** loop. For more information on empty matrices, see “Language Fundamentals” in the *User’s Guide*.

Character elements are allowed in a **let** statement:

```
let x = age pay sex;
```

```

      AGE
x =   PAY
      SEX

```

Lowercase elements can be created if quotation marks are used. Note that each element must be quoted.

```
let x = "age" "pay" "sex";
```

```

      age
x =   pay
      sex

```

Example `let x;`

```
x = 0
```

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

let

```
let x = { 1 2 3, 4 5 6, 7 8 9 };
```

```
      1 2 3  
x =   4 5 6  
      7 8 9
```

```
let x[3,3] = 1 2 3 4 5 6 7 8 9;
```

```
      1 2 3  
x =   4 5 6  
      7 8 9
```

```
let x[3,3] = 1;
```

```
      1 1 1  
x =   1 1 1  
      1 1 1
```

```
let x[3,3];
```

```
      0 0 0  
x =   0 0 0  
      0 0 0
```

```
let x = 1 2 3 4 5 6 7 8 9;
```

```
1
```

```
2
```

```
3
```

```
4
```

```
x = 5
```

```
6
```

```
7
```

```
8
```

```
9
```

```
let x = dog cat;
```

```
x = DOG  
    CAT
```

```
let x = "dog" "cat";
```

```
x = dog  
    cat
```

```
let string x = { "Median Income"  
                 "Country";
```

```
x = Median Income  
    Country
```

See also **con**, **cons**, **declare**, **load**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

lib

lib

a		
b	Purpose	Builds and updates library files.
c	Format	lib <i>library</i> <i>[[file]]</i> <i>[[-flag -flag...]]</i> ;
d	Input	<i>library</i> literal, name of library.
e		<i>file</i> optional literal, name of source file to be updated or added.
f		<i>flags</i> optional literal preceded by ‘-’, controls operation of library update. To control handling of path information on source filenames:
g		-addpath (default) add paths to entries without paths and expand relative paths.
h		-gausspath reset all paths using a normal file search.
i		-leavepath leave all path information untouched.
j		-nopath drop all path information.
k		To specify a library update or a complete library build:
l		-update (default) update the symbol information for the specified file only.
m		-build update the symbol information for every library entry by compiling the actual source file.
n		-delete delete a file from the library.
o		-list list files in a library.
p		To control the symbol type information placed in the library file:
q		-strong (default) use strongly typed symbol entries.
r		-weak save no type information. This should only be used to build a library compatible with a previous version of GAUSS.
s		
t		
u		
v		
w		
x y z		

To control location of temporary files for a complete library build:

- tmp** (default) use the directory pointed to by the **tmp_path** configuration variable. The directory will usually be on a RAM disk. If **tmp_path** is not defined, **lib** will look for a **tmp** environment variable.
- disk** use the same directory listed in the **lib_path** configuration variable.

Remarks The flags can be shortened to one or two letters, as long as they remain unique — for example, **-b** to **-build** a library, **-li** to list files in a library.

If the filenames include a full path, the compilation process is faster because no unnecessary directory searching is needed during the autoloading process. The default path handling adds a path to each file listed in the library and also expands any relative paths so the system will work from any drive or subdirectory.

When a path is added to a filename containing no path information, the file is searched for on the current directory and then on each subdirectory listed in **src_path**. The first path encountered that contains the file is added to the filename in the library entry.

See also **library**

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

library

library

a							
b							
c							
d							
e							
f							
g							
h							
i							
j							
k							
l							
m							
n							
o							
p							
q							
r							
s							
t							
u							
v							
w							
x y z							
	<table><tr><td>Purpose</td><td>Sets up the list of active libraries.</td></tr><tr><td>Format</td><td>library [-l] <i>lib1,lib2,lib3,lib4;</i> library;</td></tr><tr><td>Remarks</td><td><p>If no arguments are given, the list of current libraries will be printed out.</p><p>The -l option will write a file containing a listing of libraries, files, and symbols for all active libraries. This file will reside in the directory defined by the lib_path configuration variable. Under Windows and UNIX, the file will have a unique name beginning with liblst_. Under OS/2 and DOS, the file will be called gausslib.lst; if it already exists it will be overwritten.</p><p>For more information about the library system, see “Libraries” in the <i>User’s Guide</i>.</p><p>The default extension for library files is .lcg.</p><p>If a list of library names is given, they will be the new set of active libraries. The two default libraries are user.lcg and gauss.lcg. Unless otherwise specified, user.lcg will be searched first and gauss.lcg will be searched last. Any other user-specified libraries will be searched after user.lcg in the order they were entered in the library statement.</p><p>If the statement:</p><div><pre>y = dog(x);</pre></div><p>is encountered in a program, dog will be searched for in the active libraries. If it is found, it will be compiled. If it cannot be found in a library, the deletion state determines how it is handled:</p><div><pre>autodelete on search for dog.g autodelete off return Undefined symbol error message</pre></div><p>If dog calls cat and cat calls bird and they are all in separate files, they will all be found by the autoloader.</p><p>The source browser and the help facility will search for dog in exactly the same sequence as the autoloader. The file containing dog will be displayed in the window and you can scroll up and down and look at the code and comments.</p></td></tr></table>	Purpose	Sets up the list of active libraries.	Format	library [-l] <i>lib1,lib2,lib3,lib4;</i> library;	Remarks	<p>If no arguments are given, the list of current libraries will be printed out.</p> <p>The -l option will write a file containing a listing of libraries, files, and symbols for all active libraries. This file will reside in the directory defined by the lib_path configuration variable. Under Windows and UNIX, the file will have a unique name beginning with liblst_. Under OS/2 and DOS, the file will be called gausslib.lst; if it already exists it will be overwritten.</p> <p>For more information about the library system, see “Libraries” in the <i>User’s Guide</i>.</p> <p>The default extension for library files is .lcg.</p> <p>If a list of library names is given, they will be the new set of active libraries. The two default libraries are user.lcg and gauss.lcg. Unless otherwise specified, user.lcg will be searched first and gauss.lcg will be searched last. Any other user-specified libraries will be searched after user.lcg in the order they were entered in the library statement.</p> <p>If the statement:</p> <div><pre>y = dog(x);</pre></div> <p>is encountered in a program, dog will be searched for in the active libraries. If it is found, it will be compiled. If it cannot be found in a library, the deletion state determines how it is handled:</p> <div><pre>autodelete on search for dog.g autodelete off return Undefined symbol error message</pre></div> <p>If dog calls cat and cat calls bird and they are all in separate files, they will all be found by the autoloader.</p> <p>The source browser and the help facility will search for dog in exactly the same sequence as the autoloader. The file containing dog will be displayed in the window and you can scroll up and down and look at the code and comments.</p>
Purpose	Sets up the list of active libraries.						
Format	library [-l] <i>lib1,lib2,lib3,lib4;</i> library;						
Remarks	<p>If no arguments are given, the list of current libraries will be printed out.</p> <p>The -l option will write a file containing a listing of libraries, files, and symbols for all active libraries. This file will reside in the directory defined by the lib_path configuration variable. Under Windows and UNIX, the file will have a unique name beginning with liblst_. Under OS/2 and DOS, the file will be called gausslib.lst; if it already exists it will be overwritten.</p> <p>For more information about the library system, see “Libraries” in the <i>User’s Guide</i>.</p> <p>The default extension for library files is .lcg.</p> <p>If a list of library names is given, they will be the new set of active libraries. The two default libraries are user.lcg and gauss.lcg. Unless otherwise specified, user.lcg will be searched first and gauss.lcg will be searched last. Any other user-specified libraries will be searched after user.lcg in the order they were entered in the library statement.</p> <p>If the statement:</p> <div><pre>y = dog(x);</pre></div> <p>is encountered in a program, dog will be searched for in the active libraries. If it is found, it will be compiled. If it cannot be found in a library, the deletion state determines how it is handled:</p> <div><pre>autodelete on search for dog.g autodelete off return Undefined symbol error message</pre></div> <p>If dog calls cat and cat calls bird and they are all in separate files, they will all be found by the autoloader.</p> <p>The source browser and the help facility will search for dog in exactly the same sequence as the autoloader. The file containing dog will be displayed in the window and you can scroll up and down and look at the code and comments.</p>						

library

Library files are simple ASCII files that you can create with the editor. Here is an example:

```
/*
** This is a GAUSS library file.
*/

eig.src
    eig      : proc
    eigsym   : proc
    _eigerr  : matrix
svd.src
    cond     : proc
    pinv     : proc
    rank     : proc
    svd      : proc
    _svdtol  : matrix
```

The lines not indented are the file names. The lines that are indented are the symbols defined in that file. As you can see, a GAUSS library is a dictionary of files and the global symbols they contain.

Any line beginning with `/*`, `**` or `*/` is considered a comment. Blank lines are okay.

Here is a debugging hint. If your program is acting strange and you suspect it is autoloading the wrong copy of a procedure, use the source browser or help facility to locate the suspected function. It will use the same search path that the autoloader uses.

See also **`declare`**, **`external`**, **`lib`**, **`proc`**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

#lineson, #linesoff

#lineson, #linesoff

a		
b		
c		
d		
e		
f		
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		
	Purpose	The #lineson command causes GAUSS to embed line number and file name records in a program for the purpose of reporting the location where an error occurs. The #linesoff command causes GAUSS to stop embedding line and file records in a program.
	Format	#lineson; #linesoff;
	Remarks	<p>In the “lines on” mode, GAUSS keeps track of line numbers and file names and reports the location of an error when an execution time error occurs. In the “lines off” mode, GAUSS does not keep track of lines and files at execution time. During the compile phase, line numbers and file names will always be given when errors occur in a program stored in a disk file.</p> <p>It is easier to debug a program when the locations of errors are reported, but this slows down execution. In programs with several scalar operations, the time spent tracking line numbers and file names is most significant.</p> <p>These commands have no effect on interactive programs (that is, those typed in the window and run from the command line), since there are no line numbers in such programs.</p> <p>Line number tracking can be turned on and off through the user interface, but the #lineson and #linesoff commands will override that.</p> <p>The line numbers and file names given at run-time will reflect the last record encountered in the code. If you have a mixture of procedures that were compiled without line and file records and procedures that were compiled with line and file records, use the trace command to locate exactly where the error occurs.</p> <p>The Currently active call error message will always be correct. If it states that it was executing procedure xyz at line number nnn in file ABC and xyz has no line nnn or is not in file ABC, you know that it just did not encounter any line or file records in xyz before it crashed.</p> <p>When using #include’d files, the line number and file name will be correct for the file the error was in, within the limits stated above.</p>
	See also	trace

listwise (dataloop)

Purpose Controls listwise deletion of missing values.

Format **listwise** `[[read]]` `[[write]]`;

Remarks If **read** is specified, the deletion of all rows containing missing values happens immediately after reading the input file and before any transformations. If **write** is specified, the deletion of missing values happens after any transformations and just before writing to the output file. If no **listwise** statement is present, rows with missing values are not deleted.

 The default is **read**.

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

ln

ln

a	Purpose	Computes the natural log of all elements of a matrix.
b	Format	$y = \ln(x);$
c	Input	x NxK matrix.
d	Output	y NxK matrix containing the natural log values of the elements of x .
e	Remarks	\ln is defined for $x \neq 0$. If x is negative, complex results are returned. You can turn the generation of complex numbers for negative inputs on or off in the GAUSS configuration file, and with the sysstate function, case 8. If you turn it off, \ln will generate an error for negative inputs. If x is already complex, the complex number state doesn't matter; \ln will compute a complex result. x can be any expression that returns a matrix.
f	Example	$y = \ln(16);$ $y = 2.7725887$
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

lncdfbvn

Purpose	Computes natural log of bivariate Normal cumulative distribution function.		
Format	$y = \text{lncdfbvn}(x1, x2, r);$		
Input	$x1$	NxK matrix, abscissae.	
	$x2$	LxM matrix, abscissae.	
	r	PxQ matrix, correlations.	
Output	y	max(N,L,P) x max(K,M,Q) matrix, $\ln Pr(X < x1, X < x2 \mid r)$.	
Remarks	$x1$, $x2$, and r must be ExE conformable.		
Source	lncdfn.src		
See also	cdfbvn , lncdfmvn		

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

lncdfbvn2

lncdfbvn2

Purpose	Returns log of cdfbvn of a bounded rectangle.										
Format	$y = \text{lncdfbvn2}(h, dh, k, dk, r);$										
Input	<table><tr><td>h</td><td>Nx1 vector, upper limits of integration for variable 1.</td></tr><tr><td>dh</td><td>Nx1 vector, increments for variable 1.</td></tr><tr><td>k</td><td>Nx1 vector, upper limits of integration for variable 2.</td></tr><tr><td>dk</td><td>Nx1 vector, increments for variable 2.</td></tr><tr><td>r</td><td>Nx1 vector, correlation coefficients between the two variables.</td></tr></table>	h	Nx1 vector, upper limits of integration for variable 1.	dh	Nx1 vector, increments for variable 1.	k	Nx1 vector, upper limits of integration for variable 2.	dk	Nx1 vector, increments for variable 2.	r	Nx1 vector, correlation coefficients between the two variables.
h	Nx1 vector, upper limits of integration for variable 1.										
dh	Nx1 vector, increments for variable 1.										
k	Nx1 vector, upper limits of integration for variable 2.										
dk	Nx1 vector, increments for variable 2.										
r	Nx1 vector, correlation coefficients between the two variables.										
Output	<table><tr><td>y</td><td>Nx1 vector, the log of the integral from h, k to $h+dh, k+dk$ of the standardized bivariate Normal distribution.</td></tr></table>	y	Nx1 vector, the log of the integral from h, k to $h+dh, k+dk$ of the standardized bivariate Normal distribution.								
y	Nx1 vector, the log of the integral from h, k to $h+dh, k+dk$ of the standardized bivariate Normal distribution.										
Remarks	<p>Scalar input arguments are okay; they will be expanded to Nx1 vectors.</p> <p>lncdfbvn2 will abort if the computed integral is negative.</p> <p>lncdfbvn2 computes an error estimate for each set of inputs--the real integral is exp(y)±err. The size of the error depends on the input arguments. If trap 2 is set, a warning message is displayed when $\text{err} \geq \text{exp}(y)/100$.</p> <p>For an estimate of the actual error, see cdfbvn2e.</p>										
Example	<p>Example 1</p> <pre>lncdfbvn2(1,1,1,1,0.5);</pre> <p>produces:</p> <pre>-3.2180110258198771e+000</pre> <p>Example 2</p> <pre>trap 0,2;</pre> <pre>lncdfbvn2(1,1e-15,1,1e-15,0.5);</pre> <p>produces:</p> <pre>-7.1171016046360151e+001</pre>										

lncdfbvn2

Example 3

```
trap 2,2;
lncdfbvn2(1,-1e-45,1,1e-45,0.5);
WARNING: Dubious accuracy from lncdfbvn2:
0.000e+000 ± 2.8e-060
-INF
```

See also **cdfbvn2**, **cdfbvn2e**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

lncdfmvn

lncdfmvn

Purpose	Computes natural log of multivariate Normal cumulative distribution function.		
Format	$y = \text{lncdfmvn}(x, r);$		
Input	x	KxL matrix, abscissae.	
	r	KxK matrix, correlation matrix.	
Output	y	Lx1 vector, $\ln Pr(X < x \mid r)$.	
Remarks	You can pass more than one set of abscissae at a time; each column of x is treated separately.		
Source	lncdfn.src		
See also	cdfmvn , lncdfbvn		

lncdfn

Purpose Computes natural log of Normal cumulative distribution function.

Format $y = \text{lncdfn}(x);$

Input x NxK matrix, abscissae.

Output y NxK matrix, $\ln Pr(X < x)$.

Source `lncdfn.src`

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

lncdfn2

lncdfn2

a	<div>Purpose</div> <div>Format</div> <div>Input</div> <div>Output</div>	<div>Computes natural log of interval of Normal cumulative distribution function.</div> <div>$y = \text{lncdfn2}(x,r);$</div> <div>x MxN matrix, abscissae. r KxL matrix, ExE conformable with x, intervals.</div> <div>y max(M,K) x max(N,L) matrix, the log of the integral from x to $x+dx$ of the Normal distribution, i.e., $\ln Pr(x < X < x + dx)$.</div>
b		
c		
d		
e	<div>Remarks</div> <div>Example</div>	<div>The relative error is:</div> <div>$x \leq 1$ and $dx \leq 1$ $\pm 1e-14$ $1 < x < 37$ and $dx < 1 x$ $\pm 1e-13$ $\min(x,x + dx) > -37$ and $y > -690$ $\pm 1e-11$ or better</div> <div>A relative error of $\pm 1e-14$ implies that the answer is accurate to better than ± 1 in the 14th digit.</div> <div><pre>print lncdfn2(-10,29); -7.6198530241605269e-24 print lncdfn2(0,1); -1.0748623268620716e+00 print lncdfn2(5,1); -1.5068446096529453e+01</pre></div>
f		
g		
h		
i	<div>Source</div> <div>See also</div>	<div>lncdfn.src</div> <div>cdfn2</div>
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

lncdfnc

Purpose	Computes natural log of complement of Normal cumulative distribution function.		
Format	$y = \text{lncdfnc}(x);$		
Input	x	NxK matrix, abscissae.	
Output	y	NxK matrix, $\ln (1 - Pr(X < x))$.	
Source	lncdfn.src		

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

lnfact

lnfact

a		
b		
c		
d		
e		
f		
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		
	Purpose	Computes the natural log of the factorial function and can be used to compute log gamma.
	Format	<code>y = lnfact(x);</code>
	Input	<code>x</code> NxK matrix, all elements must be positive.
	Output	<code>y</code> NxK matrix containing the natural log of the factorial of each of the elements of <code>x</code> .
	Remarks	<p>For integer x, this is (approximately) $\ln(x!)$. However, the computation is done using a formula, and the function is defined for noninteger x.</p> <p>In most formulae in which the factorial operator appears, it is possible to avoid computing the factorial directly, and to use lnfact instead. The advantage of this is that lnfact does not have the overflow problems that the factorial (!) operator has.</p> <p>For $x \geq 1$, this function has at least 6 digit accuracy, for $x > 4$ it has at least 9 digit accuracy, and for $x > 10$ it has at least 12 digit accuracy. For $0 < x < 1$, accuracy is not known completely but is probably at least 6 digits.</p> <p>Sometimes log gamma is required instead of log factorial. These functions are related by:</p> $\text{lngamma}(x) = \text{lnfact}(x-1);$
	Example	<pre>let x = 100 500 1000; y = lnfact(x); 363.739375560 y = 2611.33045846 5912.12817849</pre>
	Source	<code>lnfact.src</code>
	See also	gamma

lnfact

Technical Notes

For $x > 1$, Stirling’s formula is used.

For $0 < x \leq 1$, `ln(gamma(x+1))` is used.

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

lnpdfn

lnpdfn

a	Purpose	Computes standard Normal log-probabilities.
b	Format	$z = \text{lnpdfn}(x);$
c	Input	x NxK matrix, data.
d	Output	z NxK matrix, log-probabilities.
e	Remarks	This computes the log of the scalar Normal density function for each element of x . z could be computed by the following GAUSS code: $y = -\ln(\text{sqrt}(2*\text{pi})) - x.*x/2;$ For multivariate log-probabilities, see lnpdfmvn .
f	Example	$x = \{ .2, -1, 0, 1, 2 \};$ $z = \text{lnpdfn}(x);$ -2.9189385 -1.4189385 $z =$ -0.91893853 -1.4189385 -2.9189385
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

lnpdfmvn

Purpose	Computes multivariate Normal log-probabilities.		
Format	$z = \text{lnpdfmvn}(x,s);$		
Input	x	NxK matrix, data.	
	s	KxK matrix, covariance matrix.	
Output	z	Nx1 vector, log-probabilities.	
Remarks	This computes the multivariate Normal log-probability for each row of x .		

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

lnpdfmvt

lnpdfmvt

a	Purpose Computes multivariate Student's t log-probabilities.		
b			
c	Format	$z = \text{lnpdfmvt}(x,s,nu);$	
d	Input	x	NxK matrix, data.
e		s	KxK matrix, covariance matrix.
f		nu	scalar, degrees of freedom.
g	Output	z	Nx1 vector, log-probabilities.
h			
i			
j			
k			
l			
m			
n			
o			
p			
q			
r			
s			
t			
u			
v			
w			
x y z			

lnpdf^t

Purpose	Computes Student's t log-probabilities.	
Format	$z = \text{lnpdf}^t(x, nu);$	
Input	x	NxK matrix, data.
	nu	scalar, degrees of freedom.
Output	z	NxK matrix, log-probabilities.
Remarks	<p>This does not compute the log of the joint Student's t pdf. Instead, the scalar Normal density function is computed element by element.</p> <p>For multivariate probabilities with covariance matrix see lnpdfmvt.</p>	

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

load, loadf, loadk, loadm, loadp, loads

load, loadf, loadk, loadm, loadp, loads

a	Purpose	Loads from a disk file.
b	Format	load [[path= <i>path</i>]] <i>x</i> , <i>y</i> []= <i>filename</i> , <i>z</i> = <i>filename</i> ;
c	Remarks	All the loadxx commands use the same syntax — they only differ in the types of symbols you use them with.
d		
e		load, loadm matrix
f		loads string
g		loadf function (fn)
h		loadk keyword (keyword)
i		loadp procedure (proc)
j		If no filename is given as with <i>x</i> above, then the symbol name the file is to be loaded into is used as the filename and the proper extension is added.
k		If more than one item is to be loaded in a single statement, the names should be separated by commas.
l		The filename can be either a literal or a string. If the filename is in a string variable, then the ^ (caret) operator must precede the name of the string, as in:
m		<pre>filestr = "mydata/char";</pre>
n		<pre>loadm x = ^filestr;</pre>
o		
p		If no extension is supplied, the proper extension for each type of file will be used automatically as follows:
q		load .fmt - matrix file or delimited ASCII file
r		loadm .fmt - matrix file or delimited ASCII file
s		loads .fst - string file
t		loadf .fcg - user-defined function (fn) file
u		loadk .fcg - user-defined keyword (keyword) file
v		loadp .fcg - user-defined procedure (proc) file
w		These commands also signal to the compiler what type of object the symbol is so that later references to it will be compiled correctly.
x y z		A dummy definition must exist in the program for each symbol that is loaded in using loadf , loadk , or loadp . This resolves the need to

load, loadf, loadk, loadm, loadp, loads

have the symbol initialized at compile time. When the load executes, the dummy definition will be replaced with the saved definition.

```
proc corrmatrix; endp;

loadp corrmatrix;

y = corrmatrix;

keyword regress(x); endp;

loadk regress;

regress x on y z t from data01;

fn sqrd=;

loadf sqrd;

y = sqrd(4.5);
```

To load GAUSS files created with the **save** command, no brackets are used with the symbol name.

If you use **save** to save a scalar error code 65535 (i.e., **error(65535)**), it will be interpreted as an empty matrix when you **load** it again.

ASCII data files

To load ASCII data files, square brackets follow the name of the symbol.

Numbers in ASCII files must be delimited with spaces, commas, tabs, or newlines. If the size of the matrix to be loaded is not explicitly given, as in:

```
load x[] = data.asc;
```

GAUSS will load as many elements as possible from the file and create an Nx1 matrix. This is the preferred method of loading ASCII data from a file, especially when you want to verify if the load was successful. Your program can then see how many elements were actually loaded by testing the matrix with the **rows** command, and if that is correct, the Nx1 matrix can be reshaped to the desired form. You could, for instance, put the number of rows and columns of the matrix right in the file as the first and second elements and reshape the remainder of the vector to the desired form using those values.

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

load, loadf, loadk, loadm, loadp, loads

If the size of the matrix is explicitly given in the **load** command, then no checking will be done. If you use:

```
load x[500,6] = data.asc;
```

GAUSS will still load as many elements as possible from the file into an $N \times 1$ matrix and then automatically reshape it using the dimensions given.

If your file contains nine numbers (1 2 3 4 5 6 7 8 9), then the matrix x that was created would be as follows:

```
load x[1,9] = data.asc;
```

```
x = 1 2 3 4 5 6 7 8 9
```

```
load x[3,3] = data.asc;
```

```
      1 2 3
x =   4 5 6
      7 8 9
```

```
load x[2,2] = data.asc;
```

```
x =  1 2
     3 4
```

```
load x[2,9] = data.asc;
```

```
x =  1 2 3 4 5 6 7 8 9
     1 2 3 4 5 6 7 8 9
```

```
load x[3,5] = data.asc;
```

```
      1 2 3 4 5
x =   6 7 8 9 1
      2 3 4 5 6
```

load accepts pathnames. The following is legal:

```
loadm k = /gauss/x;
```

This will load /gauss/x.fmt into **k**.

load, loadf, loadk, loadm, loadp, loads

If the **path=** subcommand is used with **load** and **save**, the path string will be remembered until changed in a subsequent command. This path will be used whenever none is specified. There are four separate paths for:

1. **load, loadm**
2. **loadf, loadp**
3. **loads**
4. **save**

Setting any of the four paths will not affect the others. The current path settings can be obtained (and changed) with the **sysstate** function, cases 4-7.

```
loadm path = /data;
```

This will change the **loadm** path without loading anything.

```
load path = /gauss x,y,z;
```

This will load **x.fmt**, **0y.fmt**, and **z.fmt** using **/gauss** as a path. This path will be used for the next load if none is specified.

The load path or save path can be overridden in any particular load or save by putting an explicit path on the filename given to load from or save to as follows:

```
loadm path = /miscdata;
loadm x = /data/mydata1, y, z = hisdata;
```

In the above program:

/data/mydata1.fmt would be loaded into a matrix called **x**.

/miscdata/y.fmt would be loaded into a matrix called **y**.

/miscdata/hisdata.fmt would be loaded into a matrix called **z**.

```
oldmpath = sysstate(5,"/data");
load x, y;
call sysstate(5,oldmpath);
```

This will get the old **loadm** path, set it to **/data**, load **x.fmt** and **y.fmt**, and reset the **loadm** path to its original setting.

See also **loadd, save, let, con, cons, sysstate**

loadadd

loadadd

a	Purpose	Loads a data set.
b	Format	$y = \text{loadadd}(\text{dataset});$
c	Input	dataset string, name of data set.
d	Output	y NxK matrix of data.
e	Remarks	The data set must not be larger than a single GAUSS matrix. If dataset is a null string or 0, the data set <code>temp.dat</code> will be loaded. To load a matrix file, use an <code>.fmt</code> extension on dataset .
f	Source	<code>saveload.src</code>
g	Globals	<code>__maxvec</code>
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

loadwind

Purpose	Loads a previously saved graphic panel configuration.	
Library	pgraph	
Format	<i>err</i> = loadwind (<i>namestr</i>);	
Input	<i>namestr</i>	string, name of file to be loaded.
Output	<i>err</i>	scalar, 0 if successful, 1 if graphic panel matrix is invalid. Note that the current graphic panel configuration will be overwritten in either case.
Source	pwindow.src	
See also	savewind	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

local

local

a		
b	Purpose	Declares variables that are to exist only inside a procedure.
c	Format	local <i>x, y, f:proc;</i>
d	Remarks	The statement above would place the names <i>x</i> , <i>y</i> , and <i>f</i> in the local symbol table for the current procedure being compiled. This statement is legal only between the proc statement and the endp statement of a procedure definition.
e		These symbols cannot be accessed outside of the procedure.
f		The symbol <i>f</i> in the example above will be treated as a procedure whenever it is accessed in the procedure. What is actually passed in is a pointer to a procedure.
g		See “Procedures and Keywords” in the <i>User’s Guide</i> .
h	See also	proc
i		
j		
k		

locate

Purpose	Positions the cursor in the window.
Format	locate <i>m</i> , <i>n</i> ;
Portability	Windows only Locates the cursor in the current output window.
Remarks	<i>m</i> and <i>n</i> denote the row and column, respectively, at which the cursor is to be located. The origin (1,1) is the upper left corner. <i>m</i> and <i>n</i> may be any expressions that return scalars. Nonintegers will be truncated to an integer.
Example	<pre>r = csrln; c = csrcl; cls; locate r,c;</pre> <p>In this example the window is cleared without affecting the cursor position.</p>
See also	csrln , csrcl

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

loess

loess

a		
b	Purpose	Computes coefficients of locally weighted regression.
c	Format	$\{ \hat{y}, y_s, x_s \} = \text{loess}(\text{depvar}, \text{indvars}) ;$
d	Input	<i>depvar</i> Nx1 vector, dependent variable. <i>indvars</i> NxK matrix, independent variables.
e		
f	Global Input	_loess_Span scalar, degree of smoothing. Must be greater than 2 / N. Default = .67777.
g		_loess_NumEval scalar, number of points in <i>ys</i> and <i>xs</i> . Default = 50.
h		_loess_Degree scalar, if 2, quadratic fit, otherwise linear. Default = 1.
i		_loess_WgtType scalar, type of weights. If 1, robust, symmetric weights, otherwise Gaussian. Default = 1.
j		__output scalar, if 1, iteration information and results are printed, otherwise nothing is printed.
k		
l	Output	<i>yhat</i> Nx1 vector, predicted <i>depvar</i> given <i>indvars</i> . <i>ys</i> _loess_numEval x1 vector, ordinate values given abscissae values in <i>xs</i> . <i>xs</i> _loess_numEval x1 vector, equally spaced abscissae values.
m		
n		
o		
p	Remarks	Based on Cleveland, William S. "Robust Locally Weighted Regression and Smoothing Scatterplots." <i>JASA</i> . Vol. 74, 1979, 829-36.
q	Source	loess.src
r		
s		
t		
u		
v		
w		
x y z		

log

Purpose	Computes the log ₁₀ of all elements of a matrix.	
Format	$y = \text{log}(x);$	
Input	x	NxK matrix.
Output	y	NxK matrix containing the log 10 values of the elements of x .
Remarks	<p>log is defined for $x \neq 0$.</p> <p>If x is negative, complex results are returned.</p> <p>You can turn the generation of complex numbers for negative inputs on or off in the GAUSS configuration file, and with the sysstate function, case 8. If you turn it off, log will generate an error for negative inputs.</p> <p>If x is already complex, the complex number state doesn't matter; log will compute a complex result.</p> <p>x can be any expression that returns a matrix.</p>	

```
Example  x = round(rndu(3,3)*10+1);
         y = log(x);

           4.0000000000  2.0000000000  1.0000000000
x =      10.0000000000  4.0000000000  8.0000000000
           7.0000000000  2.0000000000  6.0000000000

           0.60205999    0.30103    0.30103
y =      1.0000000000  0.60205999  0.90308999
           0.8450980400    0.30103    0.77815125
```

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

loglog

loglog

a	Purpose	Graphs X vs. Y using log coordinates.
b	Library	pgraph
c	Format	loglog(<i>x</i>,<i>y</i>) ;
d	Input	<i>x</i> Nx1 or NxM matrix. Each column contains the X values for a particular line.
e		<i>y</i> Nx1 or NxM matrix. Each column contains the Y values for a particular line.
f	Source	ploglog.src
g	See also	xy, logy, logx
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

logx

Purpose	Graphs X vs. Y using log coordinates for the X axis.	
Library	pgraph	
Format	logx(x,y);	
Input	<i>x</i>	Nx1 or NxM matrix. Each column contains the X values for a particular line.
	<i>y</i>	Nx1 or NxM matrix. Each column contains the Y values for a particular line.
Source	plogx.src	
See also	xy, logy, loglog	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

logy

logy

Purpose	Graphs X vs. Y using log coordinates for the Y axis.		
Library	pgraph		
Format	logy(<i>x</i>,<i>y</i>) ;		
Input	<i>x</i>	Nx1 or NxM matrix. Each column represents the X values for a particular line.	
	<i>y</i>	Nx1 or NxM matrix. Each column represents the Y values for a particular line.	
Source	plogy.src		
See also	xy, logx, loglog		

lower

Purpose	Converts a string or character matrix to lowercase.	
Format	<code>y = lower(x);</code>	
Input	<i>x</i>	string or NxK matrix of character data to be converted to lowercase.
Output	<i>y</i>	string or NxK matrix which contains the lowercase equivalent of the data in <i>x</i> .
Remarks	If <i>x</i> is a numeric matrix, <i>y</i> will contain garbage. No error message will be generated since GAUSS does not distinguish between numeric and character data in matrices.	
Example	<pre>x = "MATH 401"; y = lower(x); print y;</pre> <p>produces:</p> <p>math 401</p>	
See also	upper	

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

lowmat, lowmat1**lowmat, lowmat1**

Purpose Returns the lower portion of a matrix. **lowmat** returns the main diagonal and every element below. **lowmat1** is the same except it replaces the main diagonal with ones.

Format $L = \text{lowmat}(x);$
 $L = \text{lowmat1}(x);$

Input x NxN matrix.

Output L NxN matrix containing the lower elements of the matrix. The upper elements are replaced with zeros. **lowmat** returns the main diagonal intact. **lowmat1** replaces the main diagonal with ones.

Example $x = \begin{Bmatrix} 1 & 2 & -1, \\ 2 & 3 & -2, \\ 1 & -2 & 1 \end{Bmatrix};$

$L = \text{lowmat}(x);$

$L1 = \text{lowmat1}(x);$

The resulting matrices are:

$L = \begin{Bmatrix} 1 & 0 & 0 \\ 2 & 3 & 0 \\ 1 & -2 & 1 \end{Bmatrix}$

$L1 = \begin{Bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 1 & -2 & 1 \end{Bmatrix}$

Source diag.src

See also **upmat, upmat1, diag, diagrv, crout, croutp**

lpos

Purpose	Returns the current position of the print head within the printer buffer for the printer.
Format	<code>y = lpos;</code>
Remarks	<p>This function is basically equivalent to function csrcol but this returns the current column position for the standard printer.</p> <p>The value returned is the column position of the next character to be printed to the printer buffer. This does not necessarily reflect the actual physical position of the print head at the time of the call.</p> <p>If this function returns a number greater than 1, there are characters in the buffer for the standard printer which have not yet been sent to the printer. This buffer can be flushed at any time by lprint'ing a carriage return/line feed sequence, or a form feed character.</p>
Example	<pre>if lpos > 60; lprint; endif;</pre> <p>In this example, if the print buffer contains 60 characters or more, a carriage return/line feed sequence will be printed.</p>
See also	lprint , lpwidth

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

lprint

lprint

a		
b	Purpose	Controls printing to the line printer.
c	Format	lprint <i>[/typ] [/fnted] [/mf] [/jnt]</i> <i>[list of expressions separated by spaces]</i> <i>[';'];</i>
d	Remarks	This function was originally written for line printers. It is still supported for backwards compatibility purposes, but if you're using a page-oriented printer (such as a laser or inkjet printer), it may not give you the results you're expecting.
e		lprint statements work in essentially the same way that print statements work. The main difference is that lprint statements cannot be directed to the auxiliary output. Also, the locate statement has no meaning with lprint .
f		Two semicolons following an lprint statement will suppress the final line feed.
g		See print for information on <i>/typ</i> , <i>/fnted</i> , <i>/mf</i> , and <i>/jnt</i> .
h		A list of expressions is a list of GAUSS expressions, separated by spaces. In lprint statements, because a space is the delimiter between expressions, no spaces are allowed inside expressions unless they are within index brackets, they are in quotes, or the whole expression is in parentheses.
i		Printer width can be specified by the lpwidth statement:
j		<pre>lpwidth 132;</pre>
k		This statement remains in effect until cancelled. The default printer width is 80. That is, GAUSS automatically sends a line feed to the printer after printing 80 characters.
l		lpos can be used to determine the (column) position of the next character that will be printed in the buffer.
m		An lprint statement by itself will cause a blank line to be printed:
n		<pre>lprint;</pre>
o		The printing of special characters is accomplished by the use of the backslash (\) within double quotes. The options are:
p		<pre>"\b" backspace (ASCII 8)</pre>
q		<pre>"\e" escape (ASCII 27)</pre>
r		
s		
t		
u		
v		
w		
x y z		

lprint

"\f" **form feed** (ASCII 12)
"\l" **line feed** (ASCII 10)
"\r" **carriage return** (ASCII 13)
" t" **tab** (ASCII 9)
"\###" the character whose ASCII value is "**###**" (decimal)

GAUSS also has an automatic line printer mode which causes the results of all global assignment statements to be printed out on the printer. This is controlled by the **lprint on** and **lprint off** commands. (See **lprint on**, **lprint off**.)

Example `lprint 3*4 5+2;`

See also `print, lprint on, lpos, lwidth, format`

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

lpwidth

lpwidth

a		
b		
c		
d		
e		
f		
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

Purpose Specifies the width of the printer.

Format **lpwidth *n*;**

Remarks *n* is a scalar which specifies the width of the printer in columns (characters). That is, after printing *n* characters on a line, GAUSS will send a carriage return and a line feed, so that the print head will move to the beginning of the next line.

If a matrix is being printed, the line feed sequence will always be inserted between separate elements of the matrix rather than being inserted between digits of a single element.

n may be any scalar-valued expression. Nonintegers will be truncated to an integer.

The default is 80 columns.

Note: This does not send control characters to the printer to automatically switch the mode of the printer to a different character pitch because each printer is different. This only controls the frequency of carriage return/line feed sequences.

Example **lpwidth 132;**

This statement will change the printer width to 132 columns.

See also **lprint, lpos, outwidth**

ltrisol

Purpose Computes the solution of $Lx = b$ where L is a lower triangular matrix.

Format $x = \text{ltrisol}(b, L);$

Input b PxK matrix.
 L PxP lower triangular matrix.

Output x PxK matrix.
 ltrisol applies a forward solve to $Lx = b$ to solve for x . If b has more than one column, each column will be solved for separately, i.e., **ltrisol** will apply a forward solve to $L * x[:,i] = b[:,i]$.

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

lu

lu

Purpose	Computes the LU decomposition of a square matrix with partial (row) pivoting, such that $X = LU$.		
Format	{ <i>l,u</i> } = lu(<i>x</i>);		
Input	<i>x</i>	NxN square nonsingular matrix.	
Output	<i>l</i>	NxN “scrambled” lower triangular matrix. This is a lower triangular matrix that has been reordered based on the row pivoting.	
	<i>u</i>	NxN upper triangular matrix.	
Example	<pre>rndseed 13; format /rd 10,4; x = complex(rndn(3,3),rndn(3,3)); { l,u } = lu(x); x2 = l*u;</pre> $x = \begin{bmatrix} 0.1523 + 0.7685i & -0.8957 + 0.0342i & 2.4353 + 2.7736i \\ -1.1953 + 1.2187i & 1.2118 + 0.2571i & -0.0446 - 1.7768i \\ 0.8038 + 1.3668i & 1.2950 - 1.6929i & 1.6267 + 0.2844i \end{bmatrix}$ $l = \begin{bmatrix} 0.2589 - 0.3789i & -1.2417 + 0.5225i & 1.0000 \\ 1.0000 & 0.0000 & 0.0000 \\ 0.2419 - 0.8968i & 1.0000 & 0.0000 \end{bmatrix}$ $u = \begin{bmatrix} -1.1953 + 1.2187i & 1.2118 + 0.2571i & -0.0446 - 1.7768i \\ 0.0000 & 0.7713 - 0.6683i & 3.2309 + 0.6742i \\ 0.0000 & 0.0000 & 6.7795 + 5.7420i \end{bmatrix}$		

$$x2 = \begin{bmatrix} 0.1523 + 0.7685i & -0.8957 + 0.0342i & 2.4353 + 2.7736i \\ -1.1953 + 1.2187i & 1.2118 + 0.2571i & -0.0446 - 1.7768i \\ 0.8038 + 1.3668i & 1.2950 - 1.6929i & 1.6267 + 0.2844i \end{bmatrix}$$

See also **crout**, **croutp**, **chol**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

lusol

lusol

Purpose	Computes the solution of $LUx = b$ where L is a lower triangular matrix and U is an upper triangular matrix.		
Format	$x = \text{lusol}(b, L, U);$		
Input	b	PxK matrix.	
	L	PxP lower triangular matrix.	
	U	PxP upper triangular matrix.	
Output	x	PxK matrix.	
Remarks	If b has more than one column, each column is solved for separately, i.e., lusol solves $LUx[:,i] = b[:,i]$.		

make (dataloop)

Purpose	Specifies the creation of a new variable within a data loop.
Format	make [[#]] <i>numvar</i> = <i>numeric_expression</i> ; make \$ <i>charvar</i> = <i>character_expression</i> ;
Remarks	<p>A <i>numeric_expression</i> is any valid expression returning a numeric vector. A <i>character_expression</i> is any valid expression returning a character vector. If neither '\$' nor '#' is specified, '#' is assumed.</p> <p>The expression may contain explicit variable names and/or GAUSS commands. Any variables referenced must already exist, either as elements of the source data set, as externs, or as the result of a previous make, vector, or code statement. The variable name must be unique. A variable cannot be made more than once, or an error is generated.</p>
Example	<pre>make sqvpt = sqrt(velocity * pressure * temp); make \$ sex = lower(sex);</pre>
See also	vector

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

makevars

makevars

Purpose	Creates separate global vectors from the columns of a matrix.		
Format	makevars (<i>x</i> , <i>vnames</i> , <i>xnames</i>) ;		
Input	<i>x</i>	NxK matrix whose columns will be converted into individual vectors.	
	<i>vnames</i>	string or Mx1 character vector containing names of global vectors to create. If 0, all names in <i>xnames</i> will be used.	
	<i>xnames</i>	string or Kx1 character vector containing names to be associated with the columns of the matrix <i>x</i> .	
Remarks	If <i>xnames</i> = 0, the prefix X will be used to create names. Therefore, if there are 9 columns in <i>x</i> , the names will be X1-X9, if there are 10, they will be X01-X10, and so on.		
	If <i>xnames</i> or <i>vnames</i> is a string, the individual names must be separated by spaces or commas.		
	<pre>vnames = "age pay sex" ;</pre>		
	Since these new vectors are created at execution time, the compiler will not know they exist until after makevars has executed once. This means that you cannot access them by name unless you previously clear them or otherwise add them to the symbol table. (See setvars for a quick interactive solution to this.)		
	This function is the opposite of mergevar .		
Example	<pre>let x[3,3] = 101 35 50000 102 29 13000 103 37 18000;</pre>		
	<pre>let xnames = id age pay;</pre>		
	<pre>let vnames = age pay;</pre>		
	<pre>makevars(x,vnames,xnames) ;</pre>		
	Two global vectors, called age and pay , are created from the columns of x .		

makevars

```
let x[3,3] = 101 35 50000
           102 29 13000
           103 37 18000;
xnames = "id age pay";
vnames = "age pay";
makevars(x,vnames,xnames);
```

This is the same as the example above, except that strings are used for the variable names.

Source `vars.src`

Globals `__vpad`

See also `mergevar`, `setvars`

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

makewind

makewind

a		
b		
c		
d		
e		
f		
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		
	Purpose	Creates a graphic panel of specific size and position and add it to the list of graphic panels.
	Library	pgraph
	Format	makewind (<i>xsize</i> , <i>ysize</i> , <i>xshft</i> , <i>yshft</i> , <i>typ</i>) ;
	Input	<i>xsize</i> scalar, horizontal size of the graphic panel in inches. <i>ysize</i> scalar, vertical size of the graphic panel in inches. <i>xshft</i> scalar, horizontal distance from left edge of window in inches. <i>yshft</i> scalar, vertical distance from bottom edge of window in inches. <i>typ</i> scalar, graphic panel attribute type. If this value is 1, the graphic panels will be transparent. If 0, the graphic panels will be nontransparent.
	Remarks	Note that if this procedure is used when rotating the page, the passed parameters are scaled appropriately to the newly oriented page. The size and shift values will not be true inches when printed, but the graphic panel size to page size ratio remains the same. The result of this implementation automates the rotation and eliminates the required graphic panel recalculations by the user. See the window command for creating tiled graphic panels. For more information on using graphic panels, see “Publication Quality Graphics” in the <i>User’s Guide</i> .
	Source	pwindow.src
	See also	window , endwind , setwind , getwind , begwind , nextwind

margin

Purpose Sets the margins for the current graph graphic panel.

Library **pgraph**

Format **margin(*l,r,t,b*)**;

Input

- l* scalar, the left margin in inches.
- r* scalar, the right margin in inches.
- t* scalar, the top margin in inches.
- b* scalar, the bottom margin in inches.

Remarks By default, the dimensions of the graph are the same as the graphic panel dimensions. With this function the graph dimensions may be decreased. The result will be a smaller plot area surrounded by the specified margin. This procedure takes into consideration the axes labels and numbers for correct placement.

All input inch values for this procedure are based on a full size window of 9 x 6.855 inches. If this procedure is used with a graphic panel, the values will be scaled to **window inches** automatically.

If the axes must be placed an exact distance from the edge of the page, **axmargin** should be used.

Source pgraph.src

See also **axmargin**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

matalloc

matalloc

a	Purpose	Allocates a matrix with unspecified contents.
b	Format	$y = \text{matalloc}(r,c);$
c	Input	r scalar, rows. c scalar, columns.
d	Output	y $r \times c$ matrix.
e	Remarks	The contents are unspecified. This function is used to allocate a matrix that will be written to in sections using indexing or used with the Foreign Language Interface as an output matrix for a function called with dllcall() .
f	See Also	matinit, ones, zeros, eye
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

matinit

Purpose Allocates a matrix with unspecified contents.

Format `y = matinit(r,c,v);`

Input

<i>r</i>	scalar, rows.
<i>c</i>	scalar, columns.
<i>v</i>	scalar, value to initialize.

Output *y* *rx**c* matrix with each element equal to the value of *v*.

See Also **matalloc, ones, zeros, eye**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

maxc

maxc

a	Purpose	Returns a column vector containing the largest element in each column of a matrix.
b		
c	Format	$y = \text{maxc}(x);$
d	Input	x NxK matrix.
e	Output	y Kx1 matrix containing the largest element in each column of x .
f	Remarks	If x is complex, maxc uses the complex modulus (abs (x)) to determine the largest elements. To find the maximum elements in each row of a matrix, transpose the matrix before applying the maxc function. To find the maximum value in the whole matrix if the matrix has more than one column, nest two calls to maxc : $y = \text{maxc}(\text{maxc}(x));$
g	Example	$x = \text{rndn}(4,2);$ $y = \text{maxc}(x);$ $x =$ <div><div>-2.124474 1.376765</div><div>0.348110 1.172391</div><div>-0.027064 0.796867</div><div>1.421940 -0.351313</div></div> $y =$ <div><div>1.421940</div><div>1.376765</div></div>
h	See also	minc , maxindc , minindc
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

maxindc

Purpose	Returns a column vector containing the index (i.e., row number) of the maximum element in each column in a matrix.		
Format	$y = \text{maxindc}(x);$		
Input	x	NxK matrix.	
Output	y	Kx1 matrix containing the index of the maximum element in each column of x .	
Remarks	<p>If x is complex, maxc uses the complex modulus (abs(x)) to determine the largest elements.</p> <p>To find the index of the maximum element in each row of a matrix, transpose the matrix before applying maxindc.</p> <p>If there are two or more “largest” elements in a column (i.e., two or more elements equal to each other and greater than all other elements), then maxindc returns the index of the first one found, which will be the smallest index.</p>		

Example

```
x = round(rndn(4,4)*5);
y = maxc(x);
z = maxindc(x);
```

```

      1 -11  0  5
x =   0  0 -2 -6
     -8  0  3  2
     -11  5 -4  5
```

```

      1
y =   5
      3
      5
```

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

maxindc

$$z = \begin{matrix} 1 \\ 4 \\ 3 \\ 1 \end{matrix}$$

See also **maxc, minindc, minc**

maxvec

Purpose	Returns maximum vector length allowed.
Format	<code>y = maxvec;</code>
Global Input	<code>__maxvec</code> scalar, maximum vector length allowed.
Output	<code>y</code> scalar, maximum vector length.
Remarks	<p>maxvec returns the value in the global scalar <code>__maxvec</code>, which can be reset in the calling program. This must never be set to 8190.</p> <p>maxvec is called by Run-Time Library functions and applications when determining how many rows can be read from a data set in one call to readr.</p> <p>On systems without virtual memory you can use 536870910. Otherwise a smaller value like 20000-30000 is necessary to prevent excessive disk thrashing. The trick is to allow the algorithm making the disk reads to execute entirely in RAM.</p>
Example	<pre>y = maxvec; print y; produces: 20000.000</pre>
Source	<code>system.src</code>

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

mbesseli

mbesseli

a		
b		
c		
d		
e		
f		
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		
	Purpose	Computes modified and exponentially scaled modified Bessels of the first kind of the n^{th} order.
	Format	$y = \text{mbesseli}(x,n,alpha);$ $y = \text{mbesseli0}(x);$ $y = \text{mbesseli1}(x);$ $y = \text{mbesselei}(x,n,alpha);$ $y = \text{mbesselei0}(x);$ $y = \text{mbesselei1}(x);$
	Input	x Kx1 vector, abscissae. n scalar, highest order. $alpha$ scalar, $0 \leq alpha < 1$.
	Output	y KxN matrix, evaluations of the modified Bessel or the exponentially scaled modified Bessel of the first kind of the n^{th} orders.
	Remarks	For the functions that permit you to specify the order, the returned matrix contains a sequence of modified or exponentially scaled modified Bessel values of different orders. For the i^{th} row of y : $y[i, .] = I_{\alpha}(x[i]) \ I_{\alpha+1}(x[i]) \ \dots \ I_{\alpha+n-1}(x[i])$ The remaining functions generate modified Bessels of only the specified order. The exponentially scaled modified Bessels are related to the unscaled modified Bessels in the following way: $\text{mbesselei0}(x) = \exp(-x) * \text{mbesseli0}(x)$ The use of the scaled versions of the modified Bessel can improve the numerical properties of some calculations by keeping the intermediate numbers small in size.
	Example	This example produces estimates for the “circular” response regression model (Fisher, N.I. <i>Statistical Analysis of Circular Data</i> . NY: Cambridge

University Press, 1993.), where the dependent variable varies between $-\pi$ and π in a circular manner. The model is

$$y = \mu + G(XB)$$

where B is a vector of regression coefficients, X a matrix of independent variables with a column of 1's included for a constant, and y a vector of “circular” dependent variables, and where $G()$ is a function mapping XB onto the $[-\pi, \pi]$ interval.

The log-likelihood for this model is from Fisher, N.I. ... 1993, 159;

$$\log L = -N \times \ln((I_0(\kappa) + \kappa) \sum_i^N \cos(y_i - \mu - G(X_i B)))$$

To generate estimates it is necessary to maximize this function using an iterative method. **QNNewton** is used here.

κ is required to be nonnegative and therefore in the example below, the exponential of this parameter is estimated instead. Also, the exponentially scaled modified Bessel is used to improve numerical properties of the calculations.

The **arctan** function is used in $G()$ to map XB to the $[-\pi, \pi]$ interval as suggested by Fisher, N.I. ... 1993, 158.

```
proc G(u);
    retp(2*atan(u));
endp;

proc lpr(b);
    local dev;
```

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

mbesseli

```
/*
** b[1] - kappa
** b[2] - mu
** b[3] - constant
** b[4:rows(b)] - coefficients
*/
    dev = y - b[2] - G(b[3] + x * b[4:rows(b)]);
    retp(rows(dev)*ln(mbesselei0(exp(b[1])) -
        sumc(exp(b[1])*(cos(dev)-1)))));
endp;

loadm data;
y0 = data[:,1];
x0 = data[:,2:cols(data)];

b0 = 2*ones(cols(x),1);

{ b,fct,grd,ret } = QNewton(&lpr,b0);

cov = invpd(hessp(&lpr,b));

print "estimates    standard errors";
print;
print b~sqrt(diag(cov));
```

Source ribes1.src

mean

Purpose Computes the mean of every column of a matrix.

Format $y = \text{mean}(x);$

Input x NxK matrix.

Output y Kx1 matrix containing the mean of every column of x .

Example $x = \text{mean}(\text{randu}(2000, 4));$

```

0.492446
0.503543
0.502905
0.509283

```

In this example, 4 columns of uniform random numbers are generated in a matrix, and the mean is computed for each column.

See also **std**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

median

median

Purpose	Computes the medians of the columns of a matrix.		
Format	m	<code>= median(x);</code>	
Input	x	NxK matrix.	
Output	m	Kx1 vector containing the medians of the respective columns of x .	
Example	$x = \begin{Bmatrix} 8 & 4, \\ 6 & 8, \\ 3 & 7 \end{Bmatrix};$ $y = \text{median}(x);$ $y = \begin{Bmatrix} 6.0000000 \\ 7.0000000 \end{Bmatrix}$		
Source	median.src		

mergeby

Purpose Merges two sorted files by a common variable.

Format **mergeby**(*infile1*,*infile2*,*outfile*,*keytyp*) ;

Input

<i>infile1</i>	string, name of input file 1.
<i>infile2</i>	string, name of input file 2.
<i>outfile</i>	string, name of output file.
<i>keytyp</i>	scalar, data type of key variable.
1	numeric
2	character

Remarks This will combine the variables in the two files to create a single large file. The following assumptions hold:

1. Both files have the key variable in the first column.
2. All of the values of the key variable within a file are unique.
3. Each file is already sorted on that variable.

The output file will contain the key variable in its first column.

It is not necessary for the two files to have the same number of rows. For each row for which the key variables match, a row will be created in the output file. *outfile* will contain the columns from *infile1* followed by the columns of *infile2* minus the key column from the second file.

If the inputs are null or 0, the procedure will ask for them.

Example `mergeby("freq", "freqdata", "mergex",1);`

Source `sortd.src`

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

mergevar

mergevar

a	Purpose	Accepts a list of names of global matrices, and concatenates the corresponding matrices horizontally to form a single matrix.
b		
c	Format	$x = \text{mergevar}(vnames);$
d		
e	Input	$vnames$ string or $K \times 1$ column vector containing the names of K global matrices.
f		
g	Output	x $N \times M$ matrix that contains the concatenated matrices, where M is the sum of the columns in the K matrices specified in $vnames$.
h		
i	Remarks	The matrices specified in $vnames$ must be globals and they must all have the same number of rows. This function is the opposite of makevars .
j		
k	Example	<pre>let vnames = age pay sex; x = mergevar(vnames);</pre> <p>The matrices age, pay, and sex will be concatenated horizontally to create x.</p>
l		
m		
n	Source	vars.src
o	See also	makevars
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

minc

Purpose	Returns a column vector containing the smallest element in each column in a matrix.
Format	<code>y = minc(x);</code>
Input	<code>x</code> <code>NxK</code> matrix.
Output	<code>y</code> <code>Kx1</code> matrix containing the smallest element in each column of <code>x</code> .
Remarks	<p>If <code>x</code> is complex, minc uses the complex modulus (abs(<code>x</code>)) to determine the smallest elements.</p> <p>To find the minimum element in each row, transpose the matrix before applying the minc function.</p> <p>To find the minimum value in the whole matrix, nest two calls to minc:</p> <pre>y = minc(minc(x));</pre>
Example	<pre>x = randn(4,2); y = minc(x);</pre> $x = \begin{bmatrix} -1.061321 & -0.729026 \\ -0.021965 & 0.184246 \\ 1.843242 & -1.847015 \\ 1.977621 & -0.532307 \end{bmatrix}$ $y = \begin{bmatrix} -1.061321 \\ -1.847015 \end{bmatrix}$
See also	maxc , minindc , maxindc

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

minindc

minindc

Purpose	Returns a column vector containing the index (i.e., row number) of the smallest element in each column in a matrix.		
Format	$y = \text{minindc}(x);$		
Input	x	NxK matrix.	
Output	y	Kx1 matrix containing the index of the smallest element in each column of x .	
Remarks	<p>If x is complex, minindc uses the complex modulus (abs(x)) to determine the smallest elements.</p> <p>To find the index of the smallest element in each row, transpose the matrix before applying minindc.</p> <p>If there are two or more “smallest” elements in a column (i.e., two or more elements equal to each other and less than all other elements), then minindc returns the index of the first one found, which will be the smallest index.</p>		
Example	<pre>x = round(rndn(5,4)*5); y = minc(x); z = minindc(x);</pre> $x = \begin{bmatrix} -5 & 6 & -4 & -1 \\ 2 & -2 & 1 & 3 \\ 6 & 0 & 1 & -7 \\ -6 & 0 & 8 & -4 \\ 7 & -4 & 8 & 3 \end{bmatrix}$ $y = \begin{bmatrix} -6 \\ -4 \\ -4 \\ -7 \end{bmatrix}$		

minindc

```
      4  
x =   5  
      1  
      3
```

See also **maxindc, minc, maxc**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

miss, missrv

miss, missrv

a		
b		
c		
d	Purpose	miss converts specified elements in a matrix to GAUSS's missing value code. missrv is the reverse of this, and converts missing values into specified values.
e	Format	$y = \text{miss}(x,v);$ $y = \text{missrv}(x,v);$
f	Input	x NxK matrix. v LxM matrix, ExE conformable with x .
g		
h	Output	y max(N,L) by max(K,M) matrix.
i	Remarks	For miss , elements in x that are equal to the corresponding elements in v will be replaced with the GAUSS missing value code.
j		For missrv , elements in x that are equal to the GAUSS missing value code will be replaced with the corresponding element of v .
k		For complex matrices, the missing value code is defined as a missing value entry in the real part of the matrix. For complex x , then, miss replaces elements with a ". + 0i" value, and missrv examines only the real part of x for missing values. If, for example, an element of $x = 1 + .i$, missrv will not replace it.
l		These functions act like element-by-element operators. If v is a scalar, for instance -1, then all -1's in x are converted to missing. If v is a row (column) vector with the same number of columns (rows) as x , then each column (row) in x is transformed to missings according to the corresponding element in v . If v is a matrix of the same size as x , then the transformation is done corresponding element by corresponding element.
m		Missing values are given special treatment in the following functions and operators: b/a (matrix division when a is not square and neither a nor b is scalar), counts , ismiss , maxc , maxindc , minc , minindc , miss , missex , missrv , moment , packr , scalmiss , sortc .
n		As long as you know a matrix contains no missings to begin with, miss and missrv can be used to convert one set of numbers into another. For example:
o		$y = \text{missrv}(\text{miss}(x,0),1);$
p		will convert 0's to 1's.
q		
r		
s		
t		
u		
v		
w		
x y z		

miss, missrv

Example `v = -1~4~5;`

`y = miss(x,v);`

If **x** has 3 columns, all -1's in the first column will be changed to missings, along with all 4's in the second column and 5's in the third column.

See also **counts, ismiss, maxc, maxindc, minc, minindc, missex, moment, packr, scalmiss, sortc**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

missex

missex

a	<div>Purpose</div> <div>Converts numeric values to the missing value code according to the values given in a logical expression.</div>	
b		
c		
d		
e		
f	<div>Format</div> <div>$y = \text{missex}(x,e);$</div>	
g	<div>Input</div>	<div>x</div> <div>NxK matrix.</div>
h		<div>e</div> <div>NxK logical matrix (matrix of 0's and 1's) that serves as a "mask" for x; the 1's in e correspond to the values in x that are to be converted into missing values.</div>
i		
j	<div>Output</div> <div>y NxK matrix that equals x, but with those elements that correspond to the 1's in e converted to missing.</div>	
k	<div>Remarks</div>	<div>The matrix e will usually be created by a logical expression. For instance, to convert all numbers between 10 and 15 in x to missing, the following code could be used:</div>
l		<div>$y = \text{missex}(x, (x .> 10) .\text{and} (x .< 15));$</div>
m		<div>Note that "dot" operators MUST be used in constructing the logical expressions.</div>
n		<div>For complex matrices, the missing value code is defined as a missing value entry in the real part of the matrix. For complex x, then, missex replaces elements with a ". + 0i" value.</div>
o		<div>This function is like miss, but is more general in that a range of values can be converted into missings.</div>
p	<div>Example</div>	<div>$x = \text{rndu}(3,2);$</div>
q		<div>$/* \text{ logical expression } */$</div>
r		<div>$e = (x .> .10) .\text{and} (x .< .20);$</div>
s		<div>$y = \text{missex}(x,e);$</div>
t		<div>A 3x2 matrix of uniform random numbers is created. All values in the interval (0.10, 0.20) are converted to missing.</div>
u	<div>Source</div>	<div>datatran.src</div>
v		
w		
x y z	<div>See also</div>	<div>miss, missrv</div>

moment

Purpose	Computes a cross-product matrix. This is the same as $x'x$.		
Format	$y = \text{moment}(x,d);$		
Input	x	NxK matrix.	
	d	scalar, controls handling of missing values.	
	0	missing values will not be checked for. This is the fastest option.	
	1	“listwise deletion” is used. Any row that contains a missing value in any of its elements is excluded from the computation of the moment matrix. If every row in x contains missing values, then moment ($x,1$) will return a scalar zero.	
	2	“pairwise deletion” is used. Any element of x that is missing is excluded from the computation of the moment matrix. Note that this is seldom a satisfactory method of handling missing values, and special care must be taken in computing the relevant number of observations and degrees of freedom.	
Output	y	KxK matrix which equals $x'x$.	
Remarks	The fact that the moment matrix is symmetric is taken into account to cut execution time almost in half.		
	If there is no missing data then $d = 0$ should be used because it will be faster.		
	The / operator (matrix division) will automatically form a moment matrix (performing pairwise deletions if trap 2 is set) and will compute the ols coefficients of a regression. However, it can only be used for data sets that are small enough to fit into a single matrix. In addition, the moment matrix and its inverse cannot be recovered if the / operator is used.		

moment

Example `xx = moment(x,2);`
 `ixx = invpd(xx);`
 `b = ixx*missrv(x,0)'y;`

In this example, the regression of y on x is computed. The moment matrix **xx** is formed using the **moment** command (with pairwise deletion, since the second parameter is 2). Then **xx** is inverted using the **invpd** function. Finally, the **ols** coefficients are computed. **missrv** is used to emulate pairwise deletion by setting missing values to 0.

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

momentd

Purpose	Computes a moment ($X'X$) matrix from a GAUSS data set.	
Format	$m = \text{momentd}(\text{dataset}, \text{vars});$	
Input	<i>dataset</i>	string, name of data set.
	<i>vars</i>	Kx1 character vector, names of variables.
		or Kx1 numeric vector, indices of columns.
	These can be any size subset of the variables in the data set, and can be in any order. If a scalar 0 is passed, all columns of the data set will be used.	
	Defaults are provided for the following global input variables so they can be ignored unless you need control over the other options provided by this procedure.	
Global Input	___con	scalar, default 1.
		1 a constant term will be added.
		0 no constant term will be added.
	___miss	scalar, default 0.
		0 there are no missing values (fastest).
		1 do listwise deletion, drop an observation if any missings occur in it.
		2 do pairwise deletion. This is equivalent to setting missings to 0 when calculating m .
	___row	scalar, default 0, the number of rows to read per iteration of the read loop.
		If 0, the number of rows will be calculated internally.
		If you get an Insufficient memory error message, or you want the rounding to be exactly the same between runs, you can set the number of rows to read before calling momentd .
Output	m	MxM matrix, where $M = K + \text{___con}$, the moment matrix constructed by calculating $X'X$ where X is the data, with or without a constant vector of ones.
		Error handling is controlled by the low order bit of the trap flag.

momentd

trap 0

trap 1

terminate with error message

return scalar error code in *m*

33

too many missings

34

file not found

Example

z = { age, pay, sex };

m = momentd("freq",z);

Source

momentd.src

Globals

__con, __miss, __row

msym

Purpose	Allows the user to set the symbol that GAUSS uses when missing values are converted to ASCII and vice versa.	
Format	msym <i>str</i> ;	
Input	<i>str</i>	literal or ^string (up to 8 letters) which, if not surrounded by quotes, is forced to uppercase. This is the string to be printed for missing values. The default is ‘.’.
Remarks	<p>The entire string will be printed out when converting to ASCII in print, lprint, and printfm statements.</p> <p>When converting ASCII to binary in loadm and let statements, only the first character is significant. In other words,</p> <p style="padding-left: 40px;">msym HAT;</p> <p>will cause ‘H’ to be converted to missing on input.</p> <p>This does not affect writer which outputs data in binary format.</p>	
See also	print , lprint , printfm	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

nametype

nametype

Purpose	Provides support for programs following the upper/lowercase convention in GAUSS data sets. (See “File I/O” in the <i>User’s Guide</i> .) Returns a vector of names of the correct case and a 1/0 vector of type information.
Format	<code>{ vname, vtype } = nametype(vname, vtype);</code>
Input	<div><div><code>vname</code></div><div>Nx1 character vector of variable names.</div></div> <div><div><code>vtype</code></div><div>scalar or Nx1 vector of 1’s and 0’s to determine the type and therefore the case of the output <code>vname</code>. If this is scalar 0 or 1 it will be expanded to Nx1. If -1, nametype will assume that <code>vname</code> follows the upper/lowercase convention.</div></div>
Output	<div><div><code>vname</code></div><div>Nx1 character vector of variable names of the correct case, uppercase if numeric, lowercase if character.</div></div> <div><div><code>vtype</code></div><div>Nx1 vector of ones and zeros, 1 if variable is numeric, 0 if character.</div></div>
Example	<div><code>vn = { age, pay, sex };</code> <code>vt = { 1, 1, 0 };</code> <code>{ vn, vt } = nametype(vn, vt);</code> <code>print \$vn;</code> <div><div>AGE</div><div><code>vn =</code></div><div>PAY</div><div>sex</div></div></div>
Source	<code>nametype.src</code>
Globals	vartype .

new

Purpose Erases everything in memory including the symbol table; closes all open files, the auxiliary output, and turns the window on if it was off; also allows the size of the new symbol table and the main program space to be specified.

Format `new [[nos [,mps]]];`

Input

nos scalar, which indicates the maximum number of global symbols allowed. See your platform supplement for the maximum number of globals allowed in this implementation.

mps scalar, which indicates the number of bytes of main program space to be allocated. See your platform supplement for the maximum amount allowed in this implementation.

Remarks Procedures, user-defined functions, and global matrices strings and string arrays are all global symbols.

The main program space is the amount of memory available for nonprocedure, nonfunction program code.

This command can be used with arguments as the first statement in a program to clear the symbol table and to allocate only as much space for program code as your program actually needs. When used in this manner, the auxiliary output will not be closed. This will allow you to open the auxiliary output from command level and run a program without having to remove the **new** at the beginning of the program. If this command is not the first statement in your program, it will cause the program to terminate.

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

new

Example `new;` `/* clear global symbols. */`

`new 300;` `/* clear global symbols, set */`
 `/* maximum number of global */`
 `/* symbols to 300, and leave */`
 `/* program space unchanged. */`

`new` `/* clear global symbols, set */`
 `200,100000;` `/* maximum number of global */`
 `/* symbols to 200, and allocate */`
 `/* 100000 bytes for main */`
 `/* program code. */`

`new ,100000;` `/* clear global symbols, */`
 `/* allocate 100000 bytes for */`
 `/* main program code, and leave */`
 `/* maximum number of globals */`
 `/* unchanged. */`

See also **clear, delete, output**

nextn, nextnevn

Purpose	Returns allowable matrix dimensions for computing FFT's.		
Format	$n = \text{nextn}(n0);$ $n = \text{nextnevn}(n0);$		
Input	$n0$	scalar, the length of a vector or the number of rows or columns in a matrix.	
Output	n	scalar, the next allowable size for the given dimension for computing an FFT or RFFT. $n \geq n0$.	
Remarks	nextn and nextnevn determine allowable matrix dimensions for computing FFT's. The Temperton FFT routines (see table below) can handle any matrix whose dimensions can be expressed as:		

$$2^p \times 3^q \times 5^r \times 7^s, \quad p, q, r \text{ nonnegative integers} \\ s = 0 \text{ or } 1$$

with one restriction: the vector length or matrix column size must be even (p must be positive) when computing RFFT's.

fftn, etc., automatically pad matrices (with zeros) to the next allowable dimensions; **nextn** and **nextnevn** are provided in case you want to check or fix matrix sizes yourself.

Use the following table to determine what to call for a given function and matrix:

FFT Function	Vector Length	Matrix Rows	Matrix Columns
fftn	nextn	nextn	nextn
rfftn	nextnevn	nextn	nextnevn
rfftnp	nextnevn	nextn	nextnevn

Example `n = nextn(456);`
 `n = 480`

nextn, nextnevn

Source `optim.src`

See also `fftn, optn, optnevn, rfftn, rfftnp`

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

nextwind

Purpose	Sets the current graphic panel to the next available graphic panel.
Library	pgraph
Format	nextwind;
Remarks	<p>This function selects the next available graphic panel to be the current graphic panel. This is the graphic panel in which the next graph will be drawn.</p> <p>See the discussion on using graphic panels in “Publication Quality Graphics” in the <i>User’s Guide</i>.</p>
Source	pwindow.src
See also	endwind, begwind, setwind, getwind, makewind, window

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

null

null

a	Purpose	Computes an orthonormal basis for the (right) null space of a matrix.
b	Format	$b = \text{null}(x);$
c	Input	x NxM matrix.
d	Output	b MxK matrix, where K is the nullity of X, such that: $x*b = 0$ (NxK matrix of zeros) and $b'b = I$ (MxM identity matrix)
e		The error returns are returned in b :
f		error code reason
g		1 there is no null space
h		2 b is too large to return in a single matrix
i		Use scalerr to test for error returns.
j	Remarks	The orthogonal complement of the column space of x' is computed using the QR decomposition. This provides an orthonormal basis for the null space of x .
k	Example	let $x[2,4] = \begin{matrix} 2 & 1 & 3 & -1 \\ 3 & 5 & 1 & 2; \end{matrix}$ $b = \text{null}(x);$ $z = x*b;$ $i = b'b;$
l	Source	null.src
m	Globals	_qrdc, _qrs1
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

null1

Purpose	Computes an orthonormal basis for the (right) null space of a matrix.
Format	<code>nu = null1(x,dataset);</code>
Input	<p><i>x</i> NxM matrix.</p> <p><i>dataset</i> string, the name of a data set null1 will write.</p>
Output	<i>nu</i> scalar, the nullity of <i>x</i> .
Remarks	<p>null1 computes an MxK matrix <i>b</i>, where K is the nullity of <i>x</i>, such that:</p> $x*b = 0 \quad (\text{NxK matrix of zeros})$ <p>and</p> $b'b = I \quad (\text{MxM identity matrix})$ <p>The transpose of <i>b</i> is written to the data set named by <i>dataset</i>, unless the nullity of <i>x</i> is zero. If <i>nu</i> is zero, the data set is not written.</p>
Source	<code>null.src</code>
Globals	<code>_qrdc, _qrs1</code>

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

ols

ols

Purpose	Computes a least squares regression.		
Format	{ vnam,m,b,stb,vc,stderr,sigma,cx,rsq,resid,dwstat } = ols(dataset,depvar,indvars) ;		
Input	dataset	string, name of data set or null string.	
		If dataset is a null string, the procedure assumes that the actual data has been passed in the next two arguments.	
	depvar	If dataset contains a string:	
		string, name of dependent variable.	
		scalar, index of dependent variable. If scalar 0, the last column of the data set will be used.	
		If dataset is a null string or 0:	
		Nx1 vector, the dependent variable.	
	indvars	If dataset contains a string:	
		Kx1 character vector, names of independent variables.	
		Kx1 numeric vector, indices of independent variables.	
	These can be any size subset of the variables in the data set and can be in any order. If a scalar 0 is passed, all columns of the data set will be used except for the one used for the dependent variable.		
	If dataset is a null string or 0:		
	NxK matrix, the independent variables.		
Global Input	Defaults are provided for the following global input variables, so they can be ignored unless you need control over the other options provided by this procedure.		
	__altnam	global vector, default 0.	
		This can be a (K+1)x1 or (K+2)x1 character vector of alternate variable names for the output. If __con is 1, this must be (K+2)x1. The name of the dependent variable is the last element.	
	__con	global scalar, default 1.	
		1	a constant term will be added, D = K+1.
		0	no constant term will be added, D = K.

A constant term will always be used in constructing the moment matrix m .

__miss	global scalar, default 0.
	0 there are no missing values (fastest).
	1 listwise deletion, drop any cases in which missings occur.
	2 pairwise deletion, this is equivalent to setting missings to 0 when calculating m . The number of cases computed is equal to the total number of cases in the data set.
__output	global scalar, default 1.
	1 print the statistics.
	0 do not print statistics.
__row	global scalar, the number of rows to read per iteration of the read loop. Default 0.
	If 0, the number of rows will be calculated internally. If you get an Insufficient memory error message while executing ols , you can supply a value for __row that works on your system.
	The answers may vary slightly due to rounding error differences when a different number of rows is read per iteration. You can use __row to control this if you want to get exactly the same rounding effects between several runs.
_olsres	global scalar, default 0.
	1 compute residuals (<i>resid</i>) and Durbin-Watson statistic(<i>dwstat</i>).
	0 <i>resid</i> = 0, <i>dwstat</i> = 0.

Output

<i>vnam</i>	(K+2)x1 or (K+1)x1 character vector, the variable names used in the regression. If a constant term is used, this vector will be (K+2)x1, and the first name will be “CONSTANT”. The last name will be the name of the dependent variable.								
<i>m</i>	MxM matrix, where M = K+2, the moment matrix constructed by calculating $\mathbf{X}'\mathbf{X}$ where \mathbf{X} is a matrix containing all useable observations and having columns in the order: <table><tr><td>1.0</td><td><i>indvars</i></td><td><i>depvar</i></td></tr><tr><td>(constant)</td><td>(independent variables)</td><td>(dependent variable)</td></tr></table> A constant term is always used in computing <i>m</i> .			1.0	<i>indvars</i>	<i>depvar</i>	(constant)	(independent variables)	(dependent variable)
1.0	<i>indvars</i>	<i>depvar</i>							
(constant)	(independent variables)	(dependent variable)							
<i>b</i>	Dx1 vector, the least squares estimates of parameters.								

ols

Error handling is controlled by the low order bit of the trap flag.

- trap 0** terminate with error message
- trap 1** return scalar error code in *b*
 - 30** system singular
 - 31** system underdetermined
 - 32** same number of columns as rows
 - 33** too many missings
 - 34** file not found
 - 35** no variance in an independent variable

The system can become underdetermined if you use listwise deletion and have missing values. In that case, it is possible to skip so many cases that there are fewer useable rows than columns in the data set.

- stb* Kx1 vector, the standardized coefficients.
- vc* DXD matrix, the variance-covariance matrix of estimates.
- stderr* Dx1 vector, the standard errors of the estimated parameters.
- sigma* scalar, standard deviation of residual.
- cx* (K+1)x(K+1) matrix, correlation matrix of variables with the dependent variable as the last column.
- rsq* scalar, R square, coefficient of determination.
- resid* residuals, $resid = y - x * b$.

If **_olsres** = 1, the residuals will be computed.

If the data is taken from a data set, a new data set will be created for the residuals, using the name in the global string variable **_olsrnam**. The residuals will be saved in this data set as an Nx1 column. The *resid* return value will be a string containing the name of the new data set containing the residuals.

If the data is passed in as a matrix, the *resid* return value will be the Nx1 vector of residuals.
- dwstat* scalar, Durbin-Watson statistic.

Remarks No output file is modified, opened, or closed by this procedure. If you want output to be placed in a file, you need to open an output file before calling **ols**.

Example $y = \{ \begin{array}{l} 2, \\ 3, \\ 1, \\ 7, \\ 5 \end{array} \};$

$x = \{ \begin{array}{lll} 1 & 3 & 2, \\ 2 & 3 & 1, \\ 7 & 1 & 7, \\ 5 & 3 & 1, \\ 3 & 5 & 5 \end{array} \};$

```
output file = ols.out reset;
call ols(0,y,x);
output off;
```

In this example, the output from **ols** was put into a file called `ols.out` as well as being printed in the window. This example will compute a least squares regression of **y** on **x**. The return values were discarded by using a **call** statement.

```
data = "olsdat";
depvar = { score };
indvars = { region,age,marstat };
_olsres = 1;
output file = lpt1 on;
{ nam,m,b,stb,vc,std,sig,cx,rsq,resid,dbw } =
    ols(data,depvar,indvars);
output off;
```

In this example, the data set `olsdat.dat` was used to compute a regression. The dependent variable is **score**. The independent variables are **region**, **age**, and **marstat**. The residuals and Durbin-Watson statistic will be computed. The output will be sent to the printer as well as the window and the returned values are assigned to variables.

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

ols

Source `ols.src`

Globals `_olsres, _olsrnam, __altnam, __con, __miss,`
 `__output, __row, __vpad`

See also `olsqr`

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

olsqr

Purpose	Computes OLS coefficients using QR decomposition.	
Format	$b = \text{olsqr}(y, x);$	
Input	y	Nx1 vector containing dependent variable.
	x	NxP matrix containing independent variables.
Global Input	<code>_olsqtol</code>	global scalar, the tolerance for testing if diagonal elements are approaching zero. The default value is $10e-14$.
Output	b	Px1 vector of least squares estimates of regression of y on x . If x does not have full rank, then the coefficients that cannot be estimated will be zero.
Remarks	This provides an alternative to y/x for computing least squares coefficients.	
	This procedure is slower than the <code>/</code> operator. However, for near singular matrices it may produce better results.	
	olsqr handles matrices that do not have full rank by returning zeros for the coefficients that cannot be estimated.	
Source	<code>olsqr.src</code>	
Globals	<code>_olsqtol</code>	
See also	<code>ols</code> , <code>olsqr2</code> , <code>orth</code> , <code>qqr</code>	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

olsqr2

olsqr2

a	Purpose		Computes OLS coefficients, residuals, and predicted values using the QR decomposition.
b	Format		<code>{ b,r,p } = olsqr2(y,x);</code>
c	Input	y	Nx1 vector containing dependent variable.
d		x	NxP matrix containing independent variables.
e	Global Input		<code>_olsqtol</code> global scalar, the tolerance for testing if diagonal elements are approaching zero. The default value is 10e-14.
f	Output	b	Px1 vector of least squares estimates of regression of y on x. If x does not have full rank, then the coefficients that cannot be estimated will be zero.
g		r	Px1 vector of residuals. ($r = y - x*b$)
h		p	Px1 vector of predicted values. ($p = x*b$)
i	Remarks		This provides an alternative to y/x for computing least squares coefficients.
j			This procedure is slower than the / operator. However, for near singular matrices, it may produce better results.
k			<code>olsqr2</code> handles matrices that do not have full rank by returning zeros for the coefficients that cannot be estimated.
l	Source		<code>olsqr.src</code>
m	Globals		<code>_olsqtol</code>
n	See also		<code>olsqr</code> , <code>orth</code> , <code>qqr</code>
o			
p			
q			
r			
s			
t			
u			
v			
w			
x y z			

ones

Purpose	Creates a matrix of ones.	
Format	$y = \text{ones}(r,c);$	
Input	r	scalar, number of rows.
	c	scalar, number of columns.
Output	y	$R \times C$ matrix of ones.
Remarks	Noninteger arguments will be truncated to an integer.	
Example	$x = \text{ones}(3,2);$ $x = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix}$	
See also	zeros, eye	

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

open

open

a	Purpose	Opens an existing GAUSS data file.	
b			
c			
d	Format	open <i>fh</i> =filename [[for mode]] [[varindxi <i>offs</i>]] ;	
e			
f			
g	Input	<i>filename</i>	literal or ^string. <i>filename</i> is the name of the file on the disk. The name can include a path if the directory to be used is not the current directory. This filename will automatically be given the extension <code>.dat</code> . If an extension is specified, the <code>.dat</code> will be overridden. If the file is an <code>.fmt</code> matrix file, the extension must be explicitly given. If the name of the file is to be taken from a string variable, the name of the string must be preceded by the <code>^</code> (caret) operator.
h		<i>mode</i>	[[read]], append, update The modes supported with the optional for subcommand are: read This is the default file opening mode and will be the one used if none is specified. Files opened in this mode cannot be written to. The pointer is set to the beginning of the file and the writer function is disabled for files opened in this way. This is the only mode available for matrix files (<code>.fmt</code>), which are always written in one piece with the save command. append Files opened in this mode cannot be read. The pointer will be set to the end of the file so that a subsequent write to the file with the writer function will add data to the end of the file without overwriting any of the existing data in the file. The readr function is disabled for files opened in this way. This mode is used to add additional rows to the end of a file. update Files opened in this mode can be read from and written to. The pointer will be set to the beginning of the file. This mode is used to make changes in a file.
i			
j			
k			
l			
m			
n			
o			
p			
q			
r			
s			
t			
u			
v			
w		<i>offs</i>	scalar.
x y z			

The optional **varindxi** subcommand tells GAUSS to create a set of global scalars that contain the index (column position) of the variables in a GAUSS data file. These “index variables” will have the same names as the corresponding variables in the data file but with “**i**” added as a prefix. They can be used inside index brackets, and with functions like **submat** to access specific columns of a matrix without having to remember the column position.

The optional *offs* is an offset that will be added to the index variables. This is useful if data from multiple files are concatenated horizontally in one matrix. It can be any scalar expression. The default is 0.

The index variables are useful for creating submatrices of specific variables without requiring that the positions of the variables be known. For instance, if there are two variables, **xvar** and **yvar** in the data set, the index variables will have the names **ixvar**, **iyvar**. If **xvar** is the first column in the data file, and **yvar** is the second, and if no offset, *offs*, has been specified, then **ixvar** and **iyvar** will equal 1 and 2, respectively. If an offset of 3 had been specified, then these variables would be assigned the values 4 and 5, respectively.

The **varindxi** and **varindx** options cannot be used with **.fmt** matrix files because no column names are stored with them.

If **varindxi** is used, GAUSS will ignore the **Undefined symbol** error message for global symbols that start with “**i**”. This makes it much more convenient to use index variables because they don’t have to be cleared before they are accessed in the program. Clearing is otherwise necessary because the index variables do not exist until execution time when the data file is actually opened and the names are read in from the header of the file. At compile time a statement like: **y=x[.,ixvar];** will be illegal if the compiler has never heard of **ixvar**. If **varindxi** is used, this error will be ignored for symbols beginning with “**i**”. Any symbols that are accessed before they have been initialized with a real value will be trapped at execution time with a **Variable not initialized** error message.

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

open

Output *fh* scalar.
fh is the file handle which will be used by most commands to refer to the file within GAUSS. This file handle is actually a scalar containing an integer value that uniquely identifies each file. This value is assigned by GAUSS when the **open** command is executed. If the file was not successfully opened, the file handle will be set to -1.

Remarks The file must exist before it can be opened with the **open** command. (To create a new file, see **create** or **save**.)

A file can be opened simultaneously under more than one handle. See the second example following.

If the value that is in the file handle when the **open** command begins to execute matches that of an already open file, the process will be aborted and a **File already open** error message will be given. This gives you some protection against opening a second file with the same handle as a currently open file. If this happens, you would no longer be able to access the first file.

It is important to set unused file handles to zero because both **open** and **create** check the value that is in a file handle to see if it matches that of an open file before they proceed with the process of opening a file. This should be done with **close** or **closeall**.

Example

```
fname = "/data/rawdat";  
open dt = ^fname for append;  
if dt == -1;  
    print "File not found";  
end;  
endif;  
y = writer(dt,x);  
if y /= rows(x);  
    print "Disk Full";  
end;  
endif;  
dt = close(dt);
```

In the example above, the existing data set `/data/rawdat.dat` is opened for appending new data. The name of the file was in the string variable **fname**. In this example the file handle is tested to see if the file was opened successfully. The matrix **x** is written to this data set. The number of columns in **x** must be the same as the number of columns in the existing data set. The first row in **x** will be placed after the last row in the existing data set. The **writer** function will return the number of rows actually written. If this does not equal the number of rows that were attempted, then the disk is probably full.

```
open fin = mydata for read;
open fout = mydata for update;
do until eof(fin);
  x = readr(fin,100);
  x[.,1 3] = ln(x[.,1 3]);
  call writer(fout,x);
endo;
closeall fin,fout;
```

In the above example, the same file, `mydata.dat`, is opened twice with two different file handles. It is opened for read with the handle **fin**, and it is opened for update with the handle **fout**. This will allow the file to be transformed in place without taking up the extra space necessary for a separate output file. Notice that **fin** is used as the input handle and **fout** is used as the output handle. The loop will terminate as soon as the input handle has reached the end of the file. Inside the loop the file is read into a matrix called **x** using the input handle, the data are transformed (columns 1 and 3 are replaced with their natural logs), and the transformed data is written back out using the output handle. This type of operation works well as long as the total number of rows and columns does not change.

The following example assumes a data file named `dat1.dat` that has the variables: **visc**, **temp**, **lub**, **rpm**.

```
open f1 = dat1 varindxi;
dtx = readr(f1,100);
x = dtx[.,irpm ilub ivisc];
y = dtx[.,itemp];
call seekr(f1,1);
```

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

open

In this example, the data set `dat1.dat` is opened for reading (the `.dat` and the **for read** are implicit). **varindx** is specified with no constant. Thus, index variables are created that give the positions of the variables in the data set. The first 100 rows of the data set are read into the matrix **dtx**. Then, specified variables in a specified order are assigned to the matrices **x** and **y** using the index variables. The last line uses the **seekr** function to reset the pointer to the beginning of the file.

```
open q1 = dat1 varindx;
open q2 = dat2 varindx colsf(q1);
nr = 100;
y = readr(q1,nr)~readr(q2,nr);
closeall q1,q2;
```

In this example, two data sets are opened for reading and index variables are created for each. A constant is added to the indices for the second data set (**q2**), equal to the number of variables (columns) in the first data set (**q1**). Thus, if there are three variables **x1**, **x2**, **x3** in **q1**, and three variables **y1**, **y2**, **y3** in **q2**, the index variables that were created when the files were opened would be **ix1**, **ix2**, **ix3**, **iy1**, **iy2**, **iy3**. The values of these index variables would be 1, 2, 3, 4, 5, 6, respectively. The first 100 rows of the two data sets are read in and concatenated to give the matrix **y**. The index variables will thus give the correct positions of the variables in **y**.

```
open fx = x.fmt;
i = 1; rf = rowsf(fx);
sampsiz = round(rf*0.1);
randsmpx = zeros(sampsiz,colsf(fx));
do until i > sampsiz;
    r = ceil(rndu(1,1)*rf);
    call seekr(fx,r);
    randsmpx[i,.] = readr(fx,1);
    i = i+1;
endo;
fx = close(fx);
```


open

In this example, a 10% random sample of rows is drawn from the matrix file `x.fmt` and put into the matrix **rndsmplx**. Note that the extension `.fmt` must be specified explicitly in the **open** statement. The **rowsf** command is used to obtain the number of rows in `x.fmt`. This number is multiplied by 0.10 and the result is rounded to the nearest integer; this yields desired sample size. Then random integers (**r**) in the range 1 to **rf** are generated. **seekr** is used to locate to the appropriate row in the matrix, and the row is read with **readr** and placed in the matrix **rndsmplx**. This is continued until the complete sample has been obtained.

See also **create, close, closeall, readr, writer, seekr, eof**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

optn, optnevn

a	Purpose	Returns optimal matrix dimensions for computing FFT's.
b	Format	$n = \text{optn}(n0);$ $n = \text{optnevn}(n0);$
c	Input	$n0$ scalar, the length of a vector or the number of rows or columns in a matrix.
d	Output	n scalar, the next optimal size for the given dimension for computing an FFT or RFFT. $n \geq n0$.
e	Remarks	<p>optn and optnevn determine optimal matrix dimensions for computing FFT's. The Temperton FFT routines (see table following) can handle any matrix whose dimensions can be expressed as:</p> $2^p \times 3^q \times 5^r \times 7^s, \quad p, q, r \text{ nonnegative integers}$ $s = 0 \text{ or } 1$ <p>with one restriction: the vector length or matrix column size must be even (p must be positive) when computing RFFT's.</p> <p>fftn, etc., pad matrices to the next allowable dimensions; however, they generally run faster for matrices whose dimensions are highly composite numbers, that is, products of several factors (to various powers), rather than powers of a single factor. For example, even though it is bigger, a 33600x1 vector can compute as much as 20 percent faster than a 32768x1 vector, because 33600 is a highly composite number, $2^6 \times 3 \times 5^2 \times 7$, whereas 32768 is a simple power of 2, 2^{15}. optn and optnevn are provided so you can take advantage of this fact by hand-sizing matrices to optimal dimensions before computing the FFT.</p>
f		
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

optn, optnevn

Use the following table to determine what to call for a given function and matrix:

FFT Function	Vector Length	Matrix Rows	Matrix Columns
fftn	optn	optn	optn
rfftn	optnevn	optn	optnevn
rfftnp	optnevn	optn	optnevn

Example `n = optn(231);`
 `n = 240.00000`

See also **fftn, nextn, nextnevn, rfftn, rfftnp**

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

orth

orth

a	Purpose	Computes an orthonormal basis for the column space of a matrix.
b	Format	<code>y = orth(x);</code>
c	Input	<code>x</code> NxK matrix.
d	Global Input	<code>_orthtol</code> global scalar, the tolerance for testing if diagonal elements are approaching zero. The default is 1.0e-14.
e	Output	<code>y</code> NxL matrix such that $y'y = \mathbf{eye}(L)$ and whose columns span the same space as the columns of <code>x</code> ; L is the rank of <code>x</code> .
f	Example	<code>x = { 6 5 4,</code> <code>2 7 5 };</code> <code>y = orth(x);</code> <code>y =</code> <code>-0.58123819 -0.81373347</code> <code>-0.81373347 0.58123819</code>
g	Source	<code>qqr.src</code>
h	Globals	<code>_orthtol</code>
i	See also	<code>qqr, olsqr</code>
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

output

Purpose	This command makes it possible to direct the output of print statements to two different places simultaneously. One output device is always the window or standard output. The other can be selected by the user to be any disk file or other suitable output device such as a printer.		
Format	output [[file=filename]] [[on off reset]] ;		
Input	<i>filename</i>	literal or ^string. The file=filename subcommand selects the file or device to which output is to be sent. If the name of the file is to be taken from a string variable, the name of the string must be preceded by the ^ (caret) operator. The default file name is <code>output.out</code> .	
	on, off, reset	literal, mode flag	
		on opens the auxiliary output file or device and causes the results of all print statements to be sent to that file or device. If the file already exists, it will be opened for appending. If the file does not already exist, it will be created.	
		off closes the auxiliary output file and turns off the auxiliary output.	
		reset similar to the on subcommand, except that it always creates a new file. If the file already exists, it will be destroyed and a new file by that name will be created. If it does not exist, it will be created.	
Remarks	<p>After you have written to an output file you have to close the file before you can print it or edit it with the GAUSS editor. Use output off.</p> <p>The selection of the auxiliary output file or device remains in effect until a new selection is made, or until you exit GAUSS. Thus, if a file is named as the output device in one program, it will remain the output device in subsequent programs until a new file=filename subcommand is encountered.</p>		

output

The command **output file=filename;** will select the file or device but will not open it. A subsequent **output on;** or **output reset;** will open it and turn on the auxiliary output.

The command **output off** will close the file and turn off the auxiliary output. The filename will remain the same. A subsequent **output on** will cause the file to be opened again for appending. A subsequent **output reset** will cause the existing file to be destroyed and then recreated and will turn on the auxiliary output.

The command **output** by itself will cause the name and status (i.e., open or closed) of the current auxiliary output file to be printed in the window.

The output to the console can be turned off and on using the **screen off** and **screen on** commands. Output to the auxiliary file or device can be turned off or on using the **output off** or **output on** command. The defaults are **screen on** and **output off**.

The auxiliary file or device can be closed by an explicit **output off** statement, by an **end** statement, or by an interactive **new** statement. However, a **new** statement at the beginning of a program will not close the file. This allows programs with **new** statements in them to be run without reopening the auxiliary output file.

If a program sends data to a disk file, it will execute much faster if the window is off.

The **outwidth** command will set the line width of the output file. The default is 80.

Example `output file = out1.out on;`

This statement will open the file `out1.out` and will cause the results of all subsequent **print** statements to be sent to that file. If `out1.out` already exists, the new output will be appended.

```
output file = out2.out;
output on;
```

This is equivalent to the previous example.

```
output reset;
```

This statement will create a new output file using the current filename. If the file already exists, any data in it will be lost.

```
output file = mydata.asc reset;
screen off;
format /m1/rz 1,8;
open fp = mydata;
do until eof(fp);
    print readr(fp,200);;
endo;
fp = close(fp);
end;
```

The program above will write the contents of the GAUSS file `mydata.dat` into an ASCII file called `mydata.asc`. If there had been an existing file by the name of `mydata.asc`, it would have been overwritten.

The `/m1` parameter in the **format** statement in combination with the `;;` at the end of the **print** statement will cause one carriage return/line feed pair to be written at the beginning of each row of the output file. There will not be an extra line feed added at the end of each 200 row block.

The **end** statement above will automatically perform **output off** and **screen on**.

See also `outwidth`, `screen`, `end`, `new`

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

outtyp (dataloop)

outtyp (dataloop)

a	Purpose	Specifies the precision of the output data set.
b	Format	outtyp <i>num_constant</i> ;
c	Remarks	<i>num_constant</i> must be 2, 4, or 8, to specify integer, single precision, or double precision, respectively. If outtyp is not specified, the precision of the output data set will be that of the input data set. If character data is present in the data set, the precision will be forced to double.
d	Example	<code>outtyp 8;</code>
e		
f		
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

outwidth

Purpose	Specifies the width of the auxiliary output.
Format	outwidth <i>n</i> ;
Remarks	<p><i>n</i> specifies the width of the auxiliary output in columns (characters). After printing <i>n</i> characters on a line, GAUSS will output a line feed.</p> <p>If a matrix is being printed, the line feed sequence will always be inserted between separate elements of the matrix rather than being inserted between digits of a single element.</p> <p><i>n</i> may be any scalar-valued expression in the range of 2-256. Nonintegers will be truncated to an integer. If 256 is used, no additional lines will be inserted.</p> <p>The default is 80 columns.</p>
Example	<pre>outwidth 132;</pre> <p>This statement will change the auxiliary output width to 132 columns.</p>
See also	lpwidth, output, print

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

pacf**pacf**

Purpose Computes sample partial autocorrelations.

Format *rkk* = **pacf**(*y*,*k*,*d*);

Input

<i>y</i>	Nx1 vector, data.
<i>k</i>	scalar, maximum number of partial autocorrelations to compute.
<i>d</i>	scalar, order of differencing.

Output *rkk* Kx1 vector, sample partial autocorrelations.

Example

```

proc pacf(y,k,d);
    local a,l,j,r,t;
    r = acf(y,k,d);
    a = zeros(k,k);
    a[1,1] = r[1];
    t = 1;
    l = 2;
    do while l le k;
        a[l,1] = (r[l]-a[l-1,1:t]*rev(r[1:l-1]))/
            (1-a[l-1,1:t]*r[1:t]);
        j = 1;
        do while j <= t;
            a[l,j] = a[l-1,j] - a[l,1]*a[l-1,l-j];
            j = j+1;
        endo;
        t = t+1;
        l = l+1;
    endo;
end;

```

```
        retp(diag(a));  
    endp;  
Source  tsutil.src
```

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

packr

packr

a	Purpose	Deletes the rows of a matrix that contain any missing values.
b	Format	$y = \text{packr}(x);$
c	Input	x NxK matrix.
d	Output	y LxK submatrix of x containing only those rows that do not have missing values in any of their elements.
e	Remarks	This function is useful for handling missing values by “listwise deletion,” particularly prior to using the / operator to compute least squares coefficients.
f		If all rows of a matrix contain missing values, packr returns a scalar missing value. This can be tested for quickly with the scalmiss function.
g	Example	$x = \text{miss}(\text{ceil}(\text{rndu}(3,3)*10),1);$ $y = \text{packr}(x);$ $x = \begin{matrix} & . & 9 & 10 \\ 4 & 2 & & . \\ 3 & 4 & 9 & \end{matrix}$ $y = \begin{matrix} 3 & 4 & 9 \end{matrix}$ In this example, the matrix x is formed with random integers and missing values. packr is used to delete rows with missing values.

```
open fp = mydata;
obs = 0;
sum = 0;
do until eof(fp);
    x = packr(readr(fp,100));
    if not scalmiss(x);
        obs = obs+rows(x);
        sum = sum+sumc(x);
    endif;
endo;
mean = sum/obs;
```

In this example, the sums of each column in a data file are computed as well as a count of the rows that do not contain any missing values. **packr** is used to delete rows that contain missings and **scalmiss** is used to skip the two sum steps if all the rows are deleted for a particular iteration of the read loop. Then the sums are divided by the number of observations to obtain the means.

See also **scalmiss**, **miss**, **missrv**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

parse

parse

a	Purpose	Parses a string, returning a character vector of tokens.
b	Format	<i>tok</i> = parse (<i>str</i> , <i>delim</i>) ;
c	Input	<i>str</i> string consisting of a series of tokens and/or delimiters. <i>delim</i> NxK character matrix of delimiters that might be found in <i>str</i> .
d	Output	<i>tok</i> Mx1 character vector consisting of the tokens contained in <i>str</i> . All tokens are returned; any delimiters found in <i>str</i> are ignored.
e	Remarks	The tokens in <i>str</i> must be 8 characters or less in size. If they are longer, the contents of <i>tok</i> is unpredictable.
f	See also	token
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

pause

Purpose	Pauses for a specified number of seconds.
Format	pause (<i>sec</i>) ;
Input	<i>sec</i> seconds to pause.
Source	pause.src
See also	wait

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

pdfn

pdfn

a	Purpose	Computes the standard Normal (scalar) probability density function.
b	Format	$y = \text{pdfn}(x);$
c	Input	x NxK matrix.
d	Output	y NxK matrix containing the standard Normal probability density function of x .
e	Remarks	This does not compute the joint Normal density function. Instead, the scalar Normal density function is computed element-by-element. y could be computed by the following GAUSS code: $y = (1/\text{sqrt}(2*\text{pi}))*\text{exp}(-(x.*x)/2);$
f	Example	$x = \text{rndn}(2,2);$ $y = \text{pdfn}(x);$ $= \begin{matrix} -1.828915 & 0.514485 \\ -0.550219 & -0.275229 \end{matrix}$ $y = \begin{matrix} 0.074915 & 0.349488 \\ 0.342903 & 0.384115 \end{matrix}$
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		

pi

Purpose Returns the mathematical constant π .

Format `y = pi;`

Example `format /rdn 16,14;`
`print pi;`

produces:
3.14159265358979

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

pinv

pinv

a		
b	Purpose	Computes the Moore-Penrose pseudo-inverse of a matrix, using the singular value decomposition.
c		This pseudo-inverse is one particular type of generalized inverse.
d	Format	<code>y = pinv(x);</code>
e	Input	<code>x</code> NxM matrix.
f	Global Input	<code>_svdtol</code> global scalar, any singular values less than <code>_svdtol</code> are treated as zero in determining the rank of the input matrix. The default value for <code>_svdtol</code> is 1.0e-13.
g		
h		
i	Output	<code>y</code> MxN matrix that satisfies the 4 Moore-Penrose conditions: $XYX = X$ $YXY = Y$ XY is symmetric YX is symmetric
j		
k		
l		
m	Global Output	<code>_svderr</code> global scalar, if not all of the singular values can be computed <code>_svderr</code> will be nonzero.
n		
o	Example	<code>x = { 6 5 4, 2 7 5 } ;</code> <code>y = pinv(x);</code>
p		
q		0.22017139 -0.16348055
r		<code>y =</code> -0.05207647 0.13447594
s		-0.0151615 0.07712591
t	Source	<code>svd.src</code>
u	Globals	<code>_svdtol, _svderr</code>
v		
w		
x y z		

polar

Purpose	Graphs data using polar coordinates.	
Library	pgraph	
Format	polar(radius,theta);	
Input	<i>radius</i>	Nx1 or NxM matrix. Each column contains the magnitude for a particular line.
	<i>theta</i>	Nx1 or NxM matrix. Each column represents the angle values for a particular line.
Source	polar.src	
See also	xy, logx, logy, loglog, scale, xtics, ytics	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

polychar

polychar

a	Purpose	Computes the characteristic polynomial of a square matrix.
b	Format	$c = \text{polychar}(x);$
c	Input	x NxN matrix.
d	Output	c (N+1)x1 vector of coefficients of the N^{th} order characteristic polynomial of x :
e		$p(z)=c[1]*z^n + c[2]*z^{(n-1)} + ... + c[n]*z + c[n+1];$
f	Remarks	The coefficient of z^n is set to unity ($c[1]=1$).
g	Source	poly.src
h	See also	polymake, polymult, polyroot, polyeval
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

polyeval

Purpose	Evaluates polynomials. Can either be 1 or more scalar polynomials or a single matrix polynomial.	
Format	$y = \text{polyeval}(x, c);$	
Input	x	1xK or NxN; that is, x can either represent K separate scalar values at which to evaluate the (scalar) polynomial(s), or it can represent a single NxN matrix.
	c	(P+1)xK or (P+1)x1 matrix of coefficients of polynomials to evaluate. If x is 1xK, then c must be (P+1)xK. If x is NxN, c must be (P+1)x1. That is, if x is a matrix, it can only be evaluated at a single set of coefficients.
Output	y	Kx1 vector (if c is (P+1)xK) or NxN matrix (if c is (P+1)x1 and x is NxN): $y = (c[1, \cdot] \cdot x^p + c[2, \cdot] \cdot x^{(p-1)} + \dots + c[p+1, \cdot])';$
Remarks	In both the scalar and the matrix case, Horner's rule is used to do the evaluation. In the scalar case, the function recsercp is called (this implements an elaboration of Horner's rule).	
Example	$x = 2;$ $\text{let } c = 1 \ 1 \ 0 \ 1 \ 1;$ $y = \text{polyeval}(x, c);$ The result is 27. Note that this is the decimal value of the binary number 11011.	
	$y = \text{polyeval}(x, 1 \text{zeros}(n, 1));$ This will raise the matrix x to the n^{th} power (e.g: $x*x*x*x*\dots*x$).	
	Source	poly.src
See also	polymake , polychar , polymult , polyroot	

polyint

polyint

a	Purpose	Calculates an N^{th} order polynomial interpolation.
b	Format	$y = \text{polyint}(xa, ya, x);$
c	Input	xa $N \times 1$ vector, X values. ya $N \times 1$ vector, Y values. x scalar, X value to solve for.
d	Global Input	_poldeg global scalar, the degree of polynomial required, default 6.
e	Output	y result of interpolation or extrapolation.
f	Global Output	_polerr global scalar, interpolation error.
g	Remarks	Calculates an N^{th} order polynomial interpolation or extrapolation of X on Y given the vectors xa and ya and the scalar x . The procedure uses Neville's algorithm to determine an up to N^{th} order polynomial and an error estimate. Polynomials above degree 6 are not likely to increase the accuracy for most data. Test _polerr to determine the required _poldeg for your problem.
h	Source	polyint.src
i	Technical Notes	Press, W.P., B.P. Flannery, S.A. Tevkolsky, and W.T. Vettering. <i>Numerical Recipes: The Art of Scientific Computing</i> . NY: Cambridge Press, 1986.
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

polymake

Purpose	Computes the coefficients of a polynomial given the roots.	
Format	$c = \text{polymake}(r);$	
Input	r	$N \times 1$ vector containing roots of the desired polynomial.
Output	c	<p>$(N+1) \times 1$ vector containing the coefficients of the N^{th} order polynomial with roots r:</p> $p(z) = c[1] \cdot z^N + c[2] \cdot z^{(N-1)} + \dots + c[n] \cdot z + c[n+1];$
Remarks	The coefficient of z^N is set to unity ($c[1]=1$).	
Example	<pre> r = { 2, 1, 3 }; c = polymake(r); -1.0000000 -6.0000000 11.0000000 -6.0000000 </pre>	
Source	poly.src	
See also	polychar, polymult, polyroot, polyeval	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

polymat

polymat

Purpose	Returns a matrix containing the powers of the elements of x from 1 to p .	
Format	$y = \text{polymat}(x,p);$	
Input	x	NxK matrix.
	p	scalar, positive integer.
Output	y	Nx(p *K) matrix containing powers of the elements of x from 1 to p . The first K columns will contain first powers, the second K columns contain the second powers, and so on.
Remarks	To do polynomial regression use ols : $\{ \text{vnam},m,b,stb,vc,stderr,sigma,cx,rsq,resid, dwstat \} = \text{ols}(0,y,\text{polymat}(x,p));$	
Source	polymat.src	

polymroot

Purpose	Computes the roots of the determinant of a matrix polynomial
Format	$r = \text{polymroot}(c);$
Input	c (N+1)*KxK matrix of coefficients of an Nth order polynomial of rank K.
Output	r K*N vector containing the roots of the determinantal equation.
Remarks	<p>c is constructed of N+1 KxK coefficient matrices stacked vertically with the coefficient matrix of the t^n at the top, $t^{(n-1)}$ next, down to the t^0 matrix at the bottom.</p> <p>Note that this procedure solves the scalar problem as well, that is, the one that POLYROOT solves.</p>
Example	<p>Solve $\det(A2*t^2 + A1*t + A0) = 0$ where:</p> <pre> A2 = [1 2] [2 1] A1 = [5 8] [10 7] A0 = [3 4] [6 5] a2 = { 1 2, 2 1 }; a1 = { 5 8, 10 7 }; a0 = { 3 4, 6 5 }; print polymroot(a2 a1 a0); -4.3027756 -.69722436 -2.6180340 -.38196601 </pre>

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

polymult

polymult

a	Purpose	Multiplies polynomials.
b	Format	<code>c = polymult(c1,c2);</code>
c	Input	<code>c1</code> (D1+1)x1 vector containing the coefficients of the first polynomial.
d		<code>c2</code> (D2+1)x1 vector containing the coefficients of the second polynomial.
e	Output	<code>c</code> (D1+D2)x1 vector containing the coefficients of the product of the two polynomials.
f	Example	<code>c1 = { 2, 1 }; c2 = { 2, 0, 1 }; c = polymult(c1,c2); 4.0000000 c = 2.0000000 2.0000000 1.0000000</code>
g	Source	<code>poly.src</code>
h	See also	<code>polymake, polychar, polyroot, polyeval</code>
i	Technical Notes	If the degree of <code>c1</code> is <code>D1</code> (e.g., if <code>D1=3</code> , then the polynomial corresponding to <code>c1</code> is cubic), then there must be <code>D1+1</code> elements in <code>c1</code> (e.g., 4 elements for a cubic). Thus, for instance the coefficients for the polynomial $5x^3 + 6x + 3$ would be: <code>c1=5 0 6 3</code> . (Note that zeros must be explicitly given if there are powers of x missing.)
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

polyroot

Purpose	Computes the roots of a polynomial given the coefficients.	
Format	<code>y = polyroot(c);</code>	
Input	<i>c</i>	<p>(N+1)x1 vector of coefficients of an N^{th} order polynomial:</p> $p(z)=c[1]*z^n + c[2]*z^{(n-1)} + ...+c[n]*z + c[n+1]$ <p>Zero leading terms will be stripped from <i>c</i>. When that occurs the order of <i>y</i> will be the order of the polynomial after the leading zeros have been stripped.</p> <p><i>c</i>[1] need not be normalized to unity.</p>
Output	<i>y</i>	Nx1 vector, the roots of <i>c</i> .
Source	poly.src	
See also	polymake, polychar, polymult, polyeval	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

pop

pop

a		
b	Purpose	Provides access to a last-in, first-out stack for matrices.
c	Format	pop <i>b</i> ; pop <i>a</i> ;
d	Remarks	This is used with gosub , goto , and return statements with parameters. It permits passing parameters to subroutines or labels, and returning parameters from subroutines.
e		
f		The gosub syntax allows an implicit push statement. This syntax is almost the same as that of a standard gosub , except that the matrices to be push 'ed "into the subroutine" are in parentheses following the label name. The matrices to be push 'ed back to the main body of the program are in parentheses following the return statement. The only limit on the number of matrices that can be passed to and from subroutines in this way is the amount of room on the stack.
g		
h		No matrix expressions can be executed between the (implicit) push and the pop . Execution of such expressions will alter what is on the stack.
i		Matrices must be pop 'ed in the reverse order that they are push 'ed, therefore the statements:
j		
k		<pre>goto label(x,y,z);</pre>
l		<pre> .</pre>
m		<pre> .</pre>
n		<pre> .</pre>
o		
p		<pre>label:</pre>
q		<pre> pop c;</pre>
r		<pre> pop b;</pre>
s		<pre> pop a;</pre>
t		<pre> c = z b = y a = x</pre>
u		
v		Note that matrices are pop 'ed in reverse order, and that there is a separate pop statement for each matrix popped.
w	See also	gosub , goto , return
x y z		

pqqwin

Purpose	Sets the graphics viewer mode.
Library	pgraph
Format	pqqwin <i>arg</i> ;
Input	<i>arg</i> string literal. "one" Use only one viewer. "many" Use a new viewer for each graph.
Remarks	If "one" is set, the viewer executable will be <code>vwr.exe</code> . "manual" and "auto" are supported for backwards compatibility, manual=one, auto=many.
Example:	<pre>pqqwin one; pqqwin many;</pre>
Source	<code>pgraph.src</code>
See also	setvwrmode

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

prcsn

setvwrmode

prcsn

Purpose	Sets the computational precision of some of the matrix operators.
Format	prcsn <i>n</i> ;
Input	<i>n</i> scalar, 64 or 80.
Portability	UNIX, Windows This function has no effect under UNIX or Windows. All computations are done in 64-bit precision.
Remarks	<i>n</i> is a scalar containing either 64 or 80. The operators affected by this command are chol , solpd , invpd , and <i>b/a</i> (when neither <i>a</i> nor <i>b</i> is scalar and <i>a</i> is not square). prcsn 80 is the default. Precision is set to 80 bits (10 bytes), which corresponds to about 19 digits of precision. prcsn 64 sets the precision to 64 bits (8 bytes), which is standard IEEE double precision. This corresponds to 15-16 digits of precision. 80-bit precision is still maintained within the 80x87 math coprocessor so that actual precision is better than double precision. When prcsn 80 is in effect, all temporary storage and all computations for the operators listed above are done in 80 bits. When the operator is finished, the final result is rounded to 64-bit double precision.
See also	chol , solpd , invpd

princomp

Purpose	Computes principal components of a data matrix.	
Format	$\{p,v,a\} = \text{princomp}(x,j);$	
Input	x	NxK data matrix, $N > K$, full rank.
	j	scalar, number of principal components to be computed ($J \leq K$).
Output	p	NxJ matrix of the first j principal components of x in descending order of amount of variance explained.
	v	Jx1 vector of fractions of variance explained.
	a	JxK matrix of factor loadings, such that $x = p*a$ +error.
Remarks	Adapted from a program written by Mico Loretan.	
	The algorithm is based on Theil, Henri “Principles of Econometrics.” Wiley, NY, 1971, 46-56.	

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

print

print

a	Purpose	Prints matrices or strings to the window and/or auxiliary output.	
b	Format	print [[/flush]] [[/typ]] [[/fnted]] [[/mf]] [[/jnt]] <i>list_of_expressions</i> [:,];	
c	Input	<i>/typ</i>	literal, symbol type flag.
d		<i>/mat, /sa,</i>	Indicate which symbol types you are setting the output format for: matrices (/mat), string arrays (/sa), and/or strings (/str). You can specify more than one <i>/typ</i> flag; the format will be set for all types indicated. If no <i>/typ</i> flag is listed, print assumes /mat .
e		<i>/str</i>	
f			
g		<i>/fnted</i>	literal, enable formatting flag.
h		<i>/on, /off</i>	Enable/disable formatting. When formatting is disabled, the contents of a variable are dumped to the window in a “raw” format.
i		<i>/mf</i>	literal, matrix format. It controls the way rows of a matrix are separated from one another. The possibilities are:
j		<i>/m0</i>	no delimiters before or after rows when printing out matrices.
k		<i>/m1 or /mb1</i>	print 1 carriage return/line feed pair before each row of a matrix with more than 1 row.
l		<i>/m2 or /mb2</i>	print 2 carriage return/line feed pairs before each row of a matrix with more than 1 row.
m		<i>/m3 or /mb3</i>	print “Row 1”, “Row 2”... before each row of a matrix with more than one row.
n		<i>/ma1</i>	print 1 carriage return/line feed pair after each row of a matrix with more than 1 row.
o		<i>/ma2</i>	print 2 carriage return/line feed pairs after each row of a matrix with more than 1 row.
p			
q			
r			
s			
t			
u			
v			
w			
x y z			

print

/a1	print 1 carriage return/line feed pair after each row of a matrix.
/a2	print 2 carriage return/line feed pairs after each row of a matrix.
/b1	print 1 carriage return/line feed pair before each row of a matrix.
/b2	print 2 carriage return/line feed pairs before each row of a matrix.
/b3	print “Row 1”, “Row 2”... before each row of a matrix.
/jnt	literal, controls justification, notation, and the trailing character.

Right-Justified

/rd	Signed decimal number in the form [] #####.##### where ##### is one or more decimal digits. The number of digits before the decimal point depends on the magnitude of the number, and the number of digits after the decimal point depends on the precision. If the precision is 0, no decimal point will be printed.
/re	Signed number in the form [] #.##E±###, where # is one decimal digit, ## is one or more decimal digits depending on the precision, and ### is three decimal digits. If precision is 0, the form will be [] #E±### with no decimal point printed.
/ro	This will give a format like /rd or /re depending on which is most compact for the number being printed. A format like /re will be used only if the exponent value is less than -4 or greater than the precision. If a /re format is used a decimal point will always appear. The precision signifies the number of significant digits displayed.

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

print**/rz**

This will give a format like **/rd** or **/re** depending on which is most compact for the number being printed. A format like **/re** will be used only if the exponent value is less than -4 or greater than the precision. If a **/re** format is used, trailing zeros will be suppressed and a decimal point will appear only if one or more digits follow it. The precision signifies the number of significant digits displayed.

Left-Justified**/ld**

Signed decimal number in the form `[[]] #####.####`, where `####` is one or more decimal digits. The number of digits before the decimal point depends on the magnitude of the number, and the number of digits after the decimal point depends on the precision. If the precision is 0, no decimal point will be printed. If the number is positive, a space character will replace the leading minus sign.

/le

Signed number in the form `[[]] #.##E±###`, where `#` is one decimal digit, `##` is one or more decimal digits depending on the precision, and `###` is three decimal digits. If precision is 0, the form will be `[[]] #E±###` with no decimal point printed. If the number is positive, a space character will replace the leading minus sign.

/lo

This will give a format like **/ld** or **/le** depending on which is most compact for the number being printed. A format like **/le** will be used only if the exponent value is less than -4 or greater than the precision. If a **/le** format is used a decimal point will always appear. If the number is positive, a space character will replace the leading minus sign. The precision specifies the number of significant digits displayed.

print

/lz This will give a format like **/ld** or **/le** depending on which is most compact for the number being printed. A format like **/le** will be used only if the exponent value is less than -4 or greater than the precision. If a **/le** format is used, trailing zeros will be suppressed and a decimal point will appear only if one or more digits follow it. If the number is positive, a space character will replace the leading minus sign. The precision specifies the number of significant digits displayed.

Trailing Character

The following characters can be added to the */jnt* parameters above to control the trailing character if any:

```
format /rdn 1,3;
```

s The number will be followed immediately by a space character. This is the default.

c The number will be followed immediately with a comma.

t The number will be followed immediately with a tab character.

n No trailing character.

The default when GAUSS is first started is:

```
format /ml /r0 16,8;
```

;; Double semicolons following a **print** statement will suppress the final carriage return/line feed.

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

print

Remarks

The list of expressions **MUST** be separated by spaces. In **print** statements, because a space is the delimiter between expressions, NO SPACES are allowed inside expressions unless they are within index brackets, quotes, or parentheses.

The printing of special characters is accomplished by the use of the backslash (\) within double quotes. The options are:

- \b** backspace (ASCII 8)
- \e** escape (ASCII 27)
- \f** form feed (ASCII 12)
- \g** beep (ASCII 7)
- \l** line feed (ASCII 10)
- \r** carriage return (ASCII 13)
- \t** tab (ASCII 9)
- \###** the character whose ASCII value is “###” (decimal).

Thus, **\13\10** is a carriage return/line feed sequence. The first three digits will be picked up here. So if the character to follow a special character is a digit, be sure to use three digits in the escape sequence. For example: **\0074** will be interpreted as 2 characters (ASCII 7 followed by the character “4”).

An expression with no assignment operator is an implicit **print** statement.

If **output on** has been specified, then all subsequent **print** statements will be directed to the auxiliary output as well as the window. (See **output**.) The **locate** statement has no effect on what will be sent to the auxiliary output, so all formatting must be accomplished using tab characters or some other form of serial output.

If the name of the symbol to be printed is prefixed with a ‘\$’, it is assumed that the symbol is a matrix of characters.

`print $x;`

Note that GAUSS makes no distinction between matrices containing character data and those containing numeric data, so it is the responsibility of the user to use functions which operate on character matrices only on those matrices containing character data.

These matrices of character strings have a maximum of 8 characters per element. A precision of 8 or more should be set when printing out character matrices or the elements will be truncated.

print

Complex numbers are printed with the sign of the imaginary half separating them and an “i” appended to the imaginary half. Also, the current field width setting (see **format**) refers to the width of field for each half of the number, so a complex number printed with a field of 8 will actually take (at least) 20 spaces to print.

A **print** statement by itself will cause a blank line to be printed:

```
print;
```

GAUSS also has an *automatic print mode* which causes the results of all global assignment statements to be printed out. This is controlled by the **print on** and **print off** commands. (See **print on**.)

Example

```
x = rndn(3,3);
format /rd 16,8;
print x;
format /re 12,2;
print x;
print /rd/m3 x;
```

0.14357994	-1.39272762	-0.91942414
0.51061645	-0.02332207	-0.02511298
-1.54675893	-1.04988540	0.07992059
1.44E-001	-1.39E+000	-9.19E-001
5.11E-001	-2.33E-002	-2.51E-002
-1.55E+000	-1.05E+000	7.99E-002
Row 1		
	0.14	-1.39
Row 2		
	0.51	-0.02
Row 3		
	-1.55	-1.05
		0.08

print

In this example, a 3x3 random matrix is printed using 3 different formats. Notice that in the last statement the format is overridden in the **print** statement itself but the field and precision remain the same.

```
let x = AGE PAY SEX;
```

```
format /m1 8,8;
```

```
print $x;
```

produces:

```
      AGE
```

```
      PAY
```

```
      SEX
```

See also `lprint`, `print on`, `lprint on`, `printfm`, `printdos`

printdos

Purpose	Prints a string to the standard output.
Format	printdos <i>s</i> ;
Input	<i>s</i> string, containing the string to be printed to the standard output.
Remarks	<p>This function is useful for printing messages to the window when screen off is in effect. The output of this function will not go to the auxiliary output.</p> <p>This function can also be used to send escape sequences to the <code>ansi.sys</code> device driver.</p>
Example	<pre>printdos "\27[7m"; /* set for reverse video */ printdos "\27[0m"; /* set for normal text */</pre> <p>See the DOS manuals for more complete information.</p>
See also	print, lprint, printfm, screen

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

printfm

printfm

a	Purpose		Prints a matrix using a different format for each column of the matrix.
b	Format		$y = \text{printfm}(x, \text{mask}, \text{fmt});$
c	Input	x	NxK matrix which is to be printed and which may contain both character and numeric data.
d		mask	LxM matrix, ExE conformable with x , containing ones and zeros which is used to specify whether the particular row, column, or element is to be printed as a string (0) or numeric (1) value.
e		fmt	Kx3 or 1x3 matrix where each row specifies the format for the respective column of x .
f	Output		y scalar, 1 if the function is successful and 0 if it fails.
g	Remarks	The mask is applied to the matrix x following the rules of standard element-by-element operations. If the corresponding element of mask is 0, then that element of x is printed as a character string of up to 8 characters. If mask contains a 1, then that element of x is assumed to be a double precision floating point number.	
h		The contents of fmt are as follows:	
i		[K,1]	format string, a string 8 characters maximum.
j		[K,2]	field width, a number < 80.
k		[K,3]	precision, a number < 17.
l		The format strings correspond to the format slash commands as follows:	
m		/rdn	"*.*1f"
n		/ren	"*.*1E"
o		/ron	"#*.*1G"
p		/rzn	"*.*1G"
q		/ldn	"-*.*1f"
r		/len	"-*.*1E"
s		/lon	"-#*.*1G"
t		/lzn	"-*.*1G"
u			
v			
w			
x y z			

printfm

Complex numbers are printed with the sign of the imaginary half separating them and an “i” appended to the imaginary half. The field width refers to the width of field for each half of the number, so a complex number printed with a field of 8 will actually take (at least) 20 spaces to print.

If the precision = 0, the decimal point will be suppressed.

The format string can be a maximum of 8 characters and is appended to a % sign and passed directly to the **fprintf** function in the standard C language I/O library. The **lf**, etc., are case sensitive. If you know C, you will easily be able to use this.

If you want special characters to be printed after *x*, then include them as the last characters of the format string. For example:

```

"*. *lf,"   right-justified decimal followed by a comma.
"-*. *s"    left-justified string followed by a space.
"*. *lf"    right-justified decimal followed by nothing.

```

If you want the beginning of the field padded with zeros, then put a “0” before the first “*” in the format string:

```

"0*. *lf"   right-justified decimal

```

Example Here is an example of **printfm** being used to print a mixed numeric and character matrix:

```

let x[4,3] =
"AGE" 5.12345564 2.23456788
"PAY" 1.23456677 1.23456789
"SEX" 1.14454345 3.44718234
"JOB" 4.11429432 8.55649341;

let mask[1,3] = 0 1 1; /* character numeric */
                      /* numeric */

let fmt[3,3] =
"-*. *s" 8 8 /* first column format */
"*. *lf," 10 3 /* second column format */

```

printfm

```
"*.*le" 12 4; /* third column format */
```

```
d = printfm(x,mask,fmt);
```

The output looks like this:

```
AGE 5.123, 2.2346E+000
```

```
PAY 1.235, 1.2346E+000
```

```
SEX 1.145, 3.4471E+000
```

```
JOB 4.114, 8.5564E+000
```

When the column of *x* to be printed contains all string elements, use a format string of **"*.*s"** if you want it right-justified, or **"-*.*s"** if you want it left-justified. If the column is mixed string and numeric elements, then use the correct numeric format and **printfm** will substitute a default format string for those elements in the column that are strings.

Remember, the mask value controls whether an element will be printed as a number or a string.

See also **print**, **lprint**, **printdos**

printfmt

Purpose	Prints character, numeric, or mixed matrix using a default format controlled by the functions formatcv and formatnv .
Format	<code>y = printfmt(x,mask);</code>
Input	<p><i>x</i> NxK matrix which is to be printed.</p> <p><i>mask</i> scalar, 1 if <i>x</i> is numeric or 0 if <i>x</i> is character.</p> <p> or</p> <p> 1xK vector of 1's and 0's.</p> <p> The corresponding column of <i>x</i> will be printed as numeric where <i>mask</i> = 1 and as character where <i>mask</i> = 0.</p>
Output	<i>y</i> scalar, 1 if the function is successful and 0 if it fails.
Remarks	<p>Default format for numeric data is: "*.*lg" 16 8</p> <p>Default format for character data is: "*.*s" 8 8</p>
Example	<pre>x = rndn(5,4); call printfmt(x,1);</pre>
Source	gauss.src
Globals	__fmtcv, __fmtnv
See also	formatcv, formatnv

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

proc

proc

a		
b		
c		
d		
e		
f		
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

Purpose	Begins the definition of a multi-line recursive procedure. Procedures are user-defined functions with local or global variables.
Format	proc [(<i>nrets</i>) =] <i>name</i> (<i>arglist</i>) ;
Input	<i>nrets</i> constant, number of objects returned by the procedure. If <i>nrets</i> is not explicitly given, the default is 1. Legal values are 0 to 1023. The retp statement is used to return values from a procedure. <i>name</i> literal, name of the procedure. This name will be a global symbol. <i>arglist</i> a list of names, separated by commas, to be used inside the procedure to refer to the arguments that are passed to the procedure when the procedure is called. These will always be local to the procedure, and cannot be accessed from outside the procedure or from other procedures.
Remarks	A procedure definition begins with the proc statement and ends with the endp statement. An example of a procedure definition is: <pre>proc dog(x,y,z); /* procedure declaration */ local a,b; /* local variable declarations */ a = x .* x; b = y .* y; a = a ./ x; b = b ./ y; z = z .* z; z = inv(z); retp(a'b*z); /* return with value of */ /* a'b*z */ endp; /* end of procedure definition */</pre>

proc

Procedures can be used just as if they were functions intrinsic to the language. Below are the possible variations depending on the number of items the procedure returns.

Returns 1 item:

```
y = dog(i,j,k);
```

Returns multiple items:

```
{ x,y,z } = cat(i,j,k);
```

Returns no items:

```
fish(i,j,k);
```

If the procedure does not return any items or you want to discard the returned items:

```
call dog(i,j,k);
```

Procedure definitions may not be nested.

For more details on writing procedures, see “Procedures and Keywords” in the *User’s Guide*.

See also **keyword, call, endp, local, retp**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

prodc

prodc

a	Purpose	Computes the products of all elements in each column of a matrix.
b	Format	$y = \text{prodc}(x);$
c	Input	x NxK matrix.
d	Output	y Kx1 matrix containing the products of all elements in each column of x .
e	Remarks	To find the products of the elements in each row of a matrix, transpose before applying prodc . If x is complex, use the bookkeeping transpose (\cdot').
f		To find the products of all of the elements in a matrix, use the vecr function before applying prodc .
g	Example	<pre>let x[3,3] = 1 2 3 4 5 6 7 8 9; y = prod(x); 28 y = 80 162</pre>
h	See also	sumc, meanc, stdc
i		
j		
k		
l		
m		
n		
o		
p		

q		
r		
s		
t		
u		
v		
w		
x y z		

putf

Purpose	Writes the contents of a string to a file.
Format	<i>ret</i> = putf (<i>filename</i> , <i>str</i> , <i>start</i> , <i>len</i> , <i>mode</i> , <i>append</i>) ;
Input	<p><i>filename</i> string, name of output file.</p> <p><i>str</i> string to be written to <i>filename</i>. All or part of <i>str</i> may be written out.</p> <p><i>start</i> scalar, beginning position in <i>str</i> of output string.</p> <p><i>len</i> scalar, length of output string.</p> <p><i>mode</i> scalar, output mode, (0) ASCII or (1) binary.</p> <p><i>append</i> scalar, file write mode, (0) overwrite or (1) append.</p>
Output	<i>ret</i> scalar, return code.
Remarks	<p>If <i>mode</i> is set to (1) binary, a string of length <i>len</i> will be written to <i>filename</i>. If <i>mode</i> is set to (0) ASCII, the string will be output up to length <i>len</i> or until putf encounters a ^Z (ASCII 26) in <i>str</i>. The ^Z will not be written to <i>filename</i>.</p> <p>If <i>append</i> is set to (0) overwrite, the current contents of <i>filename</i> will be destroyed. If <i>append</i> is set to (1) append, <i>filename</i> will be created if it does not already exist.</p> <p>If an error occurs, putf will either return an error code or terminate the program with an error message, depending on the trap state. If bit 2 (the 4's bit) of the trap flag is 0, putf will terminate with an error message. If bit 2 of the trap flag is 1, putf will return an error code. The value of the trap flag can be tested with trapchk.</p>

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

putf

ret can have the following values:

- 0** normal return
- 1** null file name
- 2** file open error
- 3** file write error
- 4** output string too long
- 5** null output string, or illegal *mode* value
- 6** illegal *append* value
- 16** append specified but file did not exist; file was created (warning only)

Source `putf.src`

See also **getf**

QNewton

Purpose	Optimizes a function using the BFGS descent algorithm.		
Format	{ <i>x,f,g,ret</i> } = QNewton (<i>&fct,start</i>);		
Input	<i>&fct</i>	pointer to a procedure that computes the function to be minimized. This procedure must have one input argument, a vector of parameter values, and one output argument, the value of the function evaluated at the input vector of parameter values.	
	<i>start</i>	Kx1 vector, start values.	
Global Input	_qn_RelGradTol	scalar, convergence tolerance for relative gradient of estimated coefficients. Default = 1e-5.	
	_qn_GradProc	scalar, pointer to a procedure that computes the gradient of the function with respect to the parameters. This procedure must have a single input argument, a Kx1 vector of parameter values, and a single output argument, a Kx1 vector of gradients of the function with respect to the parameters evaluated at the vector of parameter values. If _qn_GradProc is 0, QNewton uses gradp .	
	_qn_MaxIters	scalar, maximum number of iterations. Default = 1e+5. Termination can be forced by pressing C on the keyboard.	
	_qn_PrintIters	scalar, if 1, print iteration information. Default = 0. Can be toggled during iterations by pressing P on the keyboard.	
	_qn_ParNames	Kx1 vector, labels for parameters.	
	_qn_PrintResults	scalar, if 1, results are printed.	
Output	<i>x</i>	Kx1 vector, coefficients at the minimum of the function.	
	<i>f</i>	scalar, value of function at minimum.	
	<i>g</i>	Kx1 vector, gradient at the minimum of the function.	
	<i>ret</i>	scalar, return code. 0 normal convergence	

QNewton

- 1 forced termination
- 2 max iterations exceeded
- 3 function calculation failed
- 4 gradient calculation failed
- 5 step length calculation failed
- 6 function cannot be evaluated at initial parameter values

Remarks If you are running in terminal mode, GAUSS will not see any input until you press ENTER. Pressing C on the keyboard will terminate iterations, and pressing P will toggle iteration output.

To reset global variables for this function to their default values, call **qnewtonset**.

Example This example computes maximum likelihood coefficients and standard errors for a Tobit model:

```
/*
** qnewton.e - a Tobit model
*/

z = loadadd("tobit"); /* get data */
b0 = { 1, 1, 1, 1 };
{b,f,g,retcode} = qnewton(&lpr,b0);

/*
** covariance matrix of parameters
*/

h = hessp(&lpr,b);

output file = qnewton.out reset;

print "Tobit Model";
print;
```

```

print "coefficients standard errors";
print b~sqrt(diag(invpcd(h)));

output off;

/*
** log-likelihood proc
*/

proc lpr(b);
    local s,m,u;
    s = b[4];
    if s ≤ 1e-4;
        retp(error(0));
    endif;
    m = z[.,2:4]*b[1:3,.];
    u = z[.,1] ./= 0;
    retp(-sumc(u.*lnpdfn2(z[.,1]-m,s) +
                        (1-u).*(ln(cdfnc(m/sqrt(s))))));
endp;

produces:
Tobit Model
coefficients standard errors
           0.010417884   0.080220019
          -0.20805753   0.094551107
         -0.099749592   0.080006676
           0.65223067   0.099827309

```

Source qnewton.src

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

QProg

QProg

a	Purpose	Solves the quadratic programming problem.
b	Format	{ <i>x,u1,u2,u3,u4,ret</i> } = QProg (<i>start,q,r,a,b,c,d,bnds</i>) ;
c	Input	
d	<i>start</i>	Kx1 vector, start values.
e	<i>q</i>	KxK matrix, symmetric model matrix.
f	<i>r</i>	Kx1 vector, model constant vector.
g	<i>a</i>	MxK matrix, equality constraint coefficient matrix, or scalar 0, no equality constraints.
h	<i>b</i>	Mx1 vector, equality constraint constant vector, or scalar 0, will be expanded to Mx1 vector of zeros.
i	<i>c</i>	NxK matrix, inequality constraint coefficient matrix, or scalar 0, no inequality constraints.
j	<i>d</i>	Nx1 vector, inequality constraint constant vector, or scalar 0, will be expanded to Nx1 vector of zeros.
k	<i>bnds</i>	Kx2 matrix, bounds on <i>x</i> , the first column contains the lower bounds on <i>x</i> , and the second column the upper bounds. If scalar 0, the bounds for all elements will default to $\pm 1e200$.
l	Global Input	
m	_qprog_maxit	scalar, maximum number of iterations. Default = 1000.
n	Output	
o	<i>x</i>	Kx1 vector, coefficients at the minimum of the function.
p	<i>u1</i>	Mx1 vector, Lagrangian coefficients of equality constraints.
q	<i>u2</i>	Nx1 vector, Lagrangian coefficients of inequality constraints.
r	<i>u3</i>	Kx1 vector, Lagrangian coefficients of lower bounds.
s	<i>u4</i>	Kx1 vector, Lagrangian coefficients of upper bounds.
t	<i>ret</i>	scalar, return code.
u		0 successful termination
v		1 max iterations exceeded
w		2 machine accuracy is insufficient to maintain decreasing function values
x		3 model matrices not conformable
y		<0 active constraints inconsistent
z		

Remarks **QProg** solves the standard quadratic programming problem:

$$\min \frac{1}{2} x' Q x - x' R$$

subject to constraints,

$$A x = B$$

$$C x \geq D$$

and bounds,

$$x_{low} \leq x \leq x_{up}$$

Source `qprog.src`

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

qqr

qqr

a		
b		
c		
d		
e		
f		
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

Purpose	Computes the orthogonal-triangular (QR) decomposition of a matrix X , such that: $X = Q_1 R$	
Format	{ ql,r } = qqr (x) ;	
Input	x	NxP matrix.
Output	ql	NxK unitary matrix, $K = \min(N,P)$.
	r	KxP upper triangular matrix.
Remarks	Given X , there is an orthogonal matrix Q such that $Q'X$ is zero below its diagonal, i.e., $Q'X = \begin{bmatrix} R \\ 0 \end{bmatrix}$ where R is upper triangular. If we partition $Q = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix}$ where Q_1 has P columns, then $X = Q_1 R$ is the QR decomposition of X . If X has linearly independent columns, R is also the Cholesky factorization of the moment matrix of X , i.e., of $X'X$. If you want only the R matrix, see the function qr . Not computing Q_1 can produce significant improvements in computing time and memory usage. An unpivoted R matrix can also be generated using cholup : $r = \text{cholup}(\text{zeros}(\text{cols}(x), \text{cols}(x)), x);$ For linear equation or least squares problems, which require Q_2 for computing residuals and residual sums of squares, see olsqr and qtyr .	

For most problems an explicit copy of Q_1 or Q_2 is not required. Instead one of the following, $Q'Y$, QY , $Q_1'Y$, Q_1Y , $Q_2'Y$, or Q_2Y , for some Y , is required. These cases are all handled by **qtyr** and **qyr**. These functions are available because Q and Q_1 are typically very large matrices while their products with Y are more manageable.

If $N < P$ the factorization assumes the form:

$$Q'X = \begin{bmatrix} R_1 & R_2 \end{bmatrix}$$

where R_1 is a $P \times P$ upper triangular matrix and R_2 is $P \times (N - P)$. Thus Q is a $P \times P$ matrix and R is a $P \times N$ matrix containing R_1 and R_2 . This type of factorization is useful for the solution of underdetermined systems. However, unless the linearly independent columns happen to be the initial rows, such an analysis also requires pivoting (see **qre** and **qrep**).

Source `qqr.src`

See also **qre**, **qrep**, **qtyr**, **qtyre**, **qtyrep**, **qyr**, **qyre**, **qyrep**, **olsqr**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

qqre

a		
b		
c		
d		
e		
f		
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		
	Purpose	Computes the orthogonal-triangular (QR) decomposition of a matrix X , such that: $X[., E] = Q_1 R$
	Format	$\{ q1, r, e \} = \text{qqre}(x);$
	Input	x NxP matrix.
	Output	$q1$ NxK unitary matrix, $K = \min(N,P)$. r KxP upper triangular matrix. e Px1 permutation vector.
	Remarks	Given $X[., E]$, where E is a permutation vector that permutes the columns of X , there is an orthogonal matrix Q such that $Q'X[., E]$ is zero below its diagonal, i.e., $Q'X[., E] = \begin{bmatrix} R \\ 0 \end{bmatrix}$ where R is upper triangular. If we partition $Q = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix}$ where Q_1 has P columns, then $X[., E] = Q_1 R$ is the QR decomposition of $X[., E]$. If you want only the R matrix, see qre . Not computing Q_1 can produce significant improvements in computing time and memory usage. If X has rank P, then the columns of X will not be permuted. If X has rank $M < P$, then the M linearly independent columns are permuted to the front of X by E . Partition the permuted X in the following way: $X[., E] = \begin{bmatrix} X_1 & X_2 \end{bmatrix}$

where X_1 is $N \times M$ and X_2 is $N \times (P - M)$. Further partition R in the following way:

$$R = \begin{bmatrix} R_{11} & R_{12} \\ 0 & 0 \end{bmatrix}$$

where R_{11} is $M \times M$ and R_{12} is $M \times (P - M)$. Then

$$A = R_{11}^{-1} R_{12}$$

and

$$X_2 = X_1 A$$

that is, A is an $M \times (P - N)$ matrix defining the linear combinations of X_2 with respect to X_1 .

If $N < P$ the factorization assumes the form:

$$Q'X = \begin{bmatrix} R_1 & R_2 \end{bmatrix}$$

where R_1 is a $P \times P$ upper triangular matrix and R_2 is $P \times (N - P)$. Thus Q is a $P \times P$ matrix and R is a $P \times N$ matrix containing R_1 and R_2 . This type of factorization is useful for the solution of underdetermined systems. For the solution of

$$X[., E]b = Y$$

it can be shown that

$$\mathbf{b} = \mathbf{qrsol}(Q'Y, R1) \mid \mathbf{zeros}(N-P, 1);$$

The explicit formation here of Q , which can be a very large matrix, can be avoided by using the function **qtyre**.

For further discussion of QR factorizations see the remarks under **qqr**.

Source `qqr.src`

See also **qqr**, **qtyre**, **olsqr**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

qqrep

qqrep

a		
b		
c		
d		
e		
f		
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		
	Purpose	Computes the orthogonal-triangular (QR) decomposition of a matrix X , such that: $X[., E] = Q_1 R$
	Format	$\{ ql, r, e \} = \text{qqrep}(x, pvt) ;$
	Input	x NxP matrix. pvt Px1 vector, controls the selection of the pivot columns: if $pvt[i] > 0$, $x[i]$ is an initial column if $pvt[i] = 0$, $x[i]$ is a free column if $pvt[i] < 0$, $x[i]$ is a final column The initial columns are placed at the beginning of the matrix and the final columns are placed at the end. Only the free columns will be moved during the decomposition.
	Output	ql NxK unitary matrix, $K = \min(N, P)$. r KxP upper triangular matrix. e Px1 permutation vector.
	Remarks	Given $X[., E]$, where E is a permutation vector that permutes the columns of X , there is an orthogonal matrix Q such that $Q'X[., E]$ is zero below its diagonal, i.e., $Q'X[., E] = \begin{bmatrix} R \\ 0 \end{bmatrix}$ where R is upper triangular. If we partition $Q = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix}$ where Q_1 has P columns, then $X[., E] = \begin{bmatrix} Q_1 & R \end{bmatrix}$ is the QR decomposition of $X[., E]$.

qqrep

qqrep allows you to control the pivoting. For example, suppose that X is a data set with a column of ones in the first column. If there are linear dependencies among the columns of X , the column of ones for the constant may get pivoted away. This column can be forced to be included among the linearly independent columns using *pvt*.

If you want only the R matrix, see **qrep**. Not computing Q_1 can produce significant improvements in computing time and memory usage.

Source `qqr.src`

See also **qqr**, **qre**, **olsqr**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

qr

qr

a		
b		
c		
d		
e		
f		
g		
h	Purpose	Computes the orthogonal-triangular (QR) decomposition of a matrix X , such that: $X = Q_1 R$
i	Format	$r = \text{qr}(x);$
j	Input	x NxP matrix.
k	Output	r KxP upper triangular matrix, $K = \min(N,P)$.
l	Remarks	<p>qr is the same as qqr but doesn't return the Q_1 matrix. If Q_1 is not wanted, qr will save a significant amount of time and memory usage, especially for large problems.</p> <p>Given X, there is an orthogonal matrix Q such that $Q'X$ is zero below its diagonal, i.e.,</p> $Q'X = \begin{bmatrix} R \\ 0 \end{bmatrix}$ <p>where R is upper triangular. If we partition</p> $Q = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix}$ <p>where Q_1 has P columns, then</p> $X = Q_1 R$ <p>is the QR decomposition of X. If X has linearly independent columns, R is also the Cholesky factorization of the moment matrix of X, i.e., of $X'X$.</p> <p>qr does not return the Q_1 matrix because in most cases it is not required and can be very large. If you need the Q_1 matrix see the function qqr. If you need the entire Q matrix call qyr with Y set to a conformable identity matrix.</p> <p>For most problems $Q'Y$, $Q_1'Y$, or $Q'Y$, $Q_1'Y$, for some Y, are required. For these cases see qtyr and qyr.</p>
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

For linear equation or least squares problems, which require Q_2 for computing residuals and residual sums of squares, see **olsqr**.

If $N < P$ the factorization assumes the form:

$$Q'X = \begin{bmatrix} R_1 & R_2 \end{bmatrix}$$

where R_1 is a $P \times P$ upper triangular matrix and R_2 is $P \times (N - P)$. Thus Q is a $P \times P$ matrix and R is a $P \times N$ matrix containing R_1 and R_2 . This type of factorization is useful for the solution of underdetermined systems. However, unless the linearly independent columns happen to be the initial rows, such an analysis also requires pivoting (see **qre** and **qrep**).

Source `qr.src`

See also **qqr**, **qrep**, **qtyre**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

gre

gre

a		
b		
c		
d		
e		
f		
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		
	Purpose	Computes the orthogonal-triangular (QR) decomposition of a matrix X , such that: $X[., E] = Q_1 R$
	Format	$\{ r, e \} = \text{gre}(x);$
	Input	x NxP matrix.
	Output	r KxP upper triangular matrix, $K = \min(N,P)$. e Px1 permutation vector.
	Remarks	<p>gre is the same as qgre but doesn't return the Q_1 matrix. If Q_1 is not wanted, gre will save a significant amount of time and memory usage, especially for large problems.</p> <p>Given $X[.,E]$, where E is a permutation vector that permutes the columns of X, there is an orthogonal matrix Q such that $Q'X[.,E]$ is zero below its diagonal, i.e.,</p> $Q'X[., E] = \begin{bmatrix} R \\ 0 \end{bmatrix}$ <p>where R is upper triangular. If we partition</p> $Q = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix}$ <p>where Q_1 has P columns, then</p> $X[., E] = Q_1 R$ <p>is the QR decomposition of $X[.,E]$.</p> <p>gre does not return the Q_1 matrix because in most cases it is not required and can be very large. If you need the Q_1 matrix see the function qgre. If you need the entire Q matrix call qyre with Y set to a conformable identity matrix. For most problems $Q'Y$, $Q_1'Y$, or QY, Q_1Y, for some Y, are required. For these cases see qtyre and qyre.</p>

If X has rank P , then the columns of X will not be permuted. If X has rank $M < P$, then the M linearly independent columns are permuted to the front of X by E . Partition the permuted X in the following way:

$$X[:, E] = \begin{bmatrix} X_1 & X_2 \end{bmatrix}$$

where X_1 is $N \times M$ and X_2 is $N \times (P - M)$. Further partition R in the following way:

$$R = \begin{bmatrix} R_{11} & R_{12} \\ 0 & 0 \end{bmatrix}$$

where R_{11} is $M \times M$ and R_{12} is $M \times (P - M)$. Then

$$A = R_{11}^{-1} R_{12}$$

and

$$X_2 = X_1 A$$

that is, A is an $M \times (P - N)$ matrix defining the linear combinations of X_2 with respect to X_1 .

If $N < P$ the factorization assumes the form:

$$Q'X = \begin{bmatrix} R_1 & R_2 \end{bmatrix}$$

where R_1 is a $P \times P$ upper triangular matrix and R_2 is $P \times (N - P)$. Thus Q is a $P \times P$ matrix and R is a $P \times N$ matrix containing R_1 and R_2 . This type of factorization is useful for the solution of underdetermined systems. For the solution of

$$X[:, E]b = Y$$

it can be shown that

$$\mathbf{b} = \mathbf{qrsol}(Q'Y, R1) \mid \mathbf{zeros}(N-P, 1);$$

The explicit formation here of Q , which can be a very large matrix, can be avoided by using the function **qtyre**.

For further discussion of QR factorizations see the remarks under **qqr**.

Source `qr.src`

See also **qqr**, **olsqr**

qrep

qrep

a		
b		
c		
d		
e		
f		
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		
	Purpose	Computes the orthogonal-triangular (QR) decomposition of a matrix X , such that: $X[., E] = Q_1 R$
	Format	$\{ r, e \} = \text{qrep}(x, pvt) ;$
	Input	x $N \times P$ matrix. pvt $P \times 1$ vector, controls the selection of the pivot columns: if $pvt[i] > 0$, $x[i]$ is an initial column if $pvt[i] = 0$, $x[i]$ is a free column if $pvt[i] < 0$, $x[i]$ is a final column The initial columns are placed at the beginning of the matrix and the final columns are placed at the end. Only the free columns will be moved during the decomposition.
	Output	r $K \times P$ upper triangular matrix, $K = \min(N, P)$. e $P \times 1$ permutation vector.
	Remarks	qrep is the same as qqrep but doesn't return the Q_1 matrix. If Q_1 is not wanted, qrep will save a significant amount of time and memory usage, especially for large problems. Given $X[., E]$, where E is a permutation vector that permutes the columns of X , there is an orthogonal matrix Q such that $Q'X[., E]$ is zero below its diagonal, i.e., $Q'X[., E] = \begin{bmatrix} R \\ 0 \end{bmatrix}$ where R is upper triangular. If we partition $Q = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix}$ where Q_1 has P columns, then $X[., E] = Q_1 R$

is the QR decomposition of $X[.,E]$.

qrep does not return the Q_1 matrix because in most cases it is not required and can be very large. If you need the Q_1 matrix see the function **qqrep**. If you need the entire Q matrix call **qyrep** with Y set to a conformable identity matrix. For most problems $Q'Y$, $Q_1'Y$, or QY , Q_1Y , for some Y , are required. For these cases see **qtyrep** and **qyrep**.

qrep allows you to control the pivoting. For example, suppose that X is a data set with a column of ones in the first column. If there are linear dependencies among the columns of X , the column of ones for the constant may get pivoted away. This column can be forced to be included among the linearly independent columns using *pvt*.

Source `qr.src`

See also **qr**, **qre**, **qqrep**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

qrsol

qrsol

a	Purpose	Computes the solution of $Rx = b$ where R is an upper triangular matrix.	
b	Format	$x = \text{qrsol}(b,R);$	
c	Input	b	PxL matrix.
d		R	PxP upper triangular matrix.
e	Output	x	PxL matrix.
f	Remarks	qrsol applies a backsolve to $Rx = b$ to solve for x . Generally R will be the R matrix from a QR factorization. qrsol may be used, however, in any situation where R is upper triangular.	
g	Source	qrsol.src	
h	See also	qqr, qr, qtyr, qrtsol	
i			
j			
k			
l			
m			
n			
o			
p			
q			
r			
s			
t			
u			
v			
w			
x y z			

qrtsol

Purpose	Computes the solution of $R'x = b$ where R is an upper triangular matrix.	
Format	$x = \text{qrtsol}(b, R);$	
Input	b	PxL matrix.
	R	PxP upper triangular matrix.
Output	x	PxL matrix.
Remarks	<p>qrtsol applies a forward solve to $R'x = b$ to solve for x. Generally R will be the R matrix from a QR factorization. qrtsol may be used, however, in any situation where R is upper triangular. If R is lower triangular, transpose before calling qrtsol.</p> <p>If R is not transposed, use qrsol.</p>	
Source	qrsol.src	
See also	qqr , qr , qtyr , qrsol	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

qtyr

qtyr

a		
b		
c		
d		
e		
f		
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

Purpose	Computes the orthogonal-triangular (QR) decomposition of a matrix X and returns $Q'Y$ and R .	
Format	{ qty,r } = qtyr (y,x) ;	
Input	y	NxL matrix.
	x	NxP matrix.
Output	qty	NxL unitary matrix.
	r	KxP upper triangular matrix, $K = \min(N,P)$.
Remarks	Given X , there is an orthogonal matrix Q such that $Q'X$ is zero below its diagonal, i.e., $Q'X = \begin{bmatrix} R \\ 0 \end{bmatrix}$ where R is upper triangular. If we partition $Q = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix}$ where Q_1 has P columns, then $X = Q_1 R$ is the QR decomposition of X . If X has linearly independent columns, R is also the Cholesky factorization of the moment matrix of X , i.e., of $X'X$. For most problems Q or Q_1 is not what is required. Rather, we require $Q'Y$ or $Q_1'Y$ where Y is an NxL matrix (if either QY or Q_1Y are required, see qyr). Since Q can be a very large matrix, qtyr has been provided for the calculation of $Q'Y$ which will be a much smaller matrix. $Q_1'Y$ will be a submatrix of $Q'Y$. In particular, $G = Q_1'Y = qty[1 : P, .]$ and $Q_2'Y$ is the remaining submatrix:	

$$H = Q_2' Y = qty[P + 1 : N, .]$$

Suppose that X is an $N \times K$ data set of independent variables, and v is an $N \times 1$ vector of dependent variables. Then it can be shown that

$$b = R^{-1} G$$

and

$$s_j = \sum_{i=1}^{N-P} H_{i,j}, j = 1, 2, \dots, L$$

where b is a $P \times L$ matrix of least squares coefficients and s is a $1 \times L$ vector of residual sums of squares. Rather than invert R directly, however, it is better to apply **qrsol** to

$$Rb = Q_1' Y$$

For rank deficient least squares problems, see **qtyre** and **qtyrep**.

Example The QR algorithm is the superior numerical method for the solution of least squares problems:

```
loadm x, y;
{ qty, r } = qtyr(y,x);
qlty = qty[1:rows(r),.];
q2ty = qty[rows(r)+1:rows(qty),.];
b = qrsol(qlty,r); /* LS coefficients */
s2 = sumc(q2ty^2); /* residual sums of squares */
```

Source qtyr.src

See also **qqr**, **qtyre**, **qtyrep**, **olsqr**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

qtyre

qtyre

Purpose	Computes the orthogonal-triangular (QR) decomposition of a matrix X and returns $Q'Y$ and R .		
Format	{ qty,r,e } = qtyre (y,x);		
Input	y	NxL matrix.	
	x	NxP matrix.	
Output	qty	NxL unitary matrix.	
	r	KxP upper triangular matrix, $K = \min(N,P)$.	
	e	Px1 permutation vector.	
Remarks	Given $X[.,E]$, where E is a permutation vector that permutes the columns of X , there is an orthogonal matrix Q such that $Q'X[.,E]$ is zero below its diagonal, i.e., $Q'X[., E]= \begin{bmatrix} R \\ 0 \end{bmatrix}$ where R is upper triangular. If we partition $Q = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix}$ where Q_1 has P columns, then $X[., E]= Q_1 R$ is the QR decomposition of $X[.,E]$. If X has rank P, then the columns of X will not be permuted. If X has rank $M < P$, then the M linearly independent columns are permuted to the front of X by E . Partition the permuted X in the following way: $X[., E]= \begin{bmatrix} X_1 & X_2 \end{bmatrix}$ where X_1 is NxM and X_2 is Nx(P – M). Further partition R in the following way:		

$$R = \begin{bmatrix} R_{11} & R_{12} \\ 0 & 0 \end{bmatrix}$$

where R_{11} is $M \times M$ and R_{12} is $M \times (P - M)$. Then

$$A = R_{11}^{-1} R_{12}$$

and

$$X_2 = X_1 A$$

that is, A is an $M \times (P - N)$ matrix defining the linear combinations of X_2 with respect to X_1 .

For most problems Q or Q_1 is not what is required. Rather, we require $Q'Y$ or $Q_i'Y$ where Y is an $N \times L$ matrix. Since Q can be a very large matrix, **qtyre** has been provided for the calculation of $Q'Y$ which will be a much smaller matrix. $Q_i'Y$ will be a submatrix of $Q'Y$. In particular,

$$Q_1'Y = qty[1:P, .]$$

and $Q_2'Y$ is the remaining submatrix:

$$Q_2'Y = qty[P + 1 : N, .]$$

Suppose that X is an $N \times K$ data set of independent variables and Y is an $N \times 1$ vector of dependent variables. Suppose further that X contains linearly dependent columns, i.e., X has rank $M < P$. Then define

$$C = Q_2'Y [1 : M, .]$$

$$A = R[1:M, 1:M]$$

and the vector (or matrix of $L > 1$) of least squares coefficients of the reduced, linearly independent problem is the solution of

$$Ab = C$$

To solve for b use **qrsol**:

$$b = qrsol(C, A);$$

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

qtyre

If $N < P$ the factorization assumes the form:

$$Q'X[:, E] = \begin{bmatrix} R_1 & R_2 \end{bmatrix}$$

where R_1 is a $P \times P$ upper triangular matrix and R_2 is $P \times (N - P)$. Thus Q is a $P \times P$ matrix and R is a $P \times N$ matrix containing R_1 and R_2 . This type of factorization is useful for the solution of underdetermined systems. For the solution of

$$X[:, E]b = Y$$

it can be shown that

$$\mathbf{b} = \text{qrsol}(Q'Y, R1) \mid \text{zeros}(N-P, 1);$$

Source `qtyr.src`

See also `qqr`, `qre`, `qtyr`

qtyrep

Purpose	Computes the orthogonal-triangular (QR) decomposition of a matrix X using a pivot vector and returns $Q'Y$ and R .	
Format	{ qty,r,e } = qtyrep (y,x,pvt) ;	
Input	y	NxL matrix.
	x	NxP matrix.
	pvt	Px1 vector, controls the selection of the pivot columns: if $pvt[i] > 0$, $x[i]$ is an initial column if $pvt[i] = 0$, $x[i]$ is a free column if $pvt[i] < 0$, $x[i]$ is a final column The initial columns are placed at the beginning of the matrix and the final columns are placed at the end. Only the free columns will be moved during the decomposition.
Output	qty	NxL unitary matrix.
	r	KxP upper triangular matrix, $K = \min(N,P)$.
	e	Px1 permutation vector.
Remarks	Given $X[.,E]$, where E is a permutation vector that permutes the columns of X , there is an orthogonal matrix Q such that $Q'X[.,E]$ is zero below its diagonal, i.e.,	

$$Q'X[., E]= \begin{bmatrix} R \\ 0 \end{bmatrix}$$

where R is upper triangular. If we partition

$$Q = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix}$$

where Q_1 has P columns, then

$$X[., E]= Q_1R$$

is the QR decomposition of $X[.,E]$.

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

qtyrep

qtyrep allows you to control the pivoting. For example, suppose that *X* is a data set with a column of ones in the first column. If there are linear dependencies among the columns of *X*, the column of ones for the constant may get pivoted away. This column can be forced to be included among the linearly independent columns using *pvt*.

```
Example  y = { 4 7 2,
              5 9 1,
              6 3 3 };
          x = { 12 9 5,
              4 3 5,
              4 2 7 };
          pvt = { 11, 10, 3 };
          { qty, r, e } = qtyrep(y,x,pvt);

          -6.9347609      -9.9498744  -3.0151134
          qty =  4.0998891  3.5527137e - 15   2.1929640
                3.4785054      6.3245553  0.31622777

          -13.266499   -9.6483630  -8.1408063
          r =  0.0000000  -0.95346259   4.7673129
                0.0000000   0.0000000   3.1622777

          1.0000000
          e =  2.0000000
                3.0000000
```

Source qtyr.src

See also **qrep**, **qtyre**

quantile

Purpose Computes quantiles from data in a matrix, given specified probabilities.

Format `y = quantile(x,e)`

Input

<code>x</code>	NxK matrix of data.
<code>e</code>	Lx1 vector, quantile levels or probabilities.

Output `y` LxK matrix, quantiles.

Remarks **quantile** will not succeed if `N*minc(e)` is less than 1, or `N*maxc(e)` is greater than `N - 1`. In other words, to produce a quantile for a level of .001, the input matrix must have more than 1000 rows.

Example

```

rndseed 345567;

x = rndn(1000,4); /* data */
e = { .025, .5, .975 }; /* quantile levels */
y = quantile(x,e);

```

```

print "medians";
print y[2,.];
print;
print "95 percentiles";
print y[1,.];
print y[3,.];

```

produces:

medians

```

-0.0020      -0.0408      -0.0380      -0.0247

```

95 percentiles

```

-1.8677      -1.9894      -2.1474      -1.8747

```

quantile

1.9687 2.0899 1.8576 2.0545

Source quantile.src

quantiled

Purpose Computes quantiles from data in a data set, given specified probabilities.

Format `y = quantiled(dataset,e,var);`

Input

<i>dataset</i>	string, data set name, or NxM matrix of data.
<i>e</i>	Lx1 vector, quantile levels or probabilities.
<i>var</i>	Kx1 vector or scalar zero. If Kx1, character vector of labels selected for analysis, or numeric vector of column numbers in data set of variables selected for analysis. If scalar zero, all columns are selected.

If *dataset* is a matrix *var* cannot be a character vector.

Output *y* LxK matrix, quantiles.

Remarks **quantiled** will not succeed if $N * \text{minc}(e)$ is less than 1, or $N * \text{maxc}(e)$ is greater than $N - 1$. In other words, to produce a quantile for a level of .001, the input matrix must have more than 1000 rows.

Example `y = quantiled("tobit",e,0);`

```
print "medians";
print y[2,.];
print;
print "95 percentiles";
print y[1,.];
print y[3,.];

produces:
medians
0.0000      1.0000      -0.0021      -0.1228
95 percentiles
-1.1198      1.0000      -1.8139      -2.3143
```

quantiled

2.3066 1.0000 1.4590 1.6954

Source quantile.src

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

qyr

Purpose Computes the orthogonal-triangular (QR) decomposition of a matrix X and returns QY and R .

Format $\{ qy, r \} = \mathbf{qyr}(y, x);$

Input y $N \times L$ matrix.
 x $N \times P$ matrix.

Output qy $N \times L$ unitary matrix.
 r $K \times P$ upper triangular matrix, $K = \min(N, P)$.

Remarks Given X , there is an orthogonal matrix Q such that $Q'X$ is zero below its diagonal, i.e.,

$$Q'X = \begin{bmatrix} R \\ 0 \end{bmatrix}$$

where R is upper triangular. If we partition

$$Q = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix}$$

where Q_1 has P columns, then

$$X = Q_1 R$$

is the QR decomposition of X . If X has linearly independent columns, R is also the Cholesky factorization of the moment matrix of X , i.e., of $X'X$.

For most problems Q or Q_1 is not what is required. Since Q can be a very large matrix, **qyr** has been provided for the calculation of QY , where Y is some $N \times L$ matrix, which will be a much smaller matrix.

If either $Q'Y$ or $Q_1'Y$ are required, see **qtyr**.

Example $x = \{ 1 \ 11, \ 7 \ 3, \ 2 \ 1 \};$
 $y = \{ 2 \ 6, \ 5 \ 10, \ 4 \ 3 \};$
 $\{ qy, \ r \} = \mathbf{qyr}(y, x);$

qyr

$$\begin{matrix} & 4.6288991 & 9.0506281 \\ qy = & -3.6692823 & -7.8788202 \\ & 3.1795692 & 1.0051489 \end{matrix}$$

$$\begin{matrix} r = & -7.3484692 & -4.6268140 \\ & 0.0000000 & 10.468648 \end{matrix}$$

Source `qyr.src`

See also **`qqr, qyre, qyrep, olsqr`**

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

qyre

Purpose Computes the orthogonal-triangular (QR) decomposition of a matrix X and returns QY and R .

Format $\{ qy, r, e \} = \text{qyre}(y, x);$

Input

y	$N \times L$ matrix.
x	$N \times P$ matrix.

Output

qy	$N \times L$ unitary matrix.
r	$K \times P$ upper triangular matrix, $K = \min(N, P)$.
e	$P \times 1$ permutation vector.

Remarks Given $X[:, E]$, where E is a permutation vector that permutes the columns of X , there is an orthogonal matrix Q such that $Q'X[:, E]$ is zero below its diagonal, i.e.,

$$Q'X[:, E] = \begin{bmatrix} R \\ 0 \end{bmatrix}$$

where R is upper triangular. If we partition

$$Q = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix}$$

where Q_1 has P columns, then

$$X[:, E] = Q_1 R$$

is the QR decomposition of $X[:, E]$.

For most problems Q or Q_1 is not what is required. Since Q can be a very large matrix, **qyre** has been provided for the calculation of QY , where Y is some $N \times L$ matrix, which will be a much smaller matrix.

If either $Q'Y$ or $Q_1'Y$ are required, see **qtyre**.

If $N < P$ the factorization assumes the form:

$$Q'X[:, E] = \begin{bmatrix} R_1 & R_2 \end{bmatrix}$$

qyre

where R_1 is a $P \times P$ upper triangular matrix and R_2 is $P \times (N - P)$. Thus Q is a $P \times P$ matrix and R is a $P \times N$ matrix containing R_1 and R_2 .

Example $x = \{ 1 \ 11, \ 7 \ 3, \ 2 \ 1 \};$
 $y = \{ 2 \ 6, \ 5 \ 10, \ 4 \ 3 \};$
 $\{ qy, \ r, \ e \} = qyre(y,x);$

$qy = \begin{matrix} -0.5942276 & -3.0456088 \\ -6.2442636 & -11.647846 \\ 2.3782485 & -0.22790230 \end{matrix}$

$r = \begin{matrix} -11.445523 & -2.9705938 \\ 0.0000000 & -6.7212776 \end{matrix}$

$e = \begin{matrix} 2.0000000 \\ 1.0000000 \end{matrix}$

Source `qyr.src`

See also `qqr, qre, qyr`

qyrep

Purpose	Computes the orthogonal-triangular (QR) decomposition of a matrix X using a pivot vector and returns QY and R .	
Format	{ qy,r,e } = qyrep (y,x,pvt) ;	
Input	y	$N \times L$ matrix.
	x	$N \times P$ matrix.
	pvt	$P \times 1$ vector, controls the selection of the pivot columns: if $pvt[i] > 0$, $x[i]$ is an initial column if $pvt[i] = 0$, $x[i]$ is a free column if $pvt[i] < 0$, $x[i]$ is a final column The initial columns are placed at the beginning of the matrix and the final columns are placed at the end. Only the free columns will be moved during the decomposition.
Output	qy	$N \times L$ unitary matrix.
	r	$K \times P$ upper triangular matrix, $K = \min(N,P)$.
	e	$P \times 1$ permutation vector.
Remarks	Given $X[.,E]$, where E is a permutation vector that permutes the columns of X , there is an orthogonal matrix Q such that $Q'X[.,E]$ is zero below its diagonal, i.e.,	

$$Q'X[., E] = \begin{bmatrix} R \\ 0 \end{bmatrix}$$

where R is upper triangular. If we partition

$$Q = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix}$$

where Q_1 has P columns, then

$$X[., E] = Q_1 R$$

is the QR decomposition of $X[.,E]$.

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

qyrep

qyrep allows you to control the pivoting. For example, suppose that X is a data set with a column of ones in the first column. If there are linear dependencies among the columns of X , the column of ones for the constant may get pivoted away. This column can be forced to be included among the linearly independent columns using *pvt*.

For most problems Q or Q_I is not what is required. Since Q can be a very large matrix, **qyrep** has been provided for the calculation of QY , where Y is some $N \times L$ matrix, which will be a much smaller matrix.

If either $Q'Y$ or $Q_I Y$ are required, see **qtyrep**.

If $N < P$ the factorization assumes the form:

$$Q'X[:, E] = \begin{bmatrix} R_1 & R_2 \end{bmatrix}$$

where R_1 is a $P \times P$ upper triangular matrix and R_2 is $P \times (N - P)$. Thus Q is a $P \times P$ matrix and R is a $P \times N$ matrix containing R_1 and R_2 .

Source `qyr.src`

See also **qr, qqrep, qrep, qtyrep**

rank

Purpose	Computes the rank of a matrix, using the singular value decomposition.	
Format	$k = \mathbf{rank}(x);$	
Input	x	NXP matrix.
Global Input	<code>_svd_tol</code>	global scalar, the tolerance used in determining if any of the singular values are effectively 0. The default value is $10e-13$. This can be changed before calling the procedure.
Output	k	an estimate of the rank of x . This equals the number of singular values of x that exceed a prespecified tolerance in absolute value.
Global Output	<code>_svd_err</code>	global scalar, if not all of the singular values can be computed <code>_svd_err</code> will be nonzero.
Source	<code>svd.src</code>	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

rankindx

rankindx

a		
b	Purpose	Returns the vector of ranks of a vector.
c	Format	$y = \text{rankindx}(x, flag);$
d	Input	x Nx1 vector.
e		$flag$ scalar, 1 for numeric data or 0 for character data.
f	Output	y Nx1 vector containing the ranks of x . That is, the rank of the largest element is N and the rank of the smallest is 1. (To get ranks in descending order, subtract y from $N+1$).
g		
h	Remarks	rankindx assigns different ranks to elements that have equal values (ties). Missing values are assigned the lowest ranks.
i		
j	Example	$\text{let } x = 12 \ 4 \ 15 \ 7 \ 8;$
k		$r = \text{rankindx}(x, 1);$
l		4
m		1
n		$r = 5$
o		2
p		3
q		
r		
s		
t		
u		
v		
w		
x y z		

readr

Purpose	Reads a specified number of rows of data from a GAUSS data set (.dat) file or a GAUSS matrix (.fmt) file.	
Format	<code>y = readr(fl,r);</code>	
Input	<i>fl</i>	scalar, file handle of an open file.
	<i>r</i>	scalar, number of rows to read.
Output	<i>y</i>	NxK matrix, the data read from the file.
Remarks	The first time a readr statement is encountered, the first <i>r</i> rows will be read. The next time it is encountered, the next <i>r</i> rows will be read in, and so on. If the end of the data set is reached before <i>r</i> rows can be read, then only those rows remaining will be read.	
	After the last row has been read, the pointer is placed immediately after the end of the file. An attempt to read the file in these circumstances will cause an error message.	
	To move the pointer to a specific place in the file use seekr .	
Example	<pre>open dt = dat1.dat; m = 0; do until eof(dt); x = readr(dt,400); m = m+moment(x,0); endo; dt = close(dt);</pre>	
	This code reads data from a data set 400 rows at a time. The moment matrix for each set of rows is computed and added to the sum of the previous moment matrices. The result is the moment matrix for the entire data set. eof(dt) returns 1 when the end of the data set is encountered.	
See also	open, create, writer, seekr, eof	

real

real

a	Purpose	Returns the real part of a matrix.
b	Format	$zr = \mathbf{real}(x);$
c	Input	x NxK matrix.
d	Output	zr NxK matrix, the real part of x .
e	Remarks	If x is not complex, zr will be equal to x .
f	Example	$x = \begin{Bmatrix} 1 & 11, \\ 7i & 3, \\ 2+i & 1 \end{Bmatrix};$ $zr = \mathbf{real}(x);$ $zr = \begin{bmatrix} 1.0000000 & 11.0000000 \\ 0.0000000 & 3.0000000 \\ 2.0000000 & 1.0000000 \end{bmatrix}$
g	See also	complex, imag

recode

Purpose	Changes the values of an existing vector from a vector of new values. Used in data transformations.		
Format	$y = \text{recode}(x, e, v);$		
Input	x	Nx1 vector to be recoded (changed).	
	e	NxK matrix of 1's and 0's.	
	v	Kx1 vector containing the new values to be assigned to the recoded variable.	
Output	y	Nx1 vector containing the recoded values of x .	
Remarks	There should be no more than a single 1 in any row of e .		
	For any given row N of x and e , if the K^{th} column of e is 1, the K^{th} element of v will replace the original element of x .		
	If every column of e contains a 0, the original value of x will be unchanged.		
Example	<pre>x = { 20, 45, 32, 63, 29 }; e1 = (20 .lt x) .and (x .le 30); e2 = (30 .lt x) .and (x .le 40); e3 = (40 .lt x) .and (x .le 50); e4 = (50 .lt x) .and (x .le 60); e = e1~e2~e3~e4;</pre>		

recode

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

```
v = { 1,
      2,
      3,
      4 };

y = recode(x,e,v);

      20
      45
x = 32
      63
      29

      0 0 0 0
      0 0 1 0
e = 0 1 0 0
      0 0 0 0
      1 0 0 0

      1
v = 2
      3
      4

      20
      3
y = 2
      63
      1
```

Source datatran.src

See also **code, substute**

recode (dataloop)

Purpose	Changes the value of a variable with different values based on a set of logical expressions.
Format	recode [[#]] [\$] <i>var</i> with <i>val_1</i> for <i>expression_1</i> , <i>val_2</i> for <i>expression_2</i> , . . . <i>val_n</i> for <i>expression_n</i> ;
Input	<i>var</i> literal, the new variable name. <i>val</i> scalar, value to be used if corresponding expression is true. <i>expression</i> logical scalar-returning expression that returns nonzero <i>TRUE</i> or zero <i>FALSE</i> .
Remarks	<p>If '\$' is specified, the variable will be considered a character variable. If '#' is specified, the variable will be considered numeric. If neither is specified, the type of the variable will be left unchanged.</p> <p>The logical expressions must be mutually exclusive, that is only one may return <i>TRUE</i> for a given row (observation).</p> <p>If none of the expressions is <i>TRUE</i> for a given row (observation), its value will remain unchanged.</p> <p>Any variables referenced must already exist, either as elements of the source data set, as externs, or as the result of a previous make, vector, or code statement.</p>

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

recode (dataloop)

Example recode age with

```
1 for age < 21,  
2 for age ≥ 21 and age < 35,  
3 for age ≥ 35 and age < 50,  
4 for age ≥ 50 and age < 65,  
5 for age ≥ 65;
```

recode \$ sex with

```
"MALE" for sex == 1,  
"FEMALE" for sex == 0;
```

recode # sex with

```
1 for sex $== "MALE",  
0 for sex $== "FEMALE";
```

See also **code**

recserar

Purpose Computes a vector of autoregressive recursive series.

Format `y = recserar(x,y0,a);`

Input

<i>x</i>	NxK matrix
<i>y0</i>	PxK matrix.
<i>a</i>	PxK matrix.

Output *y* NxK matrix containing the series.

Remarks **recserar** is particularly useful in dealing with time series. Typically, the result would be thought of as *K* vectors of length *N*. *y0* contains the first *P* values of each of these vectors (thus, these are prespecified). The remaining elements are constructed by computing a *P*th order “autoregressive” recursion, with weights given by *a*, and then by adding the result to the corresponding elements of *x*. That is, the *t*th row of *y* is given by:

$$y[t, :] = x[t, :] + a[1, :] * y[t - 1, :] + \dots + a[P, :] * y[t - P, :], t = P + 1, \dots, N$$

and

$$y[t, :] = y0[t, :], t = 1, \dots, P$$

Note that the first *P* rows of *x* are not used.

Example

```
n = 10;
fn multnorm(n,sigma) =
    rndn(n,rows(sigma))*chol(sigma);
let sig[2,2] = { 1 -.3, -.3 1 };
rho = 0.5~0.3;
y0 = 0~0;
e = multnorm(n,sig);
x = ones(n,1)~rndn(n,3);
b = 1|2|3|4;
```

recserar

```
y = recserar(x*b+e,y0,rho);
```

In this example, two autoregressive series are formed using simulated data. The general form of the series can be written:

$$y[1,t] = \text{rho}[1,1]*y[1,t-1] + x[t,]*b + e[1,t]$$
$$y[2,t] = \text{rho}[2,1]*y[2,t-1] + x[t,]*b + e[2,t]$$

The error terms (**e[1,t]** and **e[2,t]**) are not individually serially correlated, but they are contemporaneously correlated with each other. The variance-covariance matrix is **sig**.

See also **recsercp**, **recserrc**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

recsercp

Purpose Computes a recursive series involving products. Can be used to compute cumulative products, to evaluate polynomials using Horner's rule, and to convert from base b representations of numbers to decimal representations among other things.

Format `y = recsercp(x,z);`

Input

x	$N \times K$ or $1 \times K$ matrix
z	$N \times K$ or $1 \times K$ matrix.

Output

y	$N \times K$ matrix in which each column is a series generated by a recursion of the form:
	$y(1) = x(1) + z(1)$
	$y(t) = y(t-1) \times x(t) + z(t), t = 2, \dots, N$

Remarks The following GAUSS code could be used to emulate **recsercp** when the number of rows in x and z is the same:

```
n = rows(x); /* assume here that rows(z) */
               /* is also n */

y = zeros(n,1);
y[1,.] = x[1,.] + z[1,.];
i = 2;

do until i > n;
    y[i,.] = y[i-1,.] .* x[i,.] + z[i,.];
    i = i + 1;
endo;
```

Note that K series can be computed simultaneously, since x and z can have K columns (they must both have the same number of columns).

recsercp allows either x or z to have only 1 row.

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

recsercp

recsercp($x,0$) will produce the cumulative products of the elements in x .

Example

```
c1 = c[1,.];  
n = rows(c) - 1;  
y = recsercp(x,trim(c ./ c1,1,0));  
p = c1 .* y[n,.];
```

If \mathbf{x} is a scalar and \mathbf{c} is an $(N+1) \times 1$ vector, the result \mathbf{p} will contain the value of the polynomial whose coefficients are given in \mathbf{c} . That is:

$$p = c[1,.] \times x^n + c[2,.] \times x^{(n-1)} + \dots + c[n+1,.]$$

Note that both \mathbf{x} and \mathbf{c} could contain more than 1 column, and then this code would evaluate the entire set of polynomials at the same time. Note also that if $\mathbf{x} = 2$, and if \mathbf{c} contains the digits of the binary representation of a number, then \mathbf{p} will be the decimal representation of that number.

See also **recserar**, **recserrc**

recserrc

Purpose	Computes a recursive series involving division.		
Format	$y = \text{recserrc}(x,z);$		
Input	x	1xK or Kx1 vector.	
	z	NxK matrix.	
Output	y	NxK matrix in which each column is a series generated by a recursion of the form:	
		$y[1] = x \bmod z[1], \ x = \text{trunc}(x / z[1])$	
		$y[2] = x \bmod z[2], \ x = \text{trunc}(x / z[2])$	
		$y[3] = x \bmod z[3], \ x = \text{trunc}(x / z[3])$	
		...	
		$y[n] = x \bmod z[n]$	
Remarks	Can be used to convert from decimal to other number systems (radix conversion).		
Example	$x = 2 \mid 8 \mid 10;$		
	$b = 2;$		
	$n = \text{maxc}(\log(x) ./ \log(b)) + 1;$		
	$z = \text{reshape}(b, n, \text{rows}(x));$		
	$y = \text{rev}(\text{recserrc}(x, z))' ;$		
	The result, y , will contain in its rows (note that it is transposed in the last step) the digits representing the decimal numbers 2, 8, and 10 in base 2:		
	<pre>0 0 1 0 1 0 0 0 1 0 1 0</pre>		
Source	recserrc.src		
See also	recserar, recsercp		

rerun

rerun

	Purpose	Displays the most recently created graphics file.
	Library	pgraph
	Format	rerun;
	Portability	DOS only rerun invokes the graphics utility <code>pggrun.exe</code> .
	Remarks	rerun is used by the endwind function.
	Source	<code>pcart.src</code>
	Globals	<code>_pcmdlin</code> , <code>_pnotify</code> , <code>_psilent</code> , <code>_ptek</code> , <code>_pzoom</code>

reshape

Purpose Reshapes a matrix.

Format `y = reshape(x,r,c);`

Input

<i>x</i>	NxK matrix.
<i>r</i>	scalar, new row dimension.
<i>c</i>	scalar, new column dimension.

Output *y* RxC matrix created from the elements of *x*.

Remarks Matrices are stored in row major order.

The first *c* elements are put into the first row of *y*, the second in the second row, and so on. If there are more elements in *x* than in *y*, the remaining elements are discarded. If there are not enough elements in *x* to fill *y*, then when **reshape** runs out of elements, it goes back to the first element of *x* and starts getting additional elements from there.

Example `y = reshape(x,2,6);`

If $x = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix}$ then $y = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 7 & 8 & 9 & 10 & 11 & 12 \end{bmatrix}$

If $x = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$ then $y = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 7 & 8 & 9 & 1 & 2 & 3 \end{bmatrix}$

If $x = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \end{bmatrix}$ then $y = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 7 & 8 & 9 & 10 & 11 & 12 \end{bmatrix}$

If $x = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ then $y = \begin{bmatrix} 1 & 2 & 3 & 4 & 1 & 2 \\ 3 & 4 & 1 & 2 & 3 & 4 \end{bmatrix}$

If $x = 1$ then $y = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$

reshape

See also **submat**, **vec**

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

retp

Purpose	Returns from a procedure or keyword.
Format	retp; retp(x,y,...);
Remarks	<p>For more details, see “Procedures and Keywords” in the <i>User’s Guide</i>.</p> <p>In a retp statement 0-1023 items may be returned. The items may be expressions. Items are separated by commas.</p> <p>It is legal to return with no arguments, as long as the procedure is defined to return 0 arguments.</p>
See also	proc, keyword, endp

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

return

return

a	Purpose	Returns from a subroutine.
b	Format	return; return(x,y,...) ;
c	Remarks	The number of items that may be returned from a subroutine in a return statement is limited only by stack space. The items may be expressions. Items are separated by commas. It is legal to return with no arguments and therefore return nothing.
d	See also	gosub, pop
e		
f		
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

rev

Purpose	Reverses the order of the rows in a matrix.
Format	$y = \text{rev}(x);$
Input	x NxK matrix.
Output	y NxK matrix containing the reversed rows of x .
Remarks	The first row of y will be where the last row of x was and the last row will be where the first was and so on. This can be used to put a sorted matrix in descending order.
Example	$x = \text{round}(\text{rndn}(5,3)*10);$ $y = \text{rev}(x);$ <div> <div> 10 7 8 7 4 -9 </div> <div> $x =$ -11 0 -3 3 18 0 9 -1 20 9 -1 20 3 18 0 </div> <div> $y =$ -11 0 -3 7 4 -9 10 7 8 </div> </div>
See also	sortc

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

rfft

rfft

a		
b	Purpose	Computes a real 1- or 2-D Fast Fourier transform.
c	Format	<code>y = rfft(x);</code>
d	Input	<code>x</code> NxK real matrix.
e	Output	<code>y</code> LxM matrix, where L and M are the smallest powers of 2 greater than or equal to N and K, respectively.
f		
g	Remarks	Computes the RFFT of <code>x</code> , scaled by $1/(L*M)$. This uses a Temperton Fast Fourier algorithm. If N or K is not a power of 2, <code>x</code> will be padded out with zeros before computing the transform.
h		
i	Example	<code>x = { 6 9, 8 1 };</code> <code>y = rfft(x);</code> <code>y = 6.0000000 1.0000000</code> <code> 1.5000000 -2.5000000</code>
j		
k		
l		
m		
n	See also	<code>rfffti, fft, ffti, fftm, fftmi</code>
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

rfffti

Purpose	Computes inverse real 1- or 2-D Fast Fourier transform.	
Format	$y = \text{rfffti}(x);$	
Input	x	NxK matrix.
Output	y	LxM real matrix, where L and M are the smallest prime factor products greater than or equal to N and K.
Remarks	It is up to the user to guarantee that the input will return a real result. If in doubt, use fffti .	
Example	<pre>x = { 6 1, 1.5 -2.5 }; y = rfffti(x); y = 6.0000000 9.0000000 8.0000000 1.0000000</pre>	
See also	rfft, fft, ffti, fftm, fftmi	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

rffftip

rffftip

a	Purpose	Computes an inverse real 1- or 2-D FFT. Takes a packed format FFT as input.
b		
c	Format	<code>y = rffftip(x);</code>
d	Input	<code>x</code> NxK matrix or K-length vector.
e		
f	Output	<code>y</code> LxM real matrix or M-length vector.
g	Remarks	rffftip assumes that its input is of the same form as that output by rffft and rffftnp .
h		rffftip uses the Temperton prime factor FFT algorithm. This algorithm can compute the inverse FFT of any vector or matrix whose dimensions can be expressed as the product of selected prime number factors.
i		GAUSS implements the Temperton algorithm for any integer power of 2, 3, and 5, and one factor of 7. Thus, rffftip can handle any matrix whose dimensions can be expressed as:
j		$2^p \times 3^q \times 5^r \times 7^s, \quad \begin{matrix} p, q, r \geq 0 \\ s = 0 \text{ or } 1 \end{matrix}$
k		If a dimension of <code>x</code> does not meet this requirement, it will be padded with zeros to the next allowable size before the inverse FFT is computed. Note that rffftip assumes the length (for vectors) or column dimension (for matrices) of <code>x</code> is K-1 rather than K, since the last element or column does not hold FFT information, but the Nyquist frequencies.
l		The sizes of <code>x</code> and <code>y</code> are related as follows: L will be the smallest prime factor product greater than or equal to N, and M will be twice the smallest prime factor product greater than or equal to K-1. This takes into account the fact that <code>x</code> contains both positive and negative frequencies in the row dimension (matrices only), but only positive frequencies, and those only in the first K-1 elements or columns, in the length or column dimension.
m		It is up to the user to guarantee that the input will return a real result. If in doubt, use fffti . Note, however, that fffti expects a full FFT, including negative frequency information, for input.
n		Do not pass rffftip the output from rffft or rffftn — it will return incorrect results. Use rfffti with those routines.
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

rffftip

See also `fft`, `ffti`, `fftm`, `fftmi`, `fftn`, `rfft`, `rfffti`, `rfftn`,
`rffftnp`, `rfftp`

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

rfftn

rfftn

a	Purpose	Computes a real 1- or 2-D FFT.
b	Format	$y = \text{rfftn}(x);$
c	Input	x NxK real matrix.
d	Output	y LxM matrix, where L and M are the smallest prime factor products greater than or equal to N and K, respectively.
e	Remarks	rfftn uses the Temperton prime factor FFT algorithm. This algorithm can compute the FFT of any vector or matrix whose dimensions can be expressed as the product of selected prime number factors. GAUSS implements the Temperton algorithm for any power of 2, 3, and 5, and one factor of 7. Thus, rfftn can handle any matrix whose dimensions can be expressed as: $2^p \times 3^q \times 5^r \times 7^s,$ $\begin{array}{ll} p, q, r \geq 0 & \text{for rows of matrix} \\ p > 0, q, r \geq 0 & \text{for columns of matrix} \\ p > 0, q, r \geq 0 & \text{for length of vector} \\ s = 0 \text{ or } 1 & \text{for all dimensions} \end{array}$ If a dimension of x does not meet these requirements, it will be padded with zeros to the next allowable size before the FFT is computed. rfftn pads matrices to the next allowable size; however, it generally runs faster for matrices whose dimensions are highly composite numbers, i.e., products of several factors (to various powers), rather than powers of a single factor. For example, even though it is bigger, a 33600x1 vector can compute as much as 20 percent faster than a 32768x1 vector, because 33600 is a highly composite number, $2^6 \times 3 \times 5^2 \times 7$, whereas 32768 is a simple power of 2, 2^{15} . For this reason, you may want to hand-pad matrices to optimum dimensions before passing them to rfftn . The Run-Time Library includes two routines, optn and optnevn , for determining optimum dimensions. Use optn to determine optimum rows for matrices, and optnevn to determine optimum columns for matrices and optimum lengths for vectors. The Run-Time Library also includes the nextn and nextnevn routines, for determining allowable dimensions for matrices and vectors.
f		
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

rfftn

(You can use these to see the dimensions to which **rfftn** would pad a matrix or vector.)

rfftn scales the computed FFT by $1/(L*M)$.

See also **fft**, **ffti**, **fftm**, **fftm_i**, **fftn**, **rfft**, **rffti**, **rfftip**, **rfftnp**, **rfftp**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

rfftnp

rfftnp

a	Purpose	Computes a real 1- or 2-D FFT. Returns the results in a packed format.
b	Format	$y = \text{rfftnp}(x);$
c	Input	x NxK real matrix or K-length real vector.
d	Output	y Lx(M/2+1) matrix or (M/2+1)-length vector, where L and M are the smallest prime factor products greater than or equal to N and K, respectively.
e	Remarks	For 1-D FFT's, rfftnp returns the positive frequencies in ascending order in the first M/2 elements, and the Nyquist frequency in the last element. For 2-D FFT's, rfftnp returns the positive and negative frequencies for the row dimension, and for the column dimension it returns the positive frequencies in ascending order in the first M/2 columns, and the Nyquist frequencies in the last column. Usually the FFT of a real function is calculated to find the power density spectrum or to perform filtering on the waveform. In both these cases only the positive frequencies are required. (See also rfft and rfftn for routines that return the negative frequencies as well.)
f		rfftnp uses the Temperton prime factor FFT algorithm. This algorithm can compute the FFT of any vector or matrix whose dimensions can be expressed as the product of selected prime number factors. GAUSS implements the Temperton algorithm for any power of 2, 3, and 5, and one factor of 7. Thus, rfftnp can handle any matrix whose dimensions can be expressed as:
g		$2^p \times 3^q \times 5^r \times 7^s, \quad p, q, r \geq 0 \text{ for rows of matrix}$
h		$p > 0, q, r \geq 0 \text{ for columns of matrix}$
i		$p > 0, q, r \geq 0 \text{ for length of vector}$
j		$s = 0 \text{ or } 1 \text{ for all dimensions}$
k		If a dimension of x does not meet these requirements, it will be padded with zeros to the next allowable size before the FFT is computed.
l		rfftnp pads matrices to the next allowable size; however, it generally runs faster for matrices whose dimensions are highly composite numbers, i.e., products of several factors (to various powers), rather than powers of
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

rffftnp

a single factor. For example, even though it is bigger, a 33600x1 vector can compute as much as 20 percent faster than a 32768x1 vector, because 33600 is a highly composite number, $2^6 \times 3 \times 5^2 \times 7$, whereas 32768 is a simple power of 2, 2^{15} . For this reason, you may want to hand-pad matrices to optimum dimensions before passing them to **rffftnp**. The Run-Time Library includes two routines, **optn** and **optnevn**, for determining optimum dimensions. Use **optn** to determine optimum rows for matrices, and **optnevn** to determine optimum columns for matrices and optimum lengths for vectors.

The Run-Time Library also includes the **nextn** and **nextnevn** routines, for determining allowable dimensions for matrices and vectors. (You can use these to see the dimensions to which **rffftnp** would pad a matrix or vector.)

rffftnp scales the computed FFT by $1/(L \times M)$.

See also **fft, ffti, fftm, fftmi, fftn, rfft, rffti, rffftip, rffftn, rfftp**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

rfftp

rfftp

a	Purpose	Computes a real 1- or 2-D FFT. Returns the results in a packed format.
b	Format	$y = \text{rfftp}(x);$
c	Input	x NxK real matrix or K-length real vector.
d	Output	y Lx(M/2+1) matrix or (M/2+1)-length vector, where L and M are the smallest powers of 2 greater than or equal to N and K, respectively.
e	Remarks	<p>If a dimension of x is not a power of 2, it will be padded with zeros to the next allowable size before the FFT is computed.</p> <p>For 1-D FFT's, rfftp returns the positive frequencies in ascending order in the first M/2 elements, and the Nyquist frequency in the last element. For 2-D FFT's, rfftp returns the positive and negative frequencies for the row dimension, and for the column dimension it returns the positive frequencies in ascending order in the first M/2 columns, and the Nyquist frequencies in the last column. Usually the FFT of a real function is calculated to find the power density spectrum or to perform filtering on the waveform. In both these cases only the positive frequencies are required. (See also rfft and rfftn for routines that return the negative frequencies as well.)</p> <p>rfftp scales the computed FFT by 1/(L*M).</p> <p>rfftp uses the Temperton FFT algorithm.</p>
f	See also	fft, ffti, fftm, fftmi, fftn, rfft, rffti, rfftip, rfftn, rfftnp
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

rndbeta

Purpose	Computes pseudo-random numbers with beta distribution.		
Format	$x = \text{rndbeta}(r,c,a,b);$		
Input	r	scalar, number of rows of resulting matrix.	
	c	scalar, number of columns of resulting matrix.	
	a	MxN matrix, ExE conformable with RxC resulting matrix, shape parameters for beta distribution.	
	b	KxL matrix, ExE conformable with RxC resulting matrix, shape parameters for beta distribution.	
Output	x	RxC matrix, beta distributed pseudo-random numbers.	
Remarks	The properties of the pseudo-random numbers in x are: $E(x) = a / (a + b)$ $Var(x) = a \times b / (a + b + 1) \times (a + b)^2$ $x > 0$ $x < 1$ $a > 0$ $b > 0$		
Source	random.src		

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

rndcon, rndmult, rndseed

rndcon, rndmult, rndseed

a		
b	Purpose	Resets the parameters of the linear congruential random number generator that is the basis for rndu , rndi and rndn .
c	Format	rndcon <i>c</i> ; rndmult <i>a</i> ; rndseed <i>seed</i> ;
d	Portability	Windows, UNIX, OS/2
e		Parameter default values and ranges:
f		seed time(0), 0 < seed < 2^32
g		a 1664525 0 < a < 2^32
h		c 1013904223 0 ≤ c < 2^32
i		
j	Remarks	A linear congruential uniform random number generator is used by rndu , and is also called by rndn . These statements allow the parameters of this generator to be changed.
k		The procedure used to generate the uniform random numbers is as follows. First, the current “seed” is used to generate a new seed:
l		new_seed = (((<i>a</i> * <i>seed</i>) % 2 ³²) + <i>c</i>) % 2 ³²
m		(where % is the mod operator). Then a number between 0 and 1 is created by dividing the new seed by 2 ³² :
n		$x = \text{new_seed} / 2^{32}$
o		rndcon resets <i>c</i> .
p		rndmult resets <i>a</i> .
q		rndseed resets <i>seed</i> . This is the initial seed for the generator. The default is that GAUSS uses the clock to generate an initial seed when GAUSS is invoked.
r		GAUSS goes to the clock to seed the generator only when it is first started up. Therefore, if GAUSS is allowed to run for a long time, and if large numbers of random numbers are generated, there is a possibility of recycling (that is, the sequence of “random numbers” will repeat itself). However, the generator used has an extremely long cycle, and so that should not usually be a problem.
s		
t		
u		
v		
w		
x y z		

rndcon, rndmult, rndseed

The parameters set by these commands remain in effect until new commands are encountered, or until GAUSS is restarted.

See also **rndu, rndn, rndi, rndLCi, rndKMi**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

rndgam

rndgam

a	Purpose	Computes pseudo-random numbers with gamma distribution.
b	Format	$x = \text{rndgam}(r,c,alpha) ;$
c	Input	r scalar, number of rows of resulting matrix. c scalar, number of columns of resulting matrix. $alpha$ MxN matrix, ExE conformable with RxC resulting matrix, shape parameters for gamma distribution.
d	Output	x RxC matrix, gamma distributed pseudo-random numbers.
e	Remarks	The properties of the pseudo-random numbers in x are: $E(x) = alpha$ $Var(x) = alpha$ $x > 0$ $alpha > 0$ To generate gamma ($alpha,theta$) pseudo-random numbers where $theta$ is a scale parameter, multiply the result of rndgam by $theta$. Thus: $z = theta * \text{rndgam}(1,1,alpha)$ has the properties $E(z) = alpha \times theta$ $Var(z) = alpha \times theta^2$ $z > 0$ $alpha > 0$ $theta > 0$
f	Source	random.src
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

rndi

Purpose Returns a matrix of random integers, $0 \leq y < 2^{32}$.

Format $y = \text{rndi}(r, c);$

Input

r	scalar, row dimension.
c	scalar, column dimension.

Output

y	$r \times c$ matrix of random integers between 0 and $2^{32} - 1$, inclusive.
-----	--

Remarks r and c will be truncated to integers if necessary.

This generator is automatically seeded using the system clock when GAUSS first starts. However, that can be overridden using the **rndseed** statement or using **rndus**.

Each seed is generated from the preceding seed, using the formula

$$\text{new_seed} = (((a * \text{seed}) \% 2^{32}) + c) \% 2^{32}$$

where % is the mod operator. The new seeds are the values returned. The multiplicative constant and the additive constant may be changed using **rndmult** and **rndcon** respectively.

See also **rndu**, **rndus**, **rndn**, **rndcon**, **rndmult**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

rndKMbeta

rndKMbeta

a	Purpose	Computes beta pseudo-random numbers.
b	Format	$\{ x, newstate \} = \text{rndKMbeta}(r,c,a,b,state);$
c	Input	r scalar, number of rows of resulting matrix.
d		c scalar, number of columns of resulting matrix.
e		a $r \times c$ matrix, or $r \times 1$ vector, or $1 \times c$ vector, or scalar, first shape argument for beta distribution.
f		b $r \times c$ matrix, or $r \times 1$ vector, or $1 \times c$ vector, or scalar, second shape argument for beta distribution.
g		$state$ scalar or 500x1 vector.
h		Scalar case:
i		$state$ = starting seed value only. If -1, GAUSS computes the starting seed based on the system clock.
j		500x1 vector case:
k		$state$ = the state vector returned from a previous call to one of the rndKM random number functions.
l	Output	x $r \times c$ matrix, beta distributed random numbers.
m		$newstate$ 500x1 vector, the updated state.
n	Remarks	The properties of the pseudo-random numbers in x are:
o		$E(x) = \frac{a}{a+b}, Var(x) = \frac{(a*b)}{(a+b+1)*(a+b)^2}$
p		$0 < x < 1, a > 0, b > 0$
q		r and c will be truncated to integers if necessary.
r	Source	randkm.src
s	Technical Notes	rndKMbeta uses the recur-with-carry KISS+Monster algorithm described in the rndKMi Technical Notes.
t		
u		
v		
w		
x y z		

rndKMgam

Purpose	Computes Gamma pseudo-random numbers.		
Format	{ <i>x</i> , <i>newstate</i> } = rndKMgam (<i>r</i> , <i>c</i> , <i>alpha</i> , <i>state</i>);		
Input	<i>r</i>	scalar, number of rows of resulting matrix.	
	<i>c</i>	scalar, number of columns of resulting matrix.	
	<i>alpha</i>	<i>rxc</i> matrix, or <i>rx1</i> vector, or <i>1xc</i> vector, or scalar, shape argument for gamma distribution.	
	<i>state</i>	scalar or 500x1 vector.	
		Scalar case: <i>state</i> = starting seed value only. If -1, GAUSS computes the starting seed based on the system clock. 500x1 vector case: <i>state</i> = the state vector returned from a previous call to one of the rndKM random number functions.	
Output	<i>x</i>	<i>rxc</i> matrix, gamma distributed random numbers.	
	<i>newstate</i>	500x1 vector, the updated state.	
Remarks	<p>The properties of the pseudo-random numbers in <i>x</i> are:</p> $E(x) = \alpha, \text{Var}(x) = \alpha$ $x > 0, \alpha > 0.$ <p>To generate gamma(<i>alpha</i>, <i>theta</i>) pseudo-random numbers where <i>theta</i> is a scale parameter, multiply the result of rndgam by <i>theta</i>.</p> <p>Thus</p> $z = \text{theta} * \text{rndgam}(1,1,\alpha);$ <p>has the properties</p> $E(z) = \alpha * \text{theta}, \text{Var}(z) = \alpha * \text{theta}^2$ $z > 0, \alpha > 0, \text{theta} > 0.$ <p><i>r</i> and <i>c</i> will be truncated to integers if necessary.</p>		
Source	randkm.src		

rndKMgam

Technical Notes **rndKMgam** uses the recur-with-carry KISS+Monster algorithm described in the **rndKMi** Technical Notes.

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

rndKMi

Purpose	Returns a matrix of random integers, $0 \leq y < 2^{32}$, and the state of the random number generator.	
Format	<code>{ y, newstate } = rndKMi(r,c,state);</code>	
Input	<i>r</i>	scalar, row dimension.
	<i>c</i>	scalar, column dimension.
	<i>state</i>	scalar or 500x1 vector.
	Scalar case: <i>state</i> = starting seed value. If -1, GAUSS computes the starting seed based on the system clock.	
	500x1 vector case: <i>state</i> = the <i>state</i> vector returned from a previous call to one of the rndKM random number generators.	
Output	<i>y</i>	<i>r</i> x <i>c</i> matrix of random integers between 0 and $2^{32} - 1$, inclusive.
	<i>newstate</i>	500x1 vector, the updated state.
Remarks	<i>r</i> and <i>c</i> will be truncated to integers if necessary.	
Example	<p>This example generates two thousand vectors of random integers, each with one million elements. The state of the random number generator after each iteration is used as an input to the next generation of random numbers.</p> <pre> state = 13; n = 2000; k = 1000000; c = 0; min = 2^32+1; max = -1; </pre>	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

rndKMi

```
do while c < n;  
    { y,state } = rndKMi(k,1,state);  
    min = minc(min | minc(y));  
    max = maxc(max | maxc(y));  
    c = c + k;  
endo;  
  
print "min " min;  
print "max " max;
```

See also **rndKMn**, **rndKMu**

**Technical
Notes**

rndKMi generates random integers using a KISS+Monster algorithm developed by George Marsaglia. KISS initializes the sequence used in the recur-with-carry Monster random number generator. For more information on this generator see <http://www.Aptech.com/random>.

rndKMn

Purpose	Returns a matrix of standard normal (pseudo) random variables and the state of the random number generator.	
Format	$\{ y, \text{ newstate } \} = \text{rndKMn}(r, c, \text{state});$	
Input	<i>r</i>	scalar, row dimension.
	<i>c</i>	scalar, column dimension.
	<i>state</i>	scalar or 500x1 vector.
	Scalar case: <i>state</i> = starting seed value. If -1, GAUSS computes the starting seed based on the system clock.	
	500x1 vector case: <i>state</i> = the <i>state</i> vector returned from a previous call to one of the rndKM random number generators.	
Output	<i>y</i>	<i>r</i> x <i>c</i> matrix of standard normal random numbers.
	<i>newstate</i>	500x1 vector, the updated state.
Remarks	<i>r</i> and <i>c</i> will be truncated to integers if necessary.	
Example	<p>This example generates two thousand vectors of standard normal random numbers, each with one million elements. The state of the random number generator after each iteration is used as an input to the next generation of random numbers.</p> <pre> state = 13; n = 2000; k = 1000000; c = 0; submean = {}; </pre>	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

rndKMn

```
do while c < n;  
    { y,state } = rndKMn(k,1,state);  
    submean = submean | meanc(y);  
    c = c + k;  
endo;  
  
mean = meanc(submean);  
print mean;
```

See also **rndKMu**, **rndKMi**

**Technical
Notes**

rndKMn calls the uniform random number generator that is the basis for **rndKMu** multiple times for each normal random number generated. This is the recur-with-carry KISS+Monster algorithm described in the **rndKMi** Technical Notes. Potential normal random numbers are filtered using the fast acceptance-rejection algorithm proposed by Kinderman, A.J. and J.G. Ramage, "Computer Generation of Normal Random Numbers," Journal of the American Statistical Association, December 1976, Volume 71, Number 356, pp. 893-896.

rndKMnb

Purpose Computes negative binomial pseudo-random numbers.

Format `{ x, newstate } = rndKMnb(r,c,k,p,state);`

Input

- r* scalar, number of rows of resulting matrix.
- c* scalar, number of columns of resulting matrix.
- k* *r*×*c* matrix, or *r*×1 vector, or 1×*c* vector, or scalar, "event" argument for negative binomial distribution.
- p* *r*×*c* matrix, or *r*×1 vector, or 1×*c* vector, or scalar, "probability" argument for negative binomial distribution.
- state* scalar or 500×1 vector.

Scalar case:

state = starting seed value only. If -1, GAUSS computes the starting seed based on the system clock.

500×1 vector case:

state = the state vector returned from a previous call to one of the **rndKM** random number functions.

Output

- x* *r*×*c* matrix, negative binomial distributed random numbers.
- newstate* 500×1 vector, the updated state.

Remarks The properties of the pseudo-random numbers in *x* are:

$$E(x) = \frac{k \cdot p}{(1 - p)}, \text{Var}(x) = \frac{k \cdot p}{(1 - p)^2}$$

$$x = 0, 1, \dots, k > 0, 0 < p < 1$$

r and *c* will be truncated to integers if necessary.

Source `randkm.src`

Technical Notes **rndKMnb** uses the recur-with-carry KISS+Monster algorithm described in the **rndKMi** Technical Notes.

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

rndKMp

rndKMp

Purpose	Computes Poisson pseudo-random numbers.		
Format	{ <i>x</i> , <i>newstate</i> } = rndKMp (<i>r</i> , <i>c</i> , <i>lambda</i> , <i>state</i>);		
Input	<i>r</i>	scalar, number of rows of resulting matrix.	
	<i>c</i>	scalar, number of columns of resulting matrix.	
	<i>lambda</i>	<i>rx</i> <i>c</i> matrix, or <i>rx</i> 1 vector, or 1 <i>x</i> <i>c</i> vector, or scalar, shape argument for Poisson distribution.	
	<i>state</i>	scalar or 500x1 vector.	
	Scalar case: <i>state</i> = starting seed value only. If -1, GAUSS computes the starting seed based on the system clock. 500x1 vector case: <i>state</i> = the state vector returned from a previous call to one of the rndKM random number functions.		
Output	<i>x</i>	<i>rx</i> <i>c</i> matrix, Poisson distributed random numbers.	
	<i>newstate</i>	500x1 vector, the updated state.	
Remarks	The properties of the pseudo-random numbers in <i>x</i> are:		
	$E(x) = lambda, Var(x) = lambda$		
	$x = 0, 1, ..., lambda > 0.$		
	<i>r</i> and <i>c</i> will be truncated to integers if necessary.		
Source	randkm.src		
Technical Notes	rndKMp uses the recur-with-carry KISS+Monster algorithm described in the rndKM Technical Notes.		

rndKMu

Purpose	Returns a matrix of uniform (pseudo) random variables and the state of the random number generator.	
Format	$\{ y, \text{ newstate } \} = \text{rndKM}u(r, c, state);$	
Input	<i>r</i>	scalar, row dimension.
	<i>c</i>	scalar, column dimension.
	<i>state</i>	scalar, 2x1 vector, or 500x1 vector.
	Scalar case: <i>state</i> = starting seed value. If -1, GAUSS computes the starting seed based on the system clock. 2x1 vector case: [1] the starting seed, uses the system clock if -1 [2] 0 for $0 \leq y < 1$ 1 for $0 \leq y \leq 1$ 500x1 vector case: <i>state</i> = the <i>state</i> vector returned from a previous call to one of the rndKM random number generators.	
Output	<i>y</i>	<i>r</i> x <i>c</i> matrix of uniform random numbers, $0 \leq y < 1$.
	<i>newstate</i>	500x1 vector, the updated state.
Remarks	<i>r</i> and <i>c</i> will be truncated to integers if necessary.	
Example	This example generates two thousand vectors of uniform random numbers, each with one million elements. The state of the random number generator after each iteration is used as an input to the next generation of random numbers. <pre>state = 13; n = 2000; k = 1000000; c = 0; submean = { };</pre>	

rndKMu

```
do while c < n;  
    { y,state } = rndKMu(k,1,state);  
    submean = submean | meanc(y);  
    c = c + k;  
endo;  
  
mean = meanc(submean);  
print 0.5-mean;
```

See also **rndKMn**, **rndKMi**

**Technical
Notes**

rndKMu uses the recur-with-carry KISS-Monster algorithm described in the **rndKMi** Technical Notes. Random integer seeds from 0 to $2^{32}-1$ are generated. Each integer is divided by 2^{32} or $2^{32}-1$.

rndKMvm

Purpose	Computes von Mises pseudo-random numbers.		
Format	{ <i>x</i> , <i>newstate</i> } = rndKMvm (<i>r</i> , <i>c</i> , <i>m</i> , <i>k</i> , <i>state</i>);		
Input	<i>r</i>	scalar, number of rows of resulting matrix.	
	<i>c</i>	scalar, number of columns of resulting matrix.	
	<i>m</i>	<i>r</i> × <i>c</i> matrix, or <i>r</i> ×1 vector, or 1× <i>c</i> vector, or scalar, means for vm distribution.	
	<i>k</i>	<i>r</i> × <i>c</i> matrix, or <i>r</i> ×1 vector, or 1× <i>c</i> vector, or scalar, shape argument for vm distribution.	
	<i>state</i>	scalar or 500×1 vector.	
Scalar case:			
<i>state</i> = starting seed value only. If -1, GAUSS computes the starting seed based on the system clock.			
500×1 vector case:			
<i>state</i> = the state vector returned from a previous call to one of the rndKM random number functions.			
Output	<i>x</i>	<i>r</i> × <i>c</i> matrix, von Mises distributed random numbers.	
	<i>newstate</i>	500×1 vector, the updated state.	
Remarks	<i>r</i> and <i>c</i> will be truncated to integers if necessary.		
Source	randkm.src		
Technical Notes	rndKMvm uses the recur-with-carry KISS+Monster algorithm described in the rndKMi Technical Notes.		

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

rndLCbeta

rndLCbeta

Purpose	Computes beta pseudo-random numbers.	
Format	{ <i>x</i> , <i>newstate</i> } = rndLCbeta (<i>r</i> , <i>c</i> , <i>a</i> , <i>b</i> , <i>state</i>);	
Input	<i>r</i>	scalar, number of rows of resulting matrix.
	<i>c</i>	scalar, number of columns of resulting matrix.
	<i>a</i>	<i>r</i> × <i>c</i> matrix, or <i>r</i> ×1 vector, or 1× <i>c</i> vector, or scalar, first shape argument for beta distribution.
	<i>b</i>	<i>r</i> × <i>c</i> matrix, or <i>r</i> ×1 vector, or 1× <i>c</i> vector, or scalar, second shape argument for beta distribution.
	<i>state</i>	scalar, 3×1 vector, or a 4×1 state vector from a previous call to the function.
	Scalar case:	
	<i>state</i> = starting seed value only. System default values are used for the additive and multiplicative constants.	
	The defaults are 1013904223, and 1664525, respectively.	
	These may be changed with rndcon and rndmult .	
	if <i>state</i> = -1, GAUSS computes the starting seed based on the system clock.	
Output	3×1 vector case:	
	[1]	the starting seed, uses the system clock if -1
	[2]	the multiplicative constant
	[3]	the additive constant
	<i>x</i>	<i>r</i> × <i>c</i> matrix, beta distributed random numbers.
	<i>newstate</i>	4×1 vector:
	[1]	the updated seed
	[2]	the multiplicative constant
	[3]	the additive constant
	[4]	the original initialization seed
Remarks:	The properties of the pseudo-random numbers in <i>x</i> are:	
$E(x) = \frac{a}{a+b}, Var(x) = \frac{(a*b)}{(a+b+1)*(a+b)^2}$		

$$0 < x < 1, a > 0, b > 0$$

r and *c* will be truncated to integers if necessary.

Source
Technical
Notes

`randlc.src`

This function uses a linear congruential method, discussed in Kennedy, W.J. Jr., and J.E. Gentle, Statistical Computing, Marcel Dekker, Inc. 1980, pp. 136-147. Each seed is generated from the preceding seed using the formula

$$\text{new_seed} = ((a * \text{seed}) \% 2^{32}) + c) \% 2^{32}$$

where % is the mod operator and where a is the multiplicative constant and c is the additive constant.

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

rndLCgam

rndLCgam

a	Purpose	Computes Gamma pseudo-random numbers.	
b			
c	Format	{ <i>x</i> , <i>newstate</i> } = rndLCgam (<i>r</i> , <i>c</i> , <i>alpha</i> , <i>state</i>);	
d	Input	<i>r</i>	scalar, number of rows of resulting matrix.
e		<i>c</i>	scalar, number of columns of resulting matrix.
f		<i>alpha</i>	<i>rx</i> <i>c</i> matrix, or <i>rx</i> 1 vector, or 1 <i>x</i> <i>c</i> vector, or scalar, shape argument for gamma distribution.
g		<i>state</i>	scalar, 3x1 vector, or a 4x1 state vector from a previous call to the function.
h		Scalar case:	
i		<i>state</i> = starting seed value only. System default values are used for the additive and multiplicative constants.	
j		The defaults are 1013904223, and 1664525, respectively. These may be changed with rndcon and rndmult .	
k		if <i>state</i> = -1, GAUSS computes the starting seed based on the system clock.	
l		3x1 vector case:	
m		[1]	the starting seed, uses the system clock if -1
n	Output	[2]	the multiplicative constant
o		[3]	the additive constant
p		<i>x</i>	
q		<i>rx</i> <i>c</i> matrix, gamma distributed random numbers.	
r		<i>newstate</i>	4x1 vector:
s		[1]	the updated seed
t		[2]	the multiplicative constant
u		[3]	the additive constant
v		[4]	the original initialization seed
w	Remarks	The properties of the pseudo-random numbers in <i>x</i> are:	
x y z		$E(x) = alpha$, $Var(x) = alpha$ $x > 0$, $alpha > 0$. To generate gamma (<i>alpha</i> , <i>theta</i>) pseudo-random numbers where <i>theta</i> is a scale parameter, multiply the result of rndgam by <i>theta</i> .	

Thus

$$z = \text{theta} * \text{rndgam}(1,1,\text{alpha});$$

has the properties

$$E(z) = \text{alpha} * \text{theta}, \text{Var}(z) = \text{alpha} * \text{theta}^2$$

$$z > 0, \text{alpha} > 0, \text{theta} > 0.$$

r and c will be truncated to integers if necessary.

Source Technical Notes

`randlc.src`

This function uses a linear congruential method, discussed in Kennedy, W.J. Jr., and J.E. Gentle, Statistical Computing, Marcel Dekker, Inc. 1980, pp. 136-147. Each seed is generated from the preceding seed using the formula

$$\text{new_seed} = (((a * \text{seed}) \% 2^{32}) + c) \% 2^{32}$$

where $\%$ is the mod operator and where a is the multiplicative constant and c is the additive constant.

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

rndLCi

rndLCi

Purpose	Returns a matrix of random integers, $0 \leq y < 2^{32}$, and the state of the random number generator.			
Format	{ <i>y</i> , <i>newstate</i> } = rndLCi (<i>r</i> , <i>c</i> , <i>state</i>);			
Input	<i>r</i>	scalar, row dimension.		
	<i>c</i>	scalar, column dimension.		
	<i>state</i>	scalar, 3x1 vector, or a 4x1 state vector from a previous call to the function.		
	Scalar case:			
	<i>state</i> = starting seed value only. System default values are used for the additive and multiplicative constants.			
	The defaults are 1013904223, and 1664525, respectively.			
	These may be changed with rndcon and rndmult .			
	if <i>state</i> = 0, GAUSS computes the starting seed based on the system clock.			
	3x1 vector case:			
		[1]	the starting seed, uses the system clock if 0	
	[2]	the multiplicative constant		
	[3]	the additive constant		
Output	<i>y</i>	<i>r</i> x <i>c</i> matrix of random integers between 0 and $2^{32} - 1$, inclusive.		
	<i>newstate</i>	4x1 vector:		
		[1]	the updated seed	
		[2]	the multiplicative constant	
		[3]	the additive constant	
		[4]	the original initialization seed	
Remarks	<i>r</i> and <i>c</i> will be truncated to integers if necessary.			
	Each seed is generated from the preceding seed, using the formula			
	$\text{new_seed} = (((a * seed) \% 2^{32}) + c) \% 2^{32}$			
where % is the mod operator and where a is the multiplicative constant and c is the additive constant. The new seeds are the values returned.				

Example

```
state = 13;
n = 2000000000;
k = 1000000;
c = 0;
min = 2^32+1;
max = -1;

do while c < n;
    { y,state } = rndLCi(k,1,state);
    min = minc(min | minc(y));
    max = maxc(max | maxc(y));
    c = c + k;
endo;

print "min " min;
print "max " max;
```

See also **rndLCn, rndLCu, rndcon, rndmult**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

rndLCn

rndLCn

a	Purpose	Returns a matrix of standard normal (pseudo) random variables and the state of the random number generator.	
b			
c	Format	{ <i>y</i> , <i>newstate</i> } = rndLCn (<i>r</i> , <i>c</i> , <i>state</i>);	
d			
e	Input	<i>r</i>	scalar, row dimension.
f		<i>c</i>	scalar, column dimension.
g		<i>state</i>	scalar, 3x1 vector, or a 4x1 state vector from a previous call to the function.
h		Scalar case:	
i		<i>state</i> = starting seed value only. System default values are used for the additive and multiplicative constants.	
j	Output	The defaults are 1013904223, and 1664525, respectively.	
k		These may be changed with rndcon and rndmult .	
l		if <i>state</i> = 0, GAUSS computes the starting seed based on the system clock.	
m		3x1 vector case:	
n		[1]	the starting seed, uses the system clock if 0
o		[2]	the multiplicative constant
p		[3]	the additive constant
q			
r			
s			
t	Remarks	<i>y</i>	<i>r</i> x <i>c</i> matrix of standard normal random numbers.
u		<i>newstate</i>	4x1 vector:
v	Example	[1]	the updated seed
w		[2]	the multiplicative constant
x		[3]	the additive constant
y		[4]	the original initialization seed
z			
	Example	<i>state</i> = 13;	
		<i>n</i> = 2000000000;	
		<i>k</i> = 1000000;	
		<i>c</i> = 0;	


```

submean = {};

do while c < n;
    { y,state } = rndLCn(k,1,state);
    submean = submean | meanc(y);
    c = c + k;
enddo;

mean = meanc(submean);
print mean;

```

See also **rndLCu**, **rndLCi**, **rndcon**, **rndmult**

Technical Notes The normal random number generator is based on the uniform random number generator, using the fast acceptance-rejection algorithm proposed by Kinderman, A.J. and J.G. Ramage, "Computer Generation of Normal Random Numbers," Journal of the American Statistical Association, December 1976, Volume 71, Number 356, pp. 893-896. This algorithm calls the linear congruential uniform random number generator multiple times for each normal random number generated. See **rndLCu** for a description of the uniform random number generator algorithm.

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

rndLCnb

a	Purpose	Computes negative binomial pseudo-random numbers.	
b			
c	Format	{ <i>x</i> , <i>newstate</i> } = rndLCnb (<i>r</i> , <i>c</i> , <i>k</i> , <i>p</i> , <i>state</i>);	
d	Input	<i>r</i>	scalar, number of rows of resulting matrix.
e		<i>c</i>	scalar, number of columns of resulting matrix.
f		<i>k</i>	<i>rxc</i> matrix, or <i>rx1</i> vector, or <i>1xc</i> vector, or scalar, "event" argument for negative binomial distribution.
g		<i>p</i>	<i>rxc</i> matrix, or <i>rx1</i> vector, or <i>1xc</i> vector, or scalar, "probability" argument for negative binomial distribution.
h		<i>state</i>	scalar, 3x1 vector, or a 4x1 state vector from a previous call to the function.
i		Scalar case:	
j		<i>state</i> = starting seed value only. System default values are used for the additive and multiplicative constants.	
k		The defaults are 1013904223, and 1664525, respectively. These may be changed with rndcon and rndmult .	
l		if <i>state</i> = -1, GAUSS computes the starting seed based on the system clock.	
m		3x1 vector case:	
n	Output	[1]	the starting seed, uses the system clock if -1
o		[2]	the multiplicative constant
p		[3]	the additive constant
q			
r		<i>x</i>	<i>rxc</i> matrix, negative binomial distributed random numbers.
s		<i>newstate</i>	4x1 vector:
t		[1]	the updated seed
u		[2]	the multiplicative constant
v		[3]	the additive constant
w		[4]	the original initialization seed
x y z	Remarks	The properties of the pseudo-random numbers in <i>x</i> are:	
		$E(x) = \frac{k*p}{(1-p)}, Var(x) = \frac{k*p}{(1-p)^2}$	

$$x = 0, 1, \dots, k > 0, 0 < p < 1$$

r and *c* will be truncated to integers if necessary.

Source
Technical
Notes

`randlc.src`

This function uses a linear congruential method, discussed in Kennedy, W.J. Jr., and J.E. Gentle, Statistical Computing, Marcel Dekker, Inc. 1980, pp. 136-147. Each seed is generated from the preceding seed using the formula

$$\text{new_seed} = ((a * \text{seed}) \% 2^{32}) + c) \% 2^{32}$$

where % is the mod operator and where a is the multiplicative constant and c is the additive constant.

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

rndLCp

rndLCp

a	Purpose	Computes Poisson pseudo-random numbers.	
b			
c	Format	{ <i>x</i> , <i>newstate</i> } = rndLCp (<i>r</i> , <i>c</i> , <i>lambda</i> , <i>state</i>);	
d	Input	<i>r</i>	scalar, number of rows of resulting matrix.
e		<i>c</i>	scalar, number of columns of resulting matrix.
f		<i>lambda</i>	<i>rx</i> <i>c</i> matrix, or <i>rx</i> 1 vector, or 1 <i>x</i> <i>c</i> vector, or scalar, shape argument for Poisson distribution.
g		<i>state</i>	scalar, 3x1 vector, or a 4x1 state vector from a previous call to the function.
h		Scalar case:	
i		<i>state</i> = starting seed value only. System default values are used for the additive and multiplicative constants.	
j		The defaults are 1013904223, and 1664525, respectively. These may be changed with rndcon and rndmult .	
k		if <i>state</i> = -1, GAUSS computes the starting seed based on the system clock.	
l		3x1 vector case:	
m		[1]	the starting seed, uses the system clock if -1
n	Output	[2]	the multiplicative constant
o		[3]	the additive constant
p		<i>x</i>	
q		<i>rx</i> <i>c</i> matrix, Poisson distributed random numbers.	
r		<i>newstate</i>	4x1 vector:
s		[1]	the updated seed
t		[2]	the multiplicative constant
u		[3]	the additive constant
v		[4]	the original initialization seed
w		Remarks	
x y z		The properties of the pseudo-random numbers in <i>x</i> are: $E(x) = lambda$, $Var(x) = lambda$ $x = 0, 1, ..., lambda > 0$. <i>r</i> and <i>c</i> will be truncated to integers if necessary.	

Source	<code>randlc.src</code>
Technical Notes	<p>This function uses a linear congruential method, discussed in Kennedy, W.J. Jr., and J.E. Gentle, Statistical Computing, Marcel Dekker, Inc. 1980, pp. 136-147. Each seed is generated from the preceding seed using the formula</p> $\text{new_seed} = (((a * seed) \% 2^{32}) + c) \% 2^{32}$ <p>where % is the mod operator and where a is the multiplicative constant and c is the additive constant.</p>

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

rndLCu

rndLCu

a	Purpose	Returns a matrix of uniform (pseudo) random variables and the state of the random number generator.	
b			
c	Format	{ <i>y</i> , <i>newstate</i> } = rndLCu (<i>r</i> , <i>c</i> , <i>state</i>);	
d			
e	Input	<i>r</i>	scalar, row dimension.
f		<i>c</i>	scalar, column dimension.
g		<i>state</i>	scalar, 3x1 vector, or a 4x1 state vector from a previous call to the function.
h			Scalar case:
i			<i>state</i> = starting seed value only. System default values are used for the additive and multiplicative constants.
j			The defaults are 1013904223, and 1664525, respectively.
k			These may be changed with rndcon and rndmult .
l			if <i>state</i> = 0, GAUSS computes the starting seed based on the system clock.
m			3x1 vector case:
n		[1]	the starting seed, uses the system clock if 0
o		[2]	the multiplicative constant
p		[3]	the additive constant
q	Output	<i>y</i>	<i>r</i> x <i>c</i> matrix of uniform random numbers, 0 <= <i>y</i> < 1.
r		<i>newstate</i>	4x1 vector:
s		[1]	the updated seed
t		[2]	the multiplicative constant
u		[3]	the additive constant
v		[4]	the original initialization seed
w	Remarks	<i>r</i> and <i>c</i> will be truncated to integers if necessary.	
x y z		Each seed is generated from the preceding seed, using the formula	
		new_seed = (((<i>a</i> * <i>seed</i>) % 2 ³²) + <i>c</i>) % 2 ³²	
		where % is the mod operator and where <i>a</i> is the multiplicative constant and <i>c</i> is the additive constant. A number between 0 and 1 is created by dividing new_seed by 2 ³² .	

```

Example  state = 13;
         n = 2000000000;
         k = 1000000;
         c = 0;
         submean = {};

         do while c < n;
             { y,state } = rndLCu(k,1,state);
             submean = submean | meanc(y);
             c = c + k;
         endo;

         mean = meanc(submean);
         print 0.5-mean;

```

See also **rndLCn, rndLCi, rndcon, rndmult**

Technical Notes This function uses a linear congruential method, discussed in Kennedy, W. J. Jr., and J. E. Gentle, Statistical Computing, Marcel Dekker, Inc., 1980, pp. 136-147.

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

rndLCvm

rndLCvm

a	Purpose	Computes von Mises pseudo-random numbers.	
b			
c	Format	{ <i>x</i> , <i>newstate</i> } = rndLCvm (<i>r</i> , <i>c</i> , <i>m</i> , <i>k</i> , <i>state</i>);	
d	Input	<i>r</i>	scalar, number of rows of resulting matrix.
e		<i>c</i>	scalar, number of columns of resulting matrix.
f		<i>m</i>	<i>rx</i> <i>c</i> matrix, or <i>rx</i> 1 vector, or 1 <i>x</i> <i>c</i> vector, or scalar, means for vm distribution.
g		<i>k</i>	<i>rx</i> <i>c</i> matrix, or <i>rx</i> 1 vector, or 1 <i>x</i> <i>c</i> vector, or scalar, shape argument for vm distribution.
h		<i>state</i>	scalar, 3x1 vector, or a 4x1 state vector from a previous call to the function.
i		Scalar case:	
j		<i>state</i> = starting seed value only. System default values are used for the additive and multiplicative constants.	
k		The defaults are 1013904223, and 1664525, respectively.	
l		These may be changed with rndcon and rndmult .	
m		if <i>state</i> = -1, GAUSS computes the starting seed based on the system clock.	
n	Output	3x1 vector case:	
o		[1]	the starting seed, uses the system clock if -1
p		[2]	the multiplicative constant
q		[3]	the additive constant
r		<i>x</i>	<i>rx</i> <i>c</i> matrix, von Mises distributed random numbers.
s			
t		<i>newstate</i>	4x1 vector:
u		[1]	the updated seed
v		[2]	the multiplicative constant
w		[3]	the additive constant
x y z		[4]	the original initialization seed
	Remarks	<i>r</i> and <i>c</i> will be truncated to integers if necessary.	
	Source	randlc.src	

rndLCvm

Technical
Notes

This function uses a linear congruential method, discussed in Kennedy, W.J. Jr., and J.E. Gentle, Statistical Computing, Marcel Dekker, Inc. 1980, pp. 136-147. Each seed is generated from the preceding seed using the formula

$$\text{new_seed} = (((a * seed) \% 2^{32}) + c) \% 2^{32}$$

where % is the mod operator and where a is the multiplicative constant and c is the additive constant.

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

rndn

rndn

a	Purpose	Creates a matrix of standard Normal (pseudo) random numbers.
b	Format	$y = \text{rndn}(r, c);$
c	Input	r scalar, row dimension. c scalar, column dimension.
d	Output	y $R \times C$ matrix of Normal random numbers having a mean of 0 and standard deviation of 1.
e	Remarks	r and c will be truncated to integers if necessary. The Normal random number generator is based upon the uniform random number generator. To reseed them both, use the rndseed statement. The other parameters of the uniform generator can be changed using rndcon , rndmod , and rndmult .
f	Example	$x = \text{rndn}(8100, 1);$ $m = \text{meanc}(x);$ $s = \text{stdc}(x);$ $m = 0.002810$ $s = 0.997087$ In this example, a sample of 8100 Normal random numbers is drawn, and the mean and standard deviation are computed for the sample.
g	See also	rndu , rndcon
h	Technical Notes	This function uses the fast acceptance-rejection algorithm proposed by Kinderman, A. J., and J. G. Ramage. "Computer Generation of Normal Random Numbers." <i>Journal of the American Statistical Association</i> . Vol. 71 No. 356, Dec. 1976, 893-96.
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

rndnb

Purpose Computes pseudo-random numbers with negative binomial distribution.

Format $x = \text{rndnb}(r,c,k,p);$

Input

r	scalar, number of rows of resulting matrix.
c	scalar, number of columns of resulting matrix.
k	MxN matrix, ExE conformable with RxC resulting matrix, “event” parameters for negative binomial distribution.
p	KxL matrix, ExE conformable with RxC resulting matrix, probability parameters for negative binomial distribution.

Output x RxC matrix, negative binomial distributed pseudo-random numbers.

Remarks The properties of the pseudo-random numbers in x are:

$$\begin{aligned}
 E(x) &= k \times p / (1 - p) \\
 Var(x) &= k \times p / (1 - p)^2 \\
 x &= 0, 1, 2, \dots, k \\
 k &> 0 \\
 p &> 0 \\
 p &< 1
 \end{aligned}$$

Source random.src

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

rndp

rndp

a	Purpose	Computes pseudo-random numbers with Poisson distribution.
b	Format	$x = \mathbf{rndp}(r,c,\lambda) ;$
c	Input	r scalar, number of rows of resulting matrix.
d		c scalar, number of columns of resulting matrix.
e		λ MxN matrix, ExE conformable with RxC resulting matrix, shape parameters for Poisson distribution.
f	Output	x RxC matrix, Poisson distributed pseudo-random numbers.
g	Remarks	The properties of the pseudo-random numbers in x are:
h		$Ex = \lambda$
i		$Var(x) = \lambda$
j		$x = 0, 1, 2, \dots$
k		$\lambda > 0$
l	Source	random.src
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

rndu

Purpose	Creates a matrix of uniform (pseudo) random variables.	
Format	$y = \text{rndu}(r,c);$	
Input	r	scalar, row dimension.
	c	scalar, column dimension.
Output	y	$R \times C$ matrix of uniform random variables between 0 and 1.
Remarks	<p>r and c will be truncated to integers if necessary.</p> <p>This generator is automatically seeded using the clock when GAUSS is first started. However, that can be overridden using the rndseed statement or by using rndus.</p> <p>The seed is automatically updated as a random number is generated (see above under rndcon). Thus, if GAUSS is allowed to run for a long time, and if large numbers of random numbers are generated, there is a possibility of recycling. This is a 32-bit generator, though, so the range is sufficient for most applications.</p>	
Example	<pre>x = rndu(8100,1); y = meanc(x); z = stdc(x); y = 0.500205 z = 0.289197</pre> <p>In this example, a sample of 8100 uniform random numbers is generated, and the mean and standard deviation are computed for the sample.</p>	
See also	rndn, rndcon, rndmod, rndmult, rndseed	
Technical Notes	This function uses a multiplicative-congruential method. This method is discussed in Kennedy, W.J., Jr., and J.E. Gentle. <i>Statistical Computing</i> . Marcel Dekker, Inc., NY, 1980, 136-147.	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

rndvm

rndvm

a		
b		
c		
d		
e		
f		
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

Purpose	Computes von Mises pseudo-random numbers.	
Format	$x = \text{rndvm}(r, c, m, k);$	
Input	r	scalar, number of rows of resulting matrix.
	c	scalar, number of columns of resulting matrix.
	m	NxK matrix, ExE conformable with $r \times c$, means for von Mises distribution.
	k	LxM matrix, ExE conformable with $r \times c$, shape argument for von Mises distribution.
Output	x	$r \times c$ matrix, von Mises distributed random numbers.
Source	random.src	

rotater

Purpose Rotates the rows of a matrix.

Format `y = rotater(x,r);`

Input
`x` NxK matrix to be rotated.
`r` Nx1 or 1x1 matrix specifying the amount of rotation.

Output `y` NxK rotated matrix.

Remarks The rotation is performed horizontally within each row of the matrix. A positive rotation value will cause the elements to move to the right. A negative rotation value will cause the elements to move to the left. In either case, the elements that are pushed off the end of the row will wrap around to the opposite end of the same row.

If the rotation value is greater than or equal to the number of columns in `x`, then the rotation value will be calculated using $(r \% \text{cols}(x))$.

Example `y = rotater(x,r);`

If $x = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$ and $r = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$ Then $y = \begin{bmatrix} 3 & 1 & 2 \\ 5 & 6 & 4 \end{bmatrix}$

If $x = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix}$ and $r = \begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \end{bmatrix}$ Then $y = \begin{bmatrix} 1 & 2 & 3 \\ 6 & 4 & 5 \\ 8 & 9 & 7 \\ 10 & 11 & 12 \end{bmatrix}$

See also `shiftr`

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

round

round

a	Purpose	Rounds to the nearest integer.
b	Format	$y = \text{round}(x);$
c	Input	x NxK matrix.
d	Output	y NxK matrix containing the rounded elements of x .
e	Example	$\text{let } x = \{ \begin{matrix} 77.68 & -14.10, \\ 4.73 & -158.88 \end{matrix} \};$
f		$y = \text{round}(x);$
g		$y = \begin{matrix} 78 & -14 \\ 5 & -159 \end{matrix}$
h	See also	trunc, floor, ceil
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

rows

Purpose	Returns the number of rows in a matrix.
Format	<code>y = rows(x);</code>
Input	<code>x</code> NxK matrix.
Output	<code>y</code> scalar, number of rows in the specified matrix.
Remarks	If <code>x</code> is an empty matrix, rows (<code>x</code>) and cols (<code>x</code>) return 0.
Example	<pre> x = ones(3,5); y = rows(x); 1 1 1 1 1 x = 1 1 1 1 1 1 1 1 1 1 y = 3 </pre>
See also	cols , show

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

row**sf**

row**sf**

Purpose	Returns the number of rows in a GAUSS data set (.dat) file or GAUSS matrix (.fmt) file.
Format	$y = \text{row}\text{sf}(f);$
Input	f file handle of an open file.
Output	y scalar, number of rows in the specified file.
Example	<pre>open fp = myfile; r = rowsf(fp); c = colsf(fp);</pre>
See also	colsf, open, typef

rref

Purpose	Computes the reduced row echelon form of a matrix.
Format	<code>y = rref(x);</code>
Input	<i>x</i> MxN matrix.
Output	<i>y</i> MxN matrix containing reduced row echelon form of <i>x</i> .
Remarks	<p>The tolerance used for zeroing elements is computed inside the procedure using:</p> <pre>tol = maxc(m n) * eps * maxc(abs(sumc(x')));</pre> <p>where eps = 2.24e-16;</p> <p>This procedure can be used to find the rank of a matrix. It is not as stable numerically as the singular value decomposition (which is used in the rank function), but it is faster for large matrices.</p> <p>There is some speed advantage in having the number of rows be greater than the number of columns, so you may want to transpose if all you care about is the rank.</p> <p>The following code can be used to compute the rank of a matrix:</p> <pre>r = sumc(sumc(abs(y')) .> tol);</pre> <p>where <i>y</i> is the output from rref, and tol is the tolerance used. This finds the number of rows with any nonzero elements, which gives the rank of the matrix, disregarding numeric problems.</p>
Example	<pre>let x[3,3] = 1 2 3 4 5 6 7 8 9; y = rref(x); 1 0 -1 y = 0 1 2 0 0 0</pre>
Source	<code>rref.src</code>

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

run

run

a		
b	Purpose	Runs a source code or compiled code program.
c	Format	run <i>filename</i> ;
d	Input	<i>filename</i> literal or ^string, name of file to run.
e	Remarks	<p>The filename can be any legal file name. Filename extensions can be whatever you want, except for the compiled file extension, .gcg. Pathnames are okay. If the name is to be taken from a string variable, then the name of the string variable must be preceded by the ^ (caret) operator.</p> <p>The run statement can be used both from the command line and within a program. If used in a program, once control is given to another program through the run statement there is no return to the original program.</p> <p>If you specify a filename without an extension, GAUSS will first look for a compiled code program (i.e., a .gcg file) by that name, then a source code program by that name. For example, if you enter</p> <pre>run dog;</pre> <p>GAUSS will first look for the compiled code file dog.gcg, and run that if it finds it. If GAUSS cannot find dog.gcg, it will then look for the source code file dog with no extension.</p> <p>If a path is specified for the file, then no additional searching will be attempted if the file is not found.</p> <p>If a path is not specified the current directory will be searched first, then each directory listed in src_path. The first instance found is run. src_path is defined in gauss.cfg.</p> <pre>run /gauss/myprog.prc;</pre> <p>No additional search will be made if the file is not found.</p> <pre>run myprog.prc;</pre> <p>The directories listed in src_path will be searched for myprog.prc if the file is not found in the current directory.</p> <p>Programs can also be run by typing the filename on the OS command line when starting GAUSS.</p>
f		
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z	Example	<p>Example 1</p> <pre>run myprog.prc;</pre>

run

```
Example 2
  name = "myprog.prg";
  run ^name;
```

See also **#include**

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

save

save

a		
b	Purpose	Saves matrices, strings, or procedures to a disk file.
c	Format	save <i>[[vflag]]</i> <i>[[path=path]]</i> <i>x</i> , <i>[[lpath=]]y</i> ;
d	Input	<i>vflag</i> version flag.
e		-v89 supported on DOS, OS/2, Windows
f		-v92 supported on UNIX, Windows
g		-v96 supported on all platforms
h		See also “File I/O” in the <i>User’s Guide</i> for details on the various versions. The default format can be specified in <i>gauss.cfg</i> by setting the dat_fmt_version configuration variable. If dat_fmt_version is not set, the default is v96 .
i		<i>path</i> literal or ^string, a default path to use for this and subsequent saves .
j		<i>x</i> a symbol name, the name of the file the symbol will be saved in is the same as this with the proper extension added for the type of the symbol.
k		<i>lpath</i> literal or ^string, a local path and filename to be used for a particular symbol. This path will override the path previously set and the filename will override the name of the symbol being saved. The extension cannot be overridden.
l		<i>y</i> the symbol to be saved to <i>lpath</i> .
m	Remarks	save can be used to save matrices, strings, procedures, and functions. Procedures and functions must be compiled and resident in memory before they can be save ’d.
n		The following extensions will be given to files that are saved:
o		matrix .fmt
p		string .fst
q		procedure .fcg
r		function .fcg
s		keyword .fcg
t		if the path= subcommand is used with save , the path string will be remembered until changed in a subsequent command. This path will be
u		
v		
w		
x y z		

used whenever none is specified. The save path can be overridden in any particular save by specifying an explicit path and filename.

Example

```
spath = "/gauss";  
save path = ^spath x,y,z;
```

Save **x**, **y**, and **z** using **/gauss** as a path. This path will be used for the next save if none is specified.

```
svp = "/gauss/data";  
save path = ^svp n, k, /gauss/quad1=quad;
```

n and **k** will be saved using **/gauss/data** as the save path, **quad** will be saved in **/gauss** with the name **quad1.fmt**. On platforms that use the backslash as the path separator, the double backslash is required inside double quotes to get a backslash, because it is the escape character in quoted strings. It is not required when specifying literals.

```
save path=/procs;
```

Changes save path to /procs.

```
save path = /miscdata;  
save /data/mydata1 = x, y, hisdata = z;
```

In the above program:

x would be saved in **/data/mydata1.fmt**

y would be saved in **/miscdata/y.fmt**

z would be saved in **/miscdata/hisdata.fmt**

See also **load**, **saveall**, **saved**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

saveall

saveall

a	Purpose	Saves the current state of the machine to a compiled file. All procedures, global matrices and strings will be saved.
b		
c	Format	saveall <i>fname</i> ;
d	Input	<i>fname</i> literal or ^string, the path and filename of the compiled file to be created.
e		
f	Remarks	The file extension will be .gcg.
g		A file will be created containing all your matrices, strings, and procedures. No main code segment will be saved. This just means it will be a .gcg file with no main program code (see compile). The rest of the contents of memory will be saved including all global matrices, strings, functions and procedures. Local variables are not saved. This can be used inside a program to take a snapshot of the state of your global variables and procedures. To reload the compiled image use run or use .
h		library pgraph;
i		external proc xy,logx,logy,loglog,hist;
j		saveall pgraph;
k		
l		This would create a file called pgraph.gcg containing all the procedures, strings and matrices needed to run Publication Quality Graphics programs. Other programs could be compiled very quickly with the following statement at the top of each:
m		use pgraph;
n		
o	See also	compile, run, use
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

saved

Purpose	Writes a matrix in memory to a GAUSS data set on disk.
Format	<code>y = saved(x,dataset,vnames) ;</code>
Input	<p><i>x</i> NxK matrix to save in .dat file.</p> <p><i>dataset</i> string, name of data set.</p> <p><i>vnames</i> string or Kx1 character vector, names for the columns of the data set.</p>
Output	<i>y</i> scalar, 1 if successful, 0 if fail.
Remarks	<p>If <i>dataset</i> is null or 0, the data set name will be temp.dat.</p> <p>if <i>vnames</i> is a null or 0, the variable names will begin with “X” and be numbered 1-K.</p> <p>If <i>vnames</i> is a string or has fewer elements than <i>x</i> has columns, it will be expanded as explained under create.</p> <p>The output data type is double precision.</p>
Example	<pre> x = rndn(100,3); dataset = "mydata"; vnames = {height, weight, age }; if not saved(x,dataset,vnames); errorlog "Write error"; end; endif; </pre>
Source	saveload.src
See also	loadadd, writer, create

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

savewind

savewind

a	Purpose	Saves the current graphic panel configuration to a file.
b	Library	pgraph
c	Format	<i>err</i> = savewind (<i>filename</i>);
d	Input	<i>filename</i> Name of file.
e	Output	<i>err</i> scalar, 0 if successful, 1 if graphic panel matrix is invalid. Note that the file is written in either case.
f	Remarks	See the discussion on using graphic panels in “Publication Quality Graphics” in the <i>User’s Guide</i> .
g	Source	pwindow.src
h	See also	loadwind
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

scale

Purpose	Fixes the scaling for subsequent graphs. The axes endpoints and increments are computed as a best guess based on the data passed to it.
Library	pgraph
Format	scale (<i>x</i> , <i>y</i>) ;
Input	<i>x</i> matrix, the X axis data. <i>y</i> matrix, the Y axis data.
Remarks	<p><i>x</i> and <i>y</i> must each have at least 2 elements. Only the minimum and maximum values are necessary.</p> <p>This routine fixes the scaling for all subsequent graphs until graphset is called. This also clears xtics and ytics whenever it is called.</p> <p>If either of the arguments is a scalar missing, the main graphics function will set the scaling for that axis using the actual data.</p> <p>If an argument has 2 elements, the first will be used for the minimum and the last will be used for the maximum.</p> <p>If an argument has 2 elements, and contains a missing value, that end of the axis will be scaled from the data by the main graphics function.</p> <p>If you want direct control over the axes endpoints and tick marks, use xtics or ytics. If xtics or ytics have been called after scale, they will override scale.</p>
Source	pscale.src
See also	xtics , ytics , ztics , scale3d

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

scale3d

scale3d

a	Purpose	Fixes the scaling for subsequent graphs. The axes endpoints and increments are computed as a best guess based on the data passed to it.
b		
c	Library	pgraph
d	Format	scale3d (<i>x,y,z</i>) ;
e	Input	<i>x</i> matrix, the X axis data.
f		<i>y</i> matrix, the Y axis data.
g		<i>z</i> matrix, the Z axis data.
h	Remarks	<i>x</i> , <i>y</i> and <i>z</i> must each have at least 2 elements. Only the minimum and maximum values are necessary.
i		This routine fixes the scaling for all subsequent graphs until graphset is called. This also clears xtics , ytics and ztics whenever it is called.
j		If any of the arguments is a scalar missing, the main graphics function will set the scaling for that axis using the actual data.
k		If an argument has 2 elements, the first will be used for the minimum and the last will be used for the maximum.
l		If an argument has 2 elements, and contains a missing value, that end of the axis will be scaled from the data by the main graphics function.
m		If you want direct control over the axes endpoints and tick marks, use xtics , ytics , or ztics . If one of these functions have been called, they will override scale3d .
n	Source	pscale.src
o	See also	scale , xtics , ytics , ztics
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

scalerr

Purpose	Tests for a scalar error code.	
Format	$y = \text{scalerr}(c);$	
Input	c	NxK matrix, generally the return argument of a function or procedure call.
Output	y	scalar, which is returned as a 0 if its argument is not a scalar error code. If the argument is an error code, then scalerr returns the value of the error code as an integer.
Remarks	<p>Error codes in GAUSS are NaN's (Not A Number). These are not just scalar integer values. They are special floating point encodings that the math chip recognizes as not representing a valid number. See also error.</p> <p>scalerr can be used to test for either those error codes which are predefined in GAUSS or an error code which the user has defined using error.</p> <p>If c is an empty matrix, scalerr will return 65535.</p> <p>Certain functions will either return an error code or terminate a program with an error message, depending on the trap state. The trap command is used to set the trap state. The error code that will be returned will appear to most commands as a missing value code, but the scalerr function can distinguish between missing values and error codes and will return the value of the error code.</p>	

Following are some of the functions affected by the trap state:

function	trap 1 error code	trap 0 error message
chol	10	Matrix not positive definite
invpd	20	Matrix not positive definite
solpd	30	Matrix not positive definite
/	40	Matrix not positive definite (second argument not square)
	41	Matrix singular (second argument is square)
inv	50	Matrix singular

scalerr

Example

```
trap 1;
cm = invpd(x);
trap 0;
if scalerr(cm);
    cm = inv(x);
endif;
```

In this example **invpd** will return a scalar error code if the matrix **x** is not positive definite. If **scalerr** returns with a nonzero value, the program will use the **inv** function, which is slower, to compute the inverse. Since the trap state has been turned off, if **inv** fails the program will terminate with a **Matrix singular** error message.

See also **error, trap, trapchk**

scalinfnanmiss

Purpose Returns true if the argument is a scalar infinity, NaN, or missing value.

Format `y = scalinfnanmiss(x);`

Input `x` NxK matrix.

Output `y` scalar, 1 if `x` is a scalar, infinity, NaN, or missing value, else 0.

See Also **isinfnanmiss**, **ismiss**, **scalmiss**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

scalmiss**scalmiss**

Purpose Tests to see if its argument is a scalar missing value.

Format `y = scalmiss(x);`

Input `x` N x K matrix.

Output `y` scalar, 1 if argument is a scalar missing value, 0 if not.

Remarks **scalmiss** first tests to see if the argument is a scalar. If it is not scalar, **scalmiss** returns a 0 without testing any of the elements.

The **ismiss** function will test each element of the matrix and return 1 if it encounters any missing values. **scalmiss** will execute much faster if the argument is a large matrix since it will not test each element of the matrix but will simply return a 0.

An element of `x` is considered to be a missing if and only if it contains a missing value in the real part. Thus, **scalmiss** and **ismiss** would return a 1 for complex `x = . + 1i`, a 0 for `x = 1 + .i`.

Example

```
clear s;

do until eof(fp);
    y = readr(fp,nr);
    y = packr(y);
    if scalmiss(y);
        continue;
    endif;
    s = s+sumc(y);
enddo;
```

In this example the **packr** function will return a scalar missing if every row of its argument contains missing values, otherwise it will return a matrix that contains no missing values. **scalmiss** is used here to test for a scalar missing returned from **packr**. If that is true, then the sum step will be skipped for that iteration of the read loop because there were no rows left after the rows containing missings were packed out.

schtoc

Purpose	To reduce any 2x2 blocks on the diagonal of the real Schur matrix returned from schur . The transformation matrix is also updated.	
Format	<code>{ schc, transc } = schtoc(sch,trans);</code>	
Input	<i>sch</i>	real NxN matrix in Real Schur form, i.e., upper triangular except for possibly 2x2 blocks on the diagonal.
	<i>trans</i>	real NxN matrix, the associated transformation matrix.
Output	<i>schc</i>	NxN matrix, possibly complex, strictly upper triangular. The diagonal entries are the eigenvalues.
	<i>transc</i>	NxN matrix, possibly complex, the associated transformation matrix.
Remarks	Other than checking that the inputs are strictly real matrices, no other checks are made. If the input matrix <i>sch</i> is already upper triangular it is not changed. Small off-diagonal elements are considered to be zero. See the source code for the test used.	
Example	<code>{ schc, transc } = schtoc(schur(a));</code>	
	This example calculates the complex Schur form for a real matrix a .	
Source	<code>schtoc.src</code>	
See also	schur	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

schur

schur

a	Purpose	Computes the Schur form of a square matrix.
b	Format	$\{ s, z \} = \text{schur}(x)$
c	Input	x KxK matrix.
d	Output	s KxK matrix, Schur form.
e		z KxK matrix, transformation matrix.
f	Remarks	<p>schur computes the real Schur form of a square matrix. The real Schur form is an upper quasi-triangular matrix, that is, it is block triangular where the blocks are 2x2 submatrices which correspond to complex eigenvalues of x. If x has no complex eigenvalues, s will be strictly upper triangular. To convert s to the complex Schur form, use the Run-Time Library function schtoc.</p> <p>x is first reduced to upper Hessenberg form using orthogonal similarity transformations, then reduced to Schur form through a sequence of QR decompositions.</p> <p>schur uses the ORTRAN, ORTHES and HQR2 functions from EISPACK.</p> <p>z is an orthogonal matrix that transforms x into s and vice versa. Thus</p> $s = z'xz$ <p>and since z is orthogonal,</p> $x = zs z'$
g	Example	<pre>let x[3,3] = 1 2 3 4 5 6 7 8 9; { s, z } = schur(x);</pre> $s = \begin{bmatrix} 16.11684397 & 4.89897949 & 0.00000000 \\ -0.00000000 & -1.11684397 & -0.00000000 \\ 0.00000000 & 0.00000000 & 0.00000000 \end{bmatrix}$
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

schur

$$z = \begin{pmatrix} 0.23197069 & 0.88290596 & 0.40824829 \\ 0.52532209 & 0.23952042 & -0.81649658 \\ 0.81867350 & -0.40386512 & 0.40824829 \end{pmatrix}$$

See also **hess**

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

screen

screen

a		
b	Purpose	Controls output to the screen.
c	Format	screen on; screen off; screen;
d		
e		
f	Remarks	When this is on , the results of all print statements will be directed to the window. When this is off , print statements will not be sent to the window. This is independent of the statement output on , which will cause the results of all print statements to be routed to the current auxiliary output file.
g		
h		If you are sending a lot of output to the auxiliary output file on a disk drive, turning the window off will speed things up.
i		
j		The end statement will automatically do output off and screen on .
k		screen with no arguments will print “ Screen is on ” or “ Screen is off ” on the console.
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

screen

Example `output file = mydata.asc reset;`
 `screen off;`
 `format /m1/rz 1,8;`
 `open fp = mydata;`
 `do until eof(fp);`
 `print readr(fp,200);;`
 `endo;`
 `fp = close(fp);`
 `end;`

The program above will write the contents of the GAUSS file `mydata.dat` into an ASCII file called `mydata.asc`. If `mydata.asc` already exists, it will be overwritten.

Turning the window off will speed up execution. The **end** statement above will automatically perform **output off** and **screen on**.

See also **output, end, new**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

scroll

scroll

Purpose	Scrolls a section of the window.																		
Format	scroll <i>v</i> ;																		
Input	<i>v</i> 6x1 vector																		
Portability	Windows																		
Remarks	<p>This command is intended to be used in the DOS compatibility window to support legacy programs.</p> <p>The elements of <i>v</i> are defined as:</p> <table><tr><td>[1]</td><td>coordinate of upper left row.</td></tr><tr><td>[2]</td><td>coordinate of upper left column.</td></tr><tr><td>[3]</td><td>coordinate of lower right row.</td></tr><tr><td>[4]</td><td>coordinate of lower right column.</td></tr><tr><td>[5]</td><td>number of lines to scroll.</td></tr><tr><td>[6]</td><td>value of attribute.</td></tr></table> <p>This assumes the origin at (1,1) in the upper left just like the locate command. The window will be scrolled the number of lines up or down (positive or negative 5th element) and the value of the 6th element will be used as the attribute as follows:</p> <table><tr><td>7</td><td>regular text</td></tr><tr><td>112</td><td>reverse video</td></tr><tr><td>0</td><td>graphics black</td></tr></table>	[1]	coordinate of upper left row.	[2]	coordinate of upper left column.	[3]	coordinate of lower right row.	[4]	coordinate of lower right column.	[5]	number of lines to scroll.	[6]	value of attribute.	7	regular text	112	reverse video	0	graphics black
[1]	coordinate of upper left row.																		
[2]	coordinate of upper left column.																		
[3]	coordinate of lower right row.																		
[4]	coordinate of lower right column.																		
[5]	number of lines to scroll.																		
[6]	value of attribute.																		
7	regular text																		
112	reverse video																		
0	graphics black																		

If the number of lines (element 5) is 0, the entire window will be blanked.

scroll

Example `let v = 1 1 12 80 5 7;`
 `scroll v;`

This call would scroll a graphic panel 80 columns wide covering the upper twelve rows of the window. The graphic panel would be scrolled up 5 lines and the new lines would be displayed in regular text mode.

See also **locate, printdos**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

seekr

seekr

a	Purpose	Moves the pointer in a .dat or .fmt file to a particular row.
b	Format	$y = \text{seekr}(fh,r);$
c	Input	fh scalar, file handle of an open file. r scalar, the row number to which the pointer is to be moved.
d	Output	y scalar, the row number to which the pointer has been moved.
e	Remarks	If $r = -1$, the current row number will be returned. If $r = 0$, the pointer will be moved to the end of the file, just past the end of the last row. rowsf returns the number of rows in a file. $\text{seekr}(fh,0) == \text{rowsf}(fh) + 1;$ Do NOT try to seek beyond the end of a file.
f	See also	open, readr, rowsf
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

select (dataloop)

Purpose	Selects specific rows (observations) in a data loop based on a logical expression.
Format	select <i>logical_expression</i> ;
Remarks	Selects only those rows for which <i>logical_expression</i> is <i>TRUE</i> . Any variables referenced must already exist, either as elements of the source data set, as externs , or as the result of a previous make , vector , or code statement.
Example	<code>select age > 40 AND sex \$== 'MALE' ;</code>
See also	delete

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

selif

selif

Purpose	Selects rows from a matrix. Those selected are the rows for which there is a 1 in the corresponding row of <i>e</i> .		
Format	<i>y</i> = selif (<i>x</i> , <i>e</i>);		
Input	<i>x</i>	NxK matrix.	
	<i>e</i>	Nx1 vector of 1's and 0's.	
Output	<i>y</i>	MxK matrix consisting of the rows of <i>x</i> for which there is a 1 in the corresponding row of <i>e</i> .	
Remarks	The argument <i>e</i> will usually be generated by a logical expression using “dot” operators. <i>y</i> will be a scalar missing if no rows are selected.		
Example	<pre>y = selif(x,x[.,2] .gt 100);</pre> <p>selects all rows of <i>x</i> in which the second column is greater than 100.</p> <pre>let x[3,3] = 0 10 20 30 40 50 60 70 80;</pre> <pre>e = (x[.,1] .gt 0) .and (x[.,3] .lt 100);</pre> <pre>y = selif(x,e);</pre> <p>The resulting matrix <i>y</i> is:</p> <pre>30 40 50 60 70 80</pre> <p>All rows for which the element in column 1 is greater than 0 and the element in column 3 is less than 100 are placed into the matrix y.</p>		
Source	datatran.src		
See also	delif , scalmiss		

seqa, seqm

Purpose	seqa creates an additive sequence. seqm creates a multiplicative sequence.
Format	<pre>y = seqa(start,inc,n) ; y = seqm(start,inc,n) ;</pre>
Input	<p><i>start</i> scalar specifying the first element.</p> <p><i>inc</i> scalar specifying increment.</p> <p><i>n</i> scalar specifying the number of elements in the sequence.</p>
Output	<i>y</i> Nx1 vector containing the specified sequence.
Remarks	<p>For seqa, <i>y</i> will contain a first element equal to <i>start</i>, the second equal to <i>start+inc</i>, and the last equal to <i>start+inc*(n-1)</i>.</p> <p>For instance,</p> <pre>seqa(1,1,10)</pre> <p>will create a column vector containing the numbers 1, 2, ... 10.</p> <p>For seqm, <i>y</i> will contain a first element equal to <i>start</i>, the second equal to <i>start*inc</i>, and the last equal to <i>start*inc⁽ⁿ⁻¹⁾</i>.</p> <p>For instance,</p> <pre>seqm(10,10,10)</pre> <p>will create a column vector containing the numbers 10, 100, ... 10¹⁰.</p>
Example	<pre>a = seqa(2,2,10)' ; m = seqm(2,2,10)' ;</pre> <pre>a = 2 4 6 8 10 12 14 16 18 20 m = 2 4 8 16 32 64 128 256 512 1024</pre> <p>Note that the results have been transposed in this example. Both functions return Nx1 (column) vectors.</p>
See also	recserar, recsercp

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

setdif

setdif

Purpose	Returns the unique elements in one vector that are not present in a second vector.		
Format	$y = \text{setdif}(v1,v2,flag);$		
Input	$v1$	Nx1 vector.	
	$v2$	Mx1 vector.	
	$flag$	scalar,	if 0, case-sensitive character comparison.
			if 1, numeric comparison.
			if 2, case-insensitive character comparison.
Output	y	Lx1 sorted vector containing all unique values that are in $v1$ and are not in $v2$, or a scalar missing.	
Example	<pre>let v1 = mary jane linda john; let v2 = mary sally; flag = 0; y = setdif(v1,v2,flag); JANE y = JOHN LINDA</pre>		
Source	setdif.src		

setvars

Purpose	Reads the variable names from a data set header and creates global matrices with the same names.	
Format	<i>nvec</i> = setvars (<i>dataset</i>);	
Input	<i>dataset</i>	string, the name of the GAUSS data set. Do not use a file extension.
Output	<i>nvec</i>	Nx1 character vector, containing the variable names defined in the data set.
Remarks	setvars is designed to be used interactively.	
Example	<i>nvec</i> = setvars("freq");	
Source	vars.src	
See also	makevars	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

setvwrmode

setvwrmode

a	Purpose	Sets the graphics viewer mode.
b	Library	pgraph
c	Format	<i>oldmode</i> = setvwrmode(mode) ;
d	Input	<i>mode</i> string, new mode or null string.
e		"one" Use only one viewer.
f		"many" Use a new viewer for each graph.
g	Output	<i>oldmode</i> string, previous mode.
h	Remarks	If mode is a null string, the current mode will be returned with no changes made.
i		If "one" is set, the viewer executable will be vwr.exe.
j	Example	<i>oldmode</i> = setvwrmode("one"); call setvwrmode(<i>oldmode</i>);
k	Source	pgraph.src
l	See also	pggwin
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

setwind

Purpose	Sets the current graphic panel to a previously created graphic panel number.
Library	pgraph
Format	setwind(<i>n</i>);
Input	<i>n</i> scalar, graphic panel number.
Remarks	<p>This function selects the specified graphic panel to be the current graphic panel. This is the graphic panel in which the next graph will be drawn.</p> <p>See the discussion on using graphic panels in “Publication Quality Graphics in the <i>User’s Guide</i>.”</p>
Source	pwindow.src
See also	begwind, endwind, getwind, nextwind, makewind, window

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

shell

shell

a	Purpose	Executes an operating system command.
b	Format	shell <code>[[s]];</code>
c	Input	<i>s</i> literal or ^string, the command to be executed.
d	Remarks	shell lets you run shell commands and programs from inside GAUSS. If a command is specified, it is executed; when it finishes, you automatically return to GAUSS. If no command is specified, the shell is executed and control passes to it, so you can issue commands interactively. You have to type exit to get back to GAUSS in that case. If you specify a command in a string variable, precede it with the (caret) ^ .
e	Example	<code>comstr = "ls ./src";</code> <code>shell ^comstr;</code> This lists the contents of the <code>./src</code> subdirectory, then returns to GAUSS. <code>shell cmp n1.fmt n1.fmt.old;</code> This compares the matrix file <code>n1.fmt</code> to an older version of itself, <code>n1.fmt.old</code> , to see if it has changed. When cmp finishes, control is returned to GAUSS. <code>shell;</code> This executes an interactive shell. The OS prompt will appear and OS commands or other programs can be executed. To return to GAUSS, type exit .
f	See also	exec
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

shiftr

Purpose	Shifts the rows of a matrix.		
Format	$y = \text{shiftr}(x,s,f);$		
Input	x	NxK matrix to be shifted.	
	s	scalar or Nx1 vector specifying the amount of shift.	
	f	scalar or Nx1 vector specifying the value to fill in.	
Output	y	NxK shifted matrix.	
Remarks	The shift is performed within each row of the matrix, horizontally. If the shift value is positive, the elements in the row will be moved to the right. A negative shift value causes the elements to be moved to the left. The elements that are pushed off the end of the row are lost, and the fill value will be used for the new elements on the other end.		

Example $y = \text{shiftr}(x,s,f);$

If $x = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ and $s = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$ and $f = \begin{bmatrix} 99 \\ 999 \end{bmatrix}$

Then $y = \begin{bmatrix} 99 & 1 \\ 4 & 999 \end{bmatrix}$

If $x = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$ and $s = \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix}$ and $f = 0$

Then $y = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 4 & 5 \\ 0 & 0 & 7 \end{bmatrix}$

See also **rotater**

show, lshow

show, lshow

Purpose	Displays the global symbol table. The output from lshow is sent to the printer.				
Format	show <i>[-flags]</i> <i>[[symbol]]</i> ; lshow <i>[-flags]</i> <i>[[symbol]]</i> ;				
Input	<i>flags</i>	flags to specify the symbol type that is shown.			
	k	keywords			
	p	procedures			
	f	fn functions			
	m	matrices			
	s	strings			
	g	show only symbols with global references			
	l	show only symbols with all local references			
	n	no pause			
	<i>symbol</i>	the name of the symbol to be shown. If the last character is an asterisk (*), all symbols beginning with the supplied characters will be shown.			
Remarks	If there are no arguments, the entire symbol table will be displayed. show is directed to the auxiliary output if it is open. Here is an example listing with an explanation of the columns:				
	Memory used	Address	Name	Info	Cplx Type References
	32 bytes at [00081b74]		AREA	1=1	FUNCTION local refs
	32 bytes at [00081a14]		dotfeq	1=2	PROCEDURE global refs
	1144 bytes at [0007f1b4]		indices2	4=3	PROCEDURE local refs
	144 bytes at [0007f874]		X	3,3	C MATRIX
	352 bytes at [0007f6ec]		_IXCAT	44,1	MATRIX
	8 bytes at [0007f6dc]		_olsrnam	7 char	STRING
	32000 bytes program space, 0% used 4336375 bytes workspace, 4325655 bytes free 53 global symbols, 1500 maximum, 6 shown				

show, lshow

The “Memory used” column is the amount of memory used by the item.

The “Address” column is the address where the item is stored (hexadecimal format). This will change as matrices and strings change in size.

The “Name” column is the name of the symbol.

The “Info” column depends on the type of the symbol. If the symbol is a procedure or a function, it gives the number of values that the function or procedure returns and the number of arguments that need to be passed to it when it is called. If the symbol is a matrix, then the Info column gives the number of rows and columns. If the symbol is a string, then the Info column gives the number of characters in the string. As follows:

Rets=Args	if procedure or function
Row,Col	if matrix
Length	if string

The “Cplx” column contains a “C” if the symbol is a complex matrix.

The “Type” column specifies the symbol table type of the symbol. It can be **function**, **keyword**, **matrix**, **procedure**, or **string**.

If the symbol is a procedure, keyword or function, the “References” column will show if it makes any global references. If it makes only local references, the procedure or function can be saved to disk in an `.fcg` file with the **save** command. If the function or procedure makes any global references, it cannot be saved in an `.fcg` file.

The program space is the area of space reserved for all nonprocedure, nonfunction program code. It can be changed in size with the **new** command. The workspace is the memory used to store matrices, strings, procedures, and functions.

Example `show /fpg eig*;`

This command will show all functions and procedures that have global references and begin with **eig**.

`show /mn;`

This command will show all matrices without pausing when the window is full.

See also **new, delete**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

sin

sin

a	Purpose	Returns the sine of its argument.
b	Format	$y = \text{sin}(x);$
c	Input	x NxK matrix.
d	Output	y NxK matrix containing sine of x .
e	Remarks	For real matrices, x should contain angles measured in radians.
f		To convert degrees to radians, multiply the degrees by $\frac{\pi}{180}$.
g	Example	$\text{let } x = \{ 0, .5, 1, 1.5 \};$
h		$y = \text{sin}(x);$
i		0.00000000
j		$y = 0.47942554$
k		0.84147098
l		0.99749499
m	See also	atan, cos, sinh, pi
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

sinh

Purpose Computes the hyperbolic sine.

Format `y = sinh(x);`

Input `x` NxK matrix.

Output `y` NxK matrix containing the hyperbolic sines of the elements of `x`.

Example `let x = { -0.5, -0.25, 0, 0.25, 0.5, 1 };`

`x = x * pi;`

`y = sinh(x);`

`x =`
 -1.570796
 -0.785398
 0.000000
 0.785398
 1.570796
 3.141593

`y =`
 -2.301299
 -0.868671
 0.000000
 0.868671
 2.301299
 11.548739

Source `trig.src`

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

sleep

sleep

a		
b		
c		
d		
e		
f		
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

Purpose Sleeps for a specified number of seconds.

Format *unslept* = **sleep**(*secs*);

Input *secs* scalar, number of seconds to sleep.

Output *unslept* scalar, number of seconds not slept.

Remarks *secs* does not have to be an integer. If your system does not permit sleeping for a fractional number of seconds, *secs* will be rounded to the nearest integer, with a minimum value of 1.

 If a program sleeps for the full number of secs specified, **sleep** returns 0; otherwise, if the program is awakened early (e.g., by a signal), **sleep** returns the amount of time not slept. The DOS version always sleeps the full number of seconds, so it always returns 0.

 A program may sleep for longer than *secs* seconds, due to system scheduling.

solpd

Purpose	Solves a set of positive definite linear equations.	
Format	$x = \text{solpd}(b,A);$	
Input	b	NxK matrix.
	A	NxN symmetric positive definite matrix.
Output	x	NxK matrix, the solutions for the system of equations, $Ax=b$.
Remarks	<p>b can have more than one column. If so, the system of equations is solved for each column, i.e., $A*x[:,i] = b[:,i]$.</p> <p>This function uses the Cholesky decomposition to solve the system directly. Therefore it is more efficient than using <code>inv(A)*b</code>.</p> <p>solpd does not check to see that the matrix A is symmetric. solpd will look only at the upper half of the matrix including the principal diagonal.</p> <p>If the A matrix is not positive definite:</p> <pre> trap 1 return scalar error code 30. trap 0 terminate with an error message. </pre> <p>One obvious use for this function is to solve for least squares coefficients. The effect of this function is thus similar to that of the <code>/</code> operator.</p> <p>If X is a matrix of independent variables, and Y is a vector containing the dependent variable, then the following code will compute the least squares coefficients of the regression of Y on X:</p> <pre> b = solpd(X'Y,X'X); </pre>	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

solpd

Example

```
n = 5; format 20,8;
A = rndn(n,n);
A = A'A;
x = rndn(n,1);
b = A*x;
x2 = solpd(b,A);
print "      X      solpd(b,A)      Difference";
print x~x2~x-x2;
Produces:
           X      solpd(b,A)      Difference
-0.36334089    -0.36334089    0.00000000
 0.19683330     0.19683330    8.32667268E-017
 0.99361330     0.99361330    2.22044605E-016
-1.84167681    -1.84167681    0.00000000
-0.88455829    -0.88455829    1.11022302E-016
```

See also **scalerr, chol, invpd, trap**

sortc, sortcc

Purpose	Sorts a matrix of numeric or character data.
Format	<pre>y = sortc(x,c); y = sortcc(x,c);</pre>
Input	<p><i>x</i> NxK matrix.</p> <p><i>c</i> scalar specifying one column of <i>x</i> to sort on.</p>
Output	<i>y</i> NxK matrix equal to <i>x</i> and sorted on the column <i>c</i> .
Remarks	<p>These functions will sort the rows of a matrix with respect to a specified column. That is, they will sort the elements of a column and will arrange all rows of the matrix in the same order as the sorted column.</p> <p>sortc assumes the column to sort on is numeric. sortcc assumes that the column to sort on contains character data.</p> <p>The matrix may contain both character and numeric data, but the sort column must be all of one type. Missing values will sort as if their value is below $-\infty$.</p> <p>The sort will be in ascending order. This function uses the Quicksort algorithm.</p> <p>If you need to obtain the matrix sorted in descending order, you can use:</p> <pre>rev(sortc(x,c))</pre> <p>Example</p> <pre>let x[3,3]= 4 7 3 1 3 2 3 4 8; y = sortc(x,1); 4 7 3 x = 1 3 2 3 4 8</pre>

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

sortc, sortcc

$$y = \begin{matrix} & 1 & 3 & 2 \\ 3 & 4 & 8 \\ 4 & 7 & 3 \end{matrix}$$

See also **rev**

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

sortd

Purpose	Sorts a data file on disk with respect to a specified variable.
Format	sortd (<i>infile</i> , <i>outfile</i> , <i>keyvar</i> , <i>keytyp</i>) ;
Input	<i>infile</i> string, name of input file. <i>outfile</i> string, name of output file, must be different. <i>keyvar</i> string, name of key variable. <i>keytyp</i> scalar, type of key variable. 1 numeric key, ascending order. 2 character key, ascending order. -1 numeric key, descending order. -2 character key, descending order.
Remarks	The data set <i>infile</i> will be sorted on the variable <i>keyvar</i> , and will be placed in <i>outfile</i> . If the inputs are null or 0, the procedure will ask for them.
Source	sortd.src

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

sorthc, sorthcc

sorthc, sorthcc

a	
b	
c	
d	
e	
f	
g	
h	
i	
j	
k	
l	
m	
n	
o	
p	
q	
r	
s	
t	
u	
v	
w	
x y z	

Purpose	Sorts a matrix of numeric or character data.
Format	$y = \text{sorthc}(x,c);$ $y = \text{sorthcc}(x,c);$
Input	x NxK matrix. c scalar specifying one column of x to sort on.
Output	y NxK matrix equal to x and sorted on the column c .
Remarks	<p>These functions will sort the rows of a matrix with respect to a specified column. That is, they will sort the elements of a column and will arrange all rows of the matrix in the same order as the sorted column.</p> <p>sorthc assumes that the column to sort on is numeric. sorthcc assumes that the column to sort on contains character data.</p> <p>The matrix may contain both character and numeric data, but the sort column must be all of one type. Missing values will sort as if their value is below $-\infty$.</p> <p>The sort is in ascending order. This function uses the heap sort algorithm.</p> <p>If you need to obtain the matrix sorted in descending order, you can use:</p> <p style="text-align: center;">rev(sorthc(x,c))</p>
Example	<pre>let x[3,3]= 4 7 3 1 3 2 3 4 8; y = sorthc(x,1); 4 7 3 x = 1 3 2 3 4 8</pre>

sorthc, sorthcc

```
      1 3 2
y =   3 4 8
      4 7 3
```

See also **sortc, rev**

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

sortind, sortindc

sortind, sortindc

a	Purpose	Returns the sorted index of x .
b	Format	$ind = \text{sortind}(x);$ $ind = \text{sortindc}(x);$
c	Input	x Nx1 column vector.
d	Output	ind Nx1 vector representing sorted index of x .
e	Remarks	sortind assumes x contains numeric data. sortindc assumes x contains character data. This function can be used to sort several matrices in the same way that some other reference matrix is sorted. To do this, create the index of the reference matrix, then use submat to rearrange the other matrices in the same way.
f	Example	let $x = 5 \ 4 \ 4 \ 3 \ 3 \ 2 \ 1;$ $ind = \text{sortind}(x);$ $y = x[ind];$
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

sortmc

Purpose	Sorts a matrix on multiple columns.	
Format	$y = \text{sortmc}(x, v);$	
Input	x	NxK matrix to be sorted.
	v	Lx1 vector containing integers specifying the columns, in order, that are to be sorted. If an element is negative, that column will be interpreted as character data.
Output	y	NxK sorted matrix.
Remarks	The function works recursively and the number of sort columns is limited by the available workspace.	
Source	sortmc.src	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

sparseCols

sparseCols

Purpose	Returns the number of columns in a sparse matrix.	
Format	$c = \text{sparseCols}(x);$	
Input	x	MxN sparse matrix.
Output	c	scalar, number of columns.
Source	sparse.src	

sparseEye

Purpose	Returns a sparse identity matrix.	
Format	$y = \text{sparseEye}(n);$	
Input	n	scalar, order of identity matrix.
Output	y	NxN sparse identity matrix.

Example

```
y = sparseEye(3);  
d = denseSubmat(y,0,0);  
  
      1.0000000 0.0000000 0.0000000  
d =   0.0000000 1.0000000 0.0000000  
      0.0000000 0.0000000 1.0000000
```

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

sparseFD

sparseFD

Purpose	Converts dense matrix gto sparse matrix.	
Format	$y = \text{sparseFD}(x,eps);$	
Input	x	MxN dense matrix.
	eps	scalar, elements of x less than eps will be treated as zero.
Output	y	MxN sparse matrix.
Remarks	A dense matrix is just a normal format matrix.	
Source	sparse.src	

sparseFP

Purpose	Converts packed matrix to sparse matrix.	
Format	$y = \text{sparseFP}(x,r,c);$	
Input	x	Mx3 packed matrix, see remarks for format.
	r	scalar, rows of output matrix.
	c	scalar, columns of output matrix.
Output	y	RxC sparse matrix.
Remarks	x contains the nonzero elements of the sparse matrix. The first column of x contains the element value, the second column the row number, and the third column the column number.	
Source	sparse.src	

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

sparseHConcat

sparseHConcat

Purpose	Horizontally concatenates two sparse matrices.		
Format	$z = \text{sparseHConcat}(y,x);$		
Input	y	MxN sparse matrix, left hand matrix.	
	x	MxL sparse matrix, right hand matrix.	
Output	z	Mx(N+L) sparse matrix.	
Source	sparse.src		

sparseNZE

Purpose Returns the number of nonzero elements in a sparse matrix.

Format $r = \text{sparseNZE}(x);$

Input x MxN sparse matrix.

Output r scalar, number of nonzero elements in x .

Source `sparse.src`

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

sparseOnes

sparseOnes

Purpose	Generates sparse matrix of ones and zeros		
Format	$y = \text{sparseOnes}(x,r,c);$		
Input	x	Mx2 matrix, first column contains row numbers of the ones, and the second column contains column numbers.	
	r	scalar, rows of full matrix.	
	c	scalar, columns of full matrix.	
Output	y	sparse matrix of ones.	
Source	sparse.src		

sparseRows

Purpose Returns the number of rows in a sparse matrix.

Format `r = sparseRows(x);`

Input `x` MxN sparse matrix.

Output `r` scalar, number of rows.

Source `sparse.src`

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

sparseSet

sparseSet

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

Purpose	Resets sparse library global matrices to default values.
Format	sparseSet;
Globals	_sparse_ARnorm, _sparse_Acond, _sparse_Anorm, _sparse_Atol, _sparse_Btol, _sparse_CondLimit, _sparse_Damping, _sparse_NumIters, _sparse_RetCode, _sparse_Rnorm, _sparse_Xnorm
Source	sparse.src

sparseSolve

Purpose	Solves $Ax = B$ for x when A is a sparse matrix.	
Format	$x = \text{sparseSolve}(A,B);$	
Input	A	MxN sparse matrix.
	B	Nx1 vector.
Global Input	<code>_sparse_Damping</code>	scalar, if nonzero, damping coefficient for damped least squares solve, i.e., <div> $\begin{bmatrix} A \\ dI \end{bmatrix} = \begin{bmatrix} B \\ 0 \end{bmatrix}$ </div> is solved for X where $d = \text{_sparse_Damping}$, I is a conformable identity matrix, and 0 a conformable matrix of zeros.
	<code>_sparse_Atol</code>	scalar, an estimate of the relative error in A . If zero, <code>_sparse_Atol</code> is assumed to be machine precision. Default = 0.
	<code>_sparse_Btol</code>	an estimate of the relative error in B . If zero, <code>_sparse_Btol</code> is assumed to be machine precision. Default = 0.
	<code>_sparse_CondLimit</code>	upper limit on condition of A . Iterations will be terminated if a computed estimate of the condition of A exceeds <code>_sparse_CondLimit</code> . If zero, set to 1 / machine precision.
	<code>_sparse_NumIters</code>	maximum number of iterations.
Output	x	Nx1 vector, solution of $Ax = B$.

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

sparseSolve

Global Output		
	_sparse_RetCode	scalar, termination condition.
a		0 x is the exact solution, no iterations performed.
b		1 solution is nearly exact with accuracy on the order of _sparse_Atol and _sparse_Btol .
c		2 solution is not exact and a least squares solution has been found with accuracy on the order of _sparse_Atol .
d		3 the estimate of the condition of A has exceeded _sparse_CondLimit . The system appears to be ill-conditioned.
e		4 solution is nearly exact with reasonable accuracy.
f		5 solution is not exact and a least squares solution has been found with reasonable accuracy.
g		6 iterations halted due to poor condition given machine precision.
h		7 _sparse_NumIters exceeded.
i	_sparse_Anorm	scalar, estimate of Frobenius norm of $\begin{bmatrix} A \\ dI \end{bmatrix}$
j	_sparse_Acond	estimate of condition of A .
k	_sparse_Rnorm	estimate of norm of $\begin{bmatrix} A \\ dI \end{bmatrix}_x - \begin{bmatrix} B \\ 0 \end{bmatrix}$
l	_sparse_ARnorm	estimate of norm of $\begin{bmatrix} A \\ dI \end{bmatrix}' \begin{bmatrix} A \\ dI \end{bmatrix}$
m	_sparse_XAnorm	estimate of norm of x .
n		
o		
p		
q		
r		
s		
t		
u		
v		
w	Source	<code>sparse.src</code>
x y z		

sparseSubmat

Purpose Returns (sparse) submatrix of sparse matrix.

Format $e = \text{sparseSubmat}(x, r, c);$

Input

x	MxN sparse matrix.
r	Kx1 vector, row indices.
c	Lx1 vector, column indices.

Output e KxL sparse matrix.

Remarks If r or c are scalar zeros, all rows or columns will be returned.

Source sparse.src

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

sparseTD

sparseTD

a		
b	Purpose	Multiplies sparse matrix by dense matrix.
c	Format	$z = \text{sparseTD}(s,d);$
d	Input	s MxN sparse matrix.
e		d NxL dense matrix.
f	Output	z MxL dense matrix, the result of $s \times d$.
g	Source	<code>sparse.src</code>
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

sparseTrTD

Purpose	Multiplies sparse matrix transposed by dense matrix.	
Format	$z = \text{sparseTrTD}(s,d);$	
Input	s	NxM sparse matrix.
	d	NxL dense matrix.
Output	z	MxL dense matrix, the result of $s'd$.
Source	sparse.src	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

sparseVConcat

sparseVConcat

a	
b	
c	
d	
e	
f	
g	
h	
i	
j	
k	
l	
m	
n	
o	
p	
q	
r	
s	
t	
u	
v	
w	
x y z	

Purpose	Vertically concatenates two sparse matrices.	
Format	$z = \text{sparseVConcat}(y,x);$	
Input	y	MxN sparse matrix, top matrix.
	x	LxN sparse matrix, bottom matrix.
Output	z	(M+L)xN sparse matrix.
Source	sparse.src	

spline

Purpose Computes a two-dimensional interpolatory spline.

Format `{ u,v,w } = spline(x,y,z,sigma,g);`

Input

<i>x</i>	1xK vector, x-abscissae (x-axis values).
<i>y</i>	Nx1 vector, y-abscissae (y-axis values).
<i>z</i>	KxN matrix, ordinates (z-axis values).
<i>sigma</i>	scalar, tension factor.
<i>g</i>	scalar, grid size factor.

Output

<i>u</i>	1xK*G vector, x-abscissae, regularly spaced.
<i>v</i>	N*Gx1 vector, y-abscissae, regularly spaced.
<i>w</i>	K*G x N*G matrix, interpolated ordinates.

Remarks *sigma* contains the tension factor. This value indicates the curviness desired. If *sigma* is nearly zero (e.g., .001), the resulting surface is approximately the tensor product of cubic splines. If *sigma* is large (e.g., 50.0), the resulting surface is approximately bi-linear. If *sigma* equals zero, tensor products of cubic splines result. A standard value for *sigma* is approximately 1.

g is the grid size factor. It determines the fineness of the output grid. For *g* = 1, the output matrices are identical to the input matrices. For *g* = 2, the output grid is twice as fine as the input grid, i.e., *u* will have twice as many columns as *x*, *v* will have twice as many rows as *y*, and *w* will have twice as many rows and columns as *z*.

Source `spline.src`

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

spline1D

spline1D

Purpose	Computes a smoothing spline for a curve.		
Format	{ <i>u,v</i> } = spline1D (<i>x,y,d,s,sigma,g</i>);		
Input	<i>x</i>	Kx1	vector, x-abscissae (x-axis values).
	<i>y</i>	Kx1	vector, y-ordinates (y-axis values).
	<i>d</i>	Kx1	vector or scalar, observation weights.
	<i>s</i>	scalar,	smoothing parameter, if <i>s</i> = 0, curve performs an interpolation. If <i>d</i> contains standard deviation estimates, a reasonable value for <i>s</i> is K.
	<i>sigma</i>	scalar,	tension factor.
	<i>G</i>	scalar,	grid size factor.
Output	<i>u</i>	K*Gx1	vector, x-abscissae, regularly spaced.
	<i>v</i>	N*Gx1	vector, interpolated ordinates, regularly spaced.
Remarks	<i>sigma</i> contains the tension factor. This value indicates the curviness desired. If <i>sigma</i> is nearly zero (e.g. 0.001), the resulting curve is approximately the tensor product of cubic splines. If <i>sigma</i> is large (e.g. 50), the resulting curve is approximately bi-linear. If <i>sigma</i> equals zero, tensor products of cubic splines result. A standard value for <i>sigma</i> is approximately 1.		
 <i>G</i> is the grid size factor. It determines the fineness of the output grid. For <i>G</i> = 1, the output matrices are identical to the input matrices. For <i>G</i> = 2, the output grid is twice as fine as the input grid, i.e., <i>u</i> will have twice as many columns as <i>x</i> , and <i>v</i> will have twice as many rows as <i>y</i> .			

spline2D

Purpose Computes a smoothing spline for a surface.

Format $\{ u, v, w \} = \text{spline2D}(x, y, z, \text{sigma}, g);$

Input

x	Kx1 vector, x-abscissae (x-axis values).
y	Nx1 vector, y-abscissae (y-axis values).
z	KxN matrix, ordinates (z-axis values).
sigma	scalar, tension factor.
G	scalar, grid size factor.

Output

u	1xK*G vector, x-abscissae, regularly spaced.
v	N*Gx1 vector, y-abscissae, regularly spaced.
w	K*GxN*G matrix, interpolated ordinates.

Remarks sigma contains the tension factor. This value indicates the curviness desired. If sigma is nearly zero (e.g. 0.001), the resulting surface is approximately the tensor product of cubic splines. If sigma is large (e.g. 50.), the resulting surface is approximately bi-linear. If sigma equals zero, tensor products of cubic splines result. A standard value for sigma is approximately 1.

G is the grid size factor. It determines the fineness of the output grid. For $G = 1$, the output matrices are identical to the input matrices. For $G = 2$, the output grid is twice as fine as the input grid, i.e., u will have twice as many columns as x , and v will have twice as many rows as y .

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

sqpSolve

sqpSolve

a	Purpose	Solves the nonlinear programming problem using a sequential quadratic programming method.	
b			
c	Format	{ <i>x,f,lagr,retcode</i> } = sqpSolve (& <i>fct</i> , <i>start</i>);	
d			
e	Input	&<i>fct</i>	pointer to a procedure that computes the function to be minimized. This procedure must have one input argument, a vector of parameter values, and one output argument, the value of the function evaluated at the input vector of parameter values.
f		<i>start</i>	Kx1 vector of start values.
g	Global Input	_sqp_A	MxK matrix, linear equality constraint coefficients.
h		_sqp_B	Mx1 vector, linear equality constraint constants.
i		These globals are used to specify linear equality constraints of the following type:	
j		$_sqp_A * X = _sqp_B$	
k		where X is the Kx1 unknown parameter vector.	
l			
m			
n			
o			
p			
q			
r			
s			
t			
u			
v			
w			
x y z			

sqpSolve**_sqp_EqProc**

scalar, pointer to a procedure that computes the nonlinear equality constraints. For example, the statement:

```
_sqp_EqProc = eqproc;
```

tells **sqpSolve** that nonlinear equality constraints are to be placed on the parameters and where the procedure computing them is to be found. The procedure must have one input argument, the Kx1 vector of parameters, and one output argument, the Rx1 vector of computed constraints that are to be equal to zero. For example, suppose that you wish to place the following constraint:

$$P[1] * P[2] = P[3]$$

The procedure for this is:

```
proc eqproc(p);
    retp(p[1]*[2]-p[3]);
endp;
```

_sqp_C

MxK matrix, linear inequality constraint coefficients.

_sqp_D

Mx1 vector, linear inequality constraint constants.

These globals are used to specify linear inequality constraints of the following type:

$$_sqp_C * X \geq _sqp_D$$

where X is the Kx1 unknown parameter vector.

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

sqpSolve

	<code>_sqp_IneqProc</code>	scalar, pointer to a procedure that computes the nonlinear inequality constraints. For example the statement: <code>_sqp_EqProc = &ineqproc;</code> tells sqpSolve that nonlinear equality constraints are to be placed on the parameters and where the procedure computing them is to be found. The procedure must have one input argument, the Kx1 vector of parameters, and one output argument, the Rx1 vector of computed constraints that are to be equal to zero. For example, suppose that you wish to place the following constraint: $P[1] * P[2] \geq P[3]$ The procedure for this is: <code>proc ineqproc(p);</code> <code>ret p[1]*[2]-p[3];</code> <code>endp;</code>
	<code>_sqp_Bounds</code>	Kx2 matrix, bounds on parameters. The first column contains the lower bounds, and the second column the upper bounds. If the bounds for all the coefficients are the same, a 1x2 matrix may be used. Default is: $\begin{bmatrix} 1 & -1e256 \\ 2 & 1e256 \end{bmatrix}$

sqpSolve**_sqp_GradProc**

scalar, pointer to a procedure that computes the gradient of the function with respect to the parameters. For example, the statement:

```
_sqp_GradProc = &gradproc;
```

tells **sqpSolve** that a gradient procedure exists and where to find it. The user-provided procedure has two input arguments, a Kx1 vector of parameter values and an NxP matrix of data. The procedure returns a single output argument, an NxK matrix of gradients of the log-likelihood function with respect to the parameters evaluated at the vector of parameter values.

Default = 0, i.e., no gradient procedure has been provided.

_sqp_HessProc

scalar, pointer to a procedure that computes the Hessian, i.e., the matrix of second order partial derivatives of the function with respect to the parameters. For example, the instruction:

```
_sqp_HessProc = &hessproc;
```

will tell **sqpSolve** that a procedure has been provided for the computation of the Hessian and where to find it. The procedure that is provided by the user must have two input arguments, a Px1 vector of parameter values and an NxK data matrix. The procedure returns a single output argument, the PxP symmetric matrix of second order derivatives of the function evaluated at the parameter values.

_sqp_MaxIters

scalar, maximum number of iterations. Default = 1e+5. Termination can be forced by pressing C on the keyboard.

_sqp_DirTol

scalar, convergence tolerance for gradient of estimated coefficients. Default = 1e-5. When this criterion has been satisfied **sqpSolve** will exit the iterations.

_sqp_ParNames

Kx1 character vector, parameter names.

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

sqpSolve

	_sqp_PrintIters	scalar, if nonzero, prints iteration information. Default = 0. Can be toggled during iterations by pressing P on the keyboard.
	_sqp_FeasibleTest	scalar, if nonzero, parameters are tested for feasibility before computing function in line search. If function is defined outside inequality boundaries, then this test can be turned off.
	_sqp_RandRadius	scalar, If zero, no random search is attempted. If nonzero it is the radius of random search which is invoked whenever the usual line search fails. Default = .01.
	__output	scalar, if nonzero, results are printed. Default = 0.

Output	<i>x</i>	Kx1 vector of parameters at minimum.
	<i>f</i>	scalar, function evaluated at <i>x</i> .
	<i>lagr</i>	vector, created using vput . Contains the Lagrangean for the constraints. They may be extracted with the vread command using the following strings: “lineq” Lagrangeans of linear equality constraints “nlineq” Lagrangeans of nonlinear equality constraints “linineq” Lagrangeans of linear inequality constraints “nlinineq” Lagrangeans of nonlinear inequality constraints “bounds” Lagrangeans of bounds Whenever a constraint is active, its associated Lagrangean will be nonzero.
	<i>retcode</i>	return code: 0 normal convergence 1 forced exit 2 maximum number of iterations exceeded 3 function calculation failed 4 gradient calculation failed 5 Hessian calculation failed 6 line search failed 7 error with constraints

sqpSolve

Remarks Pressing C on the keyboard will terminate iterations, and pressing P will toggle iteration output.

sqpSolve is recursive, that is, it can call itself with another function and set of global variables.

To reset global variables for this function to their default values, call **sqpSolveSet**.

Example

```
sqpSolveSet;

proc fct(x);
    retp( (x[1] + 3*x[2] + x[3])^2 + 4*(x[1] -
        x[2])^2 );
endp;

proc ineqp(x);
    retp(6*x[2] + 4*x[3] - x[1]^3 - 3);
endp;

proc eqp(x);
    retp(1-sumc(x));
endp;

_sqp_Bounds = { 0 1e256 };

start = { .1, .7, .2 };

_sqp_IneqProc = &ineqp;

_sqp_EqProc = &eqp;

{ x,f,lagr,ret } = sqpSolve( &fct,start );
```

Source sqpsolve.src

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

sqrt

sqrt

a	Purpose	Computes the square root of every element in a matrix.
b	Format	$y = \text{sqrt}(x);$
c	Input	x NxK matrix.
d	Output	y NxK matrix, the square roots of each element of x .
e	Remarks	If x is negative, complex results are returned. You can turn the generation of complex numbers for negative inputs on or off in the GAUSS configuration file, and with the sysstate function, case 8. If you turn it off, sqrt will generate an error for negative inputs. If x is already complex, the complex number state doesn't matter; sqrt will compute a complex result.
f	Example	$\text{let } x[2,2] = 1 \ 2 \ 3 \ 4;$ $y = \text{sqrt}(x);$ $x = \begin{matrix} 1.00000000 & 2.00000000 \\ 3.00000000 & 4.00000000 \end{matrix}$ $y = \begin{matrix} 1.00000000 & 1.41421356 \\ 1.73205081 & 2.00000000 \end{matrix}$
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

stdc

Purpose	Computes the standard deviation of the elements in each column of a matrix.
Format	<code>y = stdc(x);</code>
Input	<code>x</code> NxK matrix.
Output	<code>y</code> Kx1 vector, the standard deviation of each column of <code>x</code> .
Remarks	<p>This function essentially computes:</p> $\text{sqrt}(1/(N-1) * \text{sumc}((x - \text{meanc}(x))')^2))$ <p>Thus, the divisor is N-1 rather than N, where N is the number of elements being summed. To convert to the alternate definition, multiply by</p> $\text{sqrt}((N-1)/N)$
Example	<pre>y = rndn(8100,1); std = stdc(y); std = 1.008377</pre> <p>In this example, 8100 standard Normal random variables are generated, and their standard deviation is computed.</p>
See also	meanc

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

stocv

stocv

a	Purpose	Converts a string to a character vector.
b	Format	$v = \text{stocv}(s);$
c	Input	s string, to be converted to character vector.
d	Output	v Nx1 character vector, contains the contents of s .
e	Remarks	stocv breaks s up into a vector of 8-character length matrix elements. Note that the character information in the vector is not guaranteed to be null-terminated.
f	Example	$s = \text{"Now is the time for all good men"};$ $v = \text{stocv}(s);$ "Now is t" = "he time " "for all " "good men"

See also **cvtos, vget, vlist, vput, vread**

stof

Purpose	Converts a string to floating point.	
Format	$y = \mathbf{stof}(x);$	
Input	x	string, or NxK matrix containing character elements to be converted.
Output	y	matrix, the floating point equivalents of the ASCII numbers in x .
Remarks	<p>If x is a string containing “1 2 3”, then stof will return a 3x1 matrix containing the numbers 1, 2, and 3.</p> <p>If x is a null string, stof will return a 0.</p> <p>This uses the same input conversion routine as loadm and let. It will convert character elements and missing values. stof also converts complex numbers in the same manner as let.</p>	
See also	ftos, ftocv, chrs	

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

stop

stop

Purpose	Stops a program and returns to the command prompt. Does not close files.
Format	stop;
Remarks	<p>This command has the same effect as end, except it does not close files or the auxiliary output.</p> <p>It is not necessary to put a stop or an end statement at the end of a program. If neither is found, an implicit stop is executed.</p>
See also	end, new, system

strindx

Purpose	Finds the index of one string within another string.
Format	<code>y = strindx(<i>where</i>,<i>what</i>,<i>start</i>);</code>
Input	<p><i>where</i> string or scalar, the data to be searched.</p> <p><i>what</i> string or scalar, the substring to be searched for in <i>where</i>.</p> <p><i>start</i> scalar, the starting point of the search in <i>where</i> for an occurrence of <i>what</i>. The index of the first character in a string is 1.</p>
Output	<i>y</i> scalar containing the index of the first occurrence of <i>what</i> , within <i>where</i> , which is greater than or equal to <i>start</i> . If no occurrence is found, it will be 0.
Remarks	<p>An example of the use of this function is the location of a name within a string of names:</p> <pre>z = "Whatchmacallit"; x = "call"; y = strindx(z,x,1);</pre> <p><code>y = 9</code></p> <p>This function is used with strsect for extracting substrings.</p>
See also	strrindx , strlen , strsect , strput

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

strlen

strlen

a	Purpose	Returns the length of a string.
b	Format	<code>y = strlen(x);</code>
c	Input	<code>x</code> string or NxK matrix of character data.
d	Output	<code>y</code> scalar containing the exact length of the string <code>x</code> or NxK matrix of the lengths of the elements in the matrix <code>x</code> .
e	Remarks	The null character (ASCII 0) is a legal character within strings and so embedded nulls will be counted in the length of strings. The final terminating null byte is not counted, though. For character matrices, the length is computed by counting the characters (maximum of 8) up to the first null in each element of the matrix. The null character, therefore, is not a valid character in matrices containing character data and is not counted in the lengths of the elements of those matrices.
f	Example	<code>x = "How long?";</code> <code>y = strlen(x);</code> <code>y = 9</code>
g	See also	<code>strsect</code> , <code>strindx</code> , <code>strrindx</code>
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

strput

Purpose	Lays a substring over a string.
Format	<code>y = strput(substr,str,off);</code>
Input	<i>substr</i> string, the substring to be laid over the other string. <i>str</i> string, the string to receive the substring. <i>off</i> scalar, the offset in <i>str</i> to place <i>substr</i> . The offset of the first byte is 1.
Output	<i>y</i> string, the new string.
Example	<pre>str = "max"; sub = "imum"; f = 4; y = strput(sub,str,f); print y;</pre> <p>Produces: maximum</p>
Source	strput.src

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

strrindx**strrindx**

Purpose	Finds the index of one string within another string. Searches from the end of the string to the beginning.
Format	<code>y = strrindx(where,what,start) ;</code>
Input	<p><i>where</i> string or scalar, the data to be searched.</p> <p><i>what</i> string or scalar, the substring to be searched for in <i>where</i>.</p> <p><i>start</i> scalar, the starting point of the search in <i>where</i> for an occurrence of <i>what</i>. <i>where</i> will be searched from this point backward for <i>what</i>.</p>
Output	<i>y</i> scalar containing the index of the last occurrence of <i>what</i> , within <i>where</i> , which is less than or equal to <i>start</i> . If no occurrence is found, it will be 0.
Remarks	<p>A negative value for <i>start</i> causes the search to begin at the end of the string. An example of the use of strrindx is extracting a file name from a complete path specification:</p> <pre> path = "/gauss/src/ols.src"; ps = "/"; pos = strrindx(path,ps,-1); if pos; name = strsect(path,pos+1,strlen(path)); else; name = ""; endif; pos = 11 name = "ols.src" strrindx can be used with strsect for extracting substrings. </pre>
See also	strindx , strlen , strsect , strput

strsect

Purpose	Extracts a substring of a string.	
Format	<code>y = strsect(str,start,len);</code>	
Input	<i>str</i>	string or scalar from which the segment is to be obtained.
	<i>start</i>	scalar, the index of the substring in <i>str</i> . The index of the first character is 1.
	<i>len</i>	scalar, the length of the substring.
Output	<i>y</i>	string, the extracted substring, or a null string if <i>start</i> is greater than the length of <i>str</i> .
Remarks	<p>If there are not enough characters in a string for the defined substring to be extracted, then a short string or a null string will be returned.</p> <p>If <i>str</i> is a matrix containing character data, it must be scalar.</p>	
Example	<pre>strng = "This is an example string." y = strsect(strng,12,7); y = example</pre>	
See also	strlen, strindx, strrindx	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

strsplit

strsplit

a	Purpose	Splits an Nx1 string vector into an NxK string array of the individual tokens.
b		
c	Format	<code>sa = strsplit(sv);</code>
d	Input	<code>sv</code> Nx1 string array.
e	Output	<code>sa</code> NxK string array.
f	Remarks	Each row of <code>sv</code> must contain the same number of tokens. The following characters are considered delimiters between tokens: space ASCII 32 tab ASCII 9 comma ASCII 44 newline ASCII 10 carriage return ASCII 13 Tokens containing delimiters must be enclosed in single or double quotes or parentheses. Tokens enclosed in single or double quotes will NOT retain the quotes upon translation. Tokens enclosed in parentheses WILL retain the parentheses after translation. Parentheses cannot be nested.
g		
h		
i		
j		
k		
l		
m		
n		
o	Example	<code>let string sv = { "dog 'cat fish' moose", "lion, zebra, elk", "seal owl whale" }; sa = strsplit(sv);</code>
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

strsplit

```
      'dog'   'cat fish'  'moose'  
sa = 'lion'  'zebra'     'elk'  
      'seal'  'owl'       'whale'
```

See also **strsplitPad**

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

strsplitPad

strsplitPad

a	Purpose	Splits a string vector into a string array of the individual tokens. Pads on the right with null strings.	
b	Format	<i>sa</i> = strsplitPad (<i>sv</i> , <i>cols</i>);	
c	Input	<i>sv</i>	Nx1 string array.
d		<i>cols</i>	scalar, number of columns of output string array.
e	Output	<i>sa</i>	Nxcols string array.
f	Remarks	Rows containing more than <i>cols</i> tokens are truncated and rows containing fewer than <i>cols</i> tokens are padded on the right with null strings. The following characters are considered delimiters between tokens:	
g		space	ASCII 32
h		tab	ASCII 9
i		comma	ASCII 44
j		newline	ASCII 10
k		carriage return	ASCII 13
l		Tokens containing delimiters must be enclosed in single or double quotes or parentheses. Tokens enclosed in single or double quotes will NOT retain the quotes upon translation. Tokens enclosed in parentheses WILL retain the parentheses after translation. Parentheses cannot be nested.	
m			
n			
o			
p	Example	let string sv = {	
q		"dog 'cat fish' moose",	
r		"lion, zebra, elk, bird",	
s		"seal owl whale"	
t		};	
u			
v		<i>sa</i> = strsplitPad(<i>sv</i> , 4);	
w			
x			
y			
z			

strsplitPad

```
          'dog'   'cat fish'  'moose'  ''
sa = 'lion' 'zebra'  'elk'    'bird'
          'seal'  'owl'     'whale'  ''
```

See also **strsplit**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

strtodt

strtodt

a	Purpose	Converts a string array of dates to a matrix in DT scalar format.
b	Format	$x = \text{strtodt}(sa, fmt);$
c	Input	sa N x K string array containing dates. fmt string containing date/time format characters.
d	Output	x N x K matrix of dates in DT scalar format.
e	Remarks	The DT scalar format is a double precision representation of the date and time. In the DT scalar format, the number 20010421183207 represents 18:32:07 or 6:32:07 PM on April 21, 2001. The following formats are supported: YYYY 4 digit year YR Last two digits of year MO Number of month, 01-12 DD Day of month, 01-31 HH Hour of day, 00-23 MI Minute of hour, 00-59 SS Second of minute, 00-59
f	Example	$x = \text{strtodt}("2001-03-25\ 14:58:49",$ $"YYYY-MO-DD\ HH:MI:SS");$ $\text{print } x;$ produces: 20010325145849.0
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

strtodt

```
x = strtodt("2001-03-25 14:58:49", "YYYY-MO-DD");  
print x;
```

```
produces:  
20010325000000.0
```

```
x = strtodt("14:58:49", "HH:MI:SS");  
print x;
```

```
produces:  
145849.0
```

```
x = strtodt("04-15-00", "MO-DD-YR");  
print x;
```

```
produces:  
20000415000000.0
```

See also **dttostr**, **dttoutc**, **utctodt**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

strtof

strtof

Purpose	Converts a string array to a numeric matrix.	
Format	$x = \text{strtof}(sa);$	
Input	sa	NxK string array containing numeric data.
Output	x	NxK matrix.
Remarks	This function supports real matrices only. Use strtofcplx for complex data.	
See also	strtofcplx, ftostrC	

strtofcp1x

Purpose	Converts a string array to a complex numeric matrix.	
Format	$x = \text{strtofcp1x}(sa);$	
Input	sa	NxK string array containing numeric data.
Output	x	NxK complex matrix.
Remarks	strtofcp1x supports both real and complex data. It is slower than strtof for real matrices. strtofcp1x requires the presence of the real part. The imaginary part can be absent.	
See also	strtof , ftostrC	

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

submat

submat

a	Purpose	Extracts a submatrix of a matrix, with the appropriate rows and columns given by the elements of vectors.
b	Format	<code>y = submat(x,r,c);</code>
c	Input	<code>x</code> NxK matrix. <code>r</code> LxM matrix of row indices. <code>c</code> PxQ matrix of column indices.
d	Output	<code>y</code> (L*M)x(P*Q) submatrix of <code>x</code> , <code>y</code> may be larger than <code>x</code> .
e	Remarks	If <code>r = 0</code> , then all rows of <code>x</code> will be used. If <code>c = 0</code> , then all columns of <code>x</code> will be used.
f	Example	<pre>let x[3,4] = 1 2 3 4 5 6 7 8 9 10 11 12; let v1 = 1 3; let v2 = 2 4; y = submat(x,v1,v2); z = submat(x,0,v2); 1 2 3 4 x = 5 6 7 8 9 10 11 12 2 4 y = 10 12 2 4 z = 6 8 10 12</pre>
g	See also	<code>diag</code> , <code>vec</code> , <code>reshape</code>
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

subscat

Purpose	Changes the values in a vector depending on the category a particular element falls in.		
Format	$y = \text{subscat}(x, v, s);$		
Input	x	Nx1 vector.	
	v	Px1 numeric vector, containing breakpoints specifying the ranges within which substitution is to be made. This MUST be sorted in ascending order.	
		v can contain a missing value as a separate category if the missing value is the first element in v .	
		If v is a scalar, all matches must be exact for a substitution to be made.	
	s	Px1 vector, containing values to be substituted.	
Output	y	Nx1 vector, with the elements in s substituted for the original elements of x according to which of the regions the elements of x fall into:	
		$\begin{array}{ll} v[1] < x \leq v[1] \rightarrow & s[1] \\ & x \leq v[2] \rightarrow & s[2] \\ & \dots \\ v[p-1] < x \leq v[p] \rightarrow & s[p] \\ & x > v[p] \rightarrow & \text{the original value of } x \end{array}$	
		If missing is not a category specified in v , missings in x are passed through without change.	

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

subscat

Example

```
let x = 1 2 3 4 5 6 7 8 9 10;  
let v = 4 5 8;  
let s = 10 5 0;  
y = subscat(x,v,s);
```

y =

10
10
10
10
5
0
0
0
9
10

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

substute

Purpose Substitutes new values for old values in a matrix, depending on the outcome of a logical expression.

Format `y = substute(x,e,v);`

Input

<i>x</i>	NxK matrix containing the data to be changed.
<i>e</i>	LxM matrix, ExE conformable with <i>x</i> containing 1's and 0's. Elements of <i>x</i> will be changed if the corresponding element of <i>e</i> is 1.
<i>v</i>	PxQ matrix, ExE conformable with <i>x</i> and <i>e</i> , containing the values to be substituted for the original values of <i>x</i> when the corresponding element of <i>e</i> is 1.

Output *y* max(N,L,P) by max(K,M,Q) matrix.

Remarks The *e* matrix is usually the result of an expression or set of expressions using dot conditional and boolean operators.

Example

```
x = { Y   55   30 ,
      N   57   18 ,
      Y   24    3 ,
      N   63   38 ,
      Y   55   32 ,
      N   37   11 };

e = x[.,1] .==$= "Y" .and x[.,2] .>= 55 .and x[.,3] .>=
    30;

x[.,1] = substute(x[.,1],e,0$+"R");
```

substute

$$e = \begin{matrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{matrix}$$

Here is what x looks like after substitution:

$$y = \begin{matrix} & R & 55 & 30 \\ & N & 57 & 18 \\ Y & 24 & & 3 \\ & N & 63 & 38 \\ & R & 55 & 32 \\ & N & 37 & 11 \end{matrix}$$

Source `datatran.src`

See also **`code`**, **`recode`**

sumc

Purpose	Computes the sum of each column of a matrix.
Format	$y = \text{sumc}(x);$
Input	x NxK matrix.
Output	y Kx1 vector, the sums of each column of x .
Remarks	<p>To find the sums of the elements in each row of a matrix, transpose before applying sumc. If x is complex, use the bookkeeping transpose (\cdot').</p> <p>To find the sums of all of the elements in a matrix, use the vecr function before applying sumc.</p>
Example	<pre>x = round(rndu(5,3)*5); y = sumc(x);</pre> $x = \begin{bmatrix} 2 & 4 & 3 \\ 2 & 1 & 2 \\ 5 & 1 & 3 \\ 5 & 1 & 1 \\ 3 & 3 & 4 \end{bmatrix}$ $y = \begin{bmatrix} 17 \\ 10 \\ 13 \end{bmatrix}$
See also	cumsumc , meanc , stdc

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

surface

surface

a		
b	Purpose	Graphs a 3-D surface.
c	Library	pgraph
d	Format	surface (<i>x,y,z</i>) ;
e	Input	<i>x</i> 1xK vector, the X axis data.
f		<i>y</i> Nx1 vector, the Y axis data.
g		<i>z</i> NxK matrix, the matrix of height data to be plotted.
h	Global Input	_psurf 2x1 vector, controls 3-D surface characteristics.
i		[1] if 1, show hidden lines. Default 0.
j		[2] color for base (default 7). The base is an outline of
k		the X-Y plane with a line connecting each corner to
l		the surface. If 0, no base is drawn.
m		_pticout scalar, if 0 (default), tick marks point inward, if 1, tick
n		marks point outward.
o		_pzclr Z level color control.
p		There are 3 ways to set colors for the Z levels of a surface
q		graph.
r		1. To specify a single color for the entire surface plot,
s		set the color control variable to a scalar value 1–15.
t		Example:
u		_pzclr = 15;
v		2. To specify multiple colors distributed evenly over
w		the entire Z range, set the color control variable to a
x		vector containing the desired colors only. GAUSS
y		will automatically calculate the required
z		corresponding Z values for you. The following
		example will produce a three color surface plot, the
		Z ranges being lowest=blue, middle=light blue,
		highest=white:
		_pzclr = { 1, 10, 15 };

surface

3. To specify multiple colors distributed over selected ranges, the Z ranges as well as the colors must be manually input by the user. The following example assumes -0.2 to be the minimum value in the *z* matrix:

```
_pzclr = {-0.2 1, /* z ≥ -0.2 */
          /* blue */
          0.0 10, /* z ≥ 0.0 light blue */
          0.2 15 }; /* z ≥ 0.2 white */
```

Since a Z level is required for each selected color, the user must be responsible to compute the minimum value of the *z* matrix as the first Z range element. This may be most easily accomplished by setting the **_pzclr** matrix as shown above (the first element being an arbitrary value), then reset the first element to the minimum *z* value as follows:

```
_pzclr = { -0.0 1,
           0.0 10,
           0.2 15 };
_pzclr [1,1] = minc(minc(z));
```

0 Black	8 Dark Grey
1 Blue	9 Light Blue
2 Green	10 Light Green
3 Cyan	11 Light Cyan
4 Red	12 Light Red
5 Magenta	13 Light Magenta
6 Brown	14 Yellow
7 Grey	15 White

Source `psurface.src`

See also **volume**, **view**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

svd

svd

a		
b	Purpose	Computes the singular values of a matrix.
c	Format	<code>s = svd(x);</code>
d	Input	<code>x</code> NxP matrix whose singular values are to be computed.
e	Output	<code>s</code> Mx1 vector, where M = min(N,P), containing the singular values of <code>x</code> arranged in descending order.
f		
g	Global Output	<code>_svderr</code> global scalar, if not all of the singular values can be computed <code>_svderr</code> will be nonzero. The singular values in <code>s[_svderr+1]</code> , ... <code>s[M]</code> will be correct.
h		
i	Remarks	Error handling is controlled with the low bit of the trap flag.
j		<code>trap 0</code> set <code>_svderr</code> and terminate with message
k		<code>trap 1</code> set <code>_svderr</code> and continue execution
l	Example	<code>x = { 4 3 6 7,</code> <code>8 2 9 5 },</code> <code>y = svd(x);</code> <code>y = 16.521787</code> <code>3.3212254</code>
m		
n		
o		
p	Source	<code>svd.src</code>
q	See also	<code>svd1, svd2, svds</code>
r		
s		
t		
u		
v		
w		
x y z		

svd1

Purpose	Computes the singular value decomposition of a matrix so that: $x = u * s * v'$.			
Format	{ u, s, v } = svd1 (x);			
Input	x	NxP matrix whose singular values are to be computed.		
Output	u	NxN matrix, the left singular vectors of x .		
	s	NxP diagonal matrix, containing the singular values of x arranged in descending order on the principal diagonal.		
	v	PxP matrix, the right singular vectors of x .		
Global Output	_svderr	global scalar, if all of the singular values are correct, _svderr is 0. If not all of the singular values can be computed, _svderr is set and the diagonal elements of s with indices greater than _svderr are correct.		
Remarks	Error handling is controlled with the low bit of the trap flag.			
	trap 0	set _svderr and terminate with message		
	trap 1	set _svderr and continue execution		
Example	$x = \text{rndu}(3, 3);$			
	{ u, s, v } = svd1 (x);			
	$x =$	0.97847012	0.20538614	0.59906497
		0.85474208	0.79673540	0.22482095
		0.33340653	0.74443792	0.75698778
		-0.57955818	0.65204491	1.48882486
	$u =$	-0.61005618	0.05056673	-0.79074298
		-0.54031821	-0.75649219	0.36847767

svd1

$$s = \begin{matrix} & 1.84994646 & 0.00000000 & 0.00000000 \\ & 0.00000000 & 0.60370542 & 0.00000000 \\ & 0.00000000 & 0.00000000 & 0.47539239 \end{matrix}$$
$$v = \begin{matrix} & -0.68578561 & 0.71062560 & -0.15719208 \\ & -0.54451302 & -0.64427479 & -0.53704336 \\ & -0.48291165 & -0.28270348 & 0.82877927 \end{matrix}$$

Source `svd.src`

See also **svd**, **svd2**, **svdusv**

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

svd2

Purpose	Computes the singular value decomposition of a matrix so that: $x = u * s * v'$ (compact u).	
Format	{ u, s, v } = svd2 (x);	
Input	x	NxP matrix whose singular values are to be computed.
Output	u	NxN or NxP matrix, the left singular vectors of x . If $N > P$, then u will be NxP containing only the P left singular vectors of x .
	s	NxP or PxP diagonal matrix, containing the singular values of x arranged in descending order on the principal diagonal. If $N > P$, then s will be PxP.
	v	PxP matrix, the right singular vectors of x .
Global Output	_svderr	global scalar, if all of the singular values are correct, _svderr is 0. If not all of the singular values can be computed, _svderr is set and the diagonal elements of s with indices greater than _svderr are correct.
Remarks	Error handling is controlled with the low bit of the trap flag.	
	trap 0	set _svderr and terminate with message
	trap 1	set _svderr and continue execution
Source	svd.src	
Globals	_svderr	
See also	svd , svd1 , svdcusv	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

svdcusv

svdcusv

a		
b		
c		
d		
e		
f		
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

Purpose	Computes the singular value decomposition of a matrix so that: $x = u * s * v'$ (compact u).	
Format	{ u,s,v } = svdcusv (x);	
Input	x	NxP matrix whose singular values are to be computed.
Output	u	NxN or NxP matrix, the left singular vectors of x . If $N > P$, u is NxP containing only the P left singular vectors of x .
	s	NxP or PxP diagonal matrix, the singular values of x arranged in descending order on the principal diagonal. If $N > P$, s is PxP.
	v	PxP matrix, the right singular vectors of x .
Remarks	If not all the singular values can be computed, $s[1,1]$ is set to a scalar error code. Use scalerr to convert this to an integer. The diagonal elements of s with indices greater than scalerr ($s[1,1]$) are correct. If scalerr ($s[1,1]$) returns a 0, all the singular values have been computed.	
See also	svd2 , svds , svdusv	

svds

Purpose	Computes the singular values of a matrix.	
Format	$s = \text{svds}(x);$	
Input	x	$N \times P$ matrix whose singular values are to be computed.
Output	s	$\min(N,P) \times 1$ vector, the singular values of x arranged in descending order.
Remarks	If not all the singular values can be computed, $s[1]$ is set to a scalar error code. Use scalerr to convert this to an integer. The elements of s with indices greater than scalerr ($s[1]$) are correct. If scalerr ($s[1]$) returns a 0, all the singular values have been computed.	
See also	svd , svdcusv , svdusv	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

svdusv

svdusv

a		
b		
c		
d		
e		
f		
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

Purpose	Computes the singular value decomposition of a matrix so that: $x = u * s * v'$.	
Format	{ u,s,v } = svdusv (x);	
Input	x	NxP matrix whose singular values are to be computed.
Output	u	NxN matrix, the left singular vectors of x .
	s	NxP diagonal matrix, the singular values of x arranged in descending order on the principal diagonal.
	v	PxP matrix, the right singular vectors of x .
Remarks	If not all the singular values can be computed, $s[1,1]$ is set to a scalar error code. Use scalerr to convert this to an integer. The diagonal elements of s with indices greater than scalerr ($s[1,1]$) are correct. If scalerr ($s[1,1]$) returns a 0, all the singular values have been computed.	
See also	svd1 , svdcusv , svds	

sysstate

Purpose	Gets or sets general system parameters.	
Format	{ <i>rets...</i> } = sysstate (<i>case,y</i>);	
Input	Case 1	Version Information Returns the current GAUSS version information in an 8-element numeric vector.
	Cases 2 through 7	GAUSS System Paths Get or set GAUSS system path.
	Case 8	Complex Number Toggle Controls automatic generation of complex numbers in sqrt , ln , and log for negative arguments.
	Case 9	Complex Trailing Character Get and set trailing character for the imaginary part of a complex number.
	Case 10	Printer Width Get and set lprint width.
	Case 11	Auxiliary Output Width Get and set the auxiliary output width.
	Case 12	Precision Get and set precision for positive definite matrix routines.
	Case 13	LU Tolerance Get and set singularity tolerance for LU decomposition.
	Case 14	Cholesky Tolerance Get and set singularity tolerance for Cholesky decomposition.
	Case 15	Screen State Get and set window state as controlled by screen command.
	Case 16	Automatic print Mode Get and set automatic print mode.

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

sysstate

	Case 17	Automatic lprint Mode Get and set automatic lprint mode.
a	Case 18	Auxiliary Output Get auxiliary output parameters.
b	Case 19	Get/Set Format Get and set format parameters.
c	Case 21	Imaginary Tolerance Get and set the imaginary tolerance.
d	Case 22	Source Path Get and set the path the compiler will search for source files.
e	Case 24	Dynamic Library Directory Get and set the path for the default dynamic library directory.
f	Case 25	Temporary File Path
g	Case 26	Interface Mode Returns the current interface mode.
h	Case 28	Random Number Generator Parameters
i	Case 30	Base Year Toggle Specifies whether year value returned by date is to include base year (1900) or not.
j		
k		
l		
m		

Case 1: **Version Information**
Returns the current GAUSS version information in an 8-element numeric vector.

Format `vi = sysstate(1,0);`
Output `vi` 8x1 numeric vector containing version information:
[1] Major version number.
[2] Minor version number.
[3] Revision.
[4] Machine type.
[5] Operating system.
[6] Runtime module.
[7] Light version.
[8] Always 0.

sysstate

vi[4] indicates the type of machine on which GAUSS is running:

- 1 Intel x86
- 2 Sun SPARC
- 3 IBM RS/6000
- 4 HP 9000
- 5 SGI MIPS
- 6 DEC Alpha

vi[5] indicates the operating system on which GAUSS is running:

- 1 DOS
- 2 SunOS 4.1.x
- 3 Solaris 2.x
- 4 AIX
- 5 HP-UX
- 6 IRIX
- 7 OSF/1
- 8 OS/2
- 9 Windows

Cases 2 through 7:

GAUSS System Paths

Get or set GAUSS system path.

Format *oldpath* = **sysstate**(*case,path*);

Input *case* scalar 2-7, path to set.

- 2 . exe file location.
- 3 **loadexe** path.
- 4 **save** path.
- 5 **load**, **loadm** path.
- 6 **loadf**, **loadp** path.
- 7 **loads** path.

path scalar 0 to get path, or string containing the new path.

Output *oldpath* string, original path.

Remarks If *path* is of type matrix, the path will be returned but not modified.

sysstate

Case 8: **Complex Number Toggle**

Controls automatic generation of complex numbers in **sqrt**, **ln** and **log** for negative arguments.

Format *oldstate* = **sysstate**(8,*state*) ;

Input *state* scalar, 1, 0, or -1.

Output *oldstate* scalar, the original state.

Remarks If *state* = 1, **log**, **ln**, and **sqrt** will return complex numbers for negative arguments. If *state* = 0, the program will terminate with an error message when negative numbers are passed to **log**, **ln**, and **sqrt**. If *state* = -1, the current state is returned and left unchanged. The default state is 1.

Case 9: **Complex Trailing Character**

Get and set trailing character for the imaginary part of a complex number.

Format *oldtrail* = **sysstate**(9,*trail*) ;

Input *trail* scalar 0 to get character, or string containing the new trailing character.

Output *oldtrail* string, the original trailing character.

Remarks The default character is "i".

Case 10: **Printer Width**

Get and set **lprint** width.

Format *oldwidth* = **sysstate**(10,*width*) ;

Input *width* scalar, new printer width.

Output *oldwidth* scalar, the current original width.

Remarks If *width* is 0, the printer width will not be changed.
This may also be set with the **lpwidth** command.

See also **lpwidth**

Case 11: Auxiliary Output Width

Get and set the auxiliary output width.

Format *oldwidth* = **sysstate(11,width);**

Input *width* scalar, new output width.

Output *oldwidth* scalar, the original output width.

Remarks If *width* is 0 then the output width will not be changed.
This may also be set with the **outwidth** command.

See also **outwidth**

Case 12: Precision

Get and set precision for positive definite matrix routines.

Format *oldprec* = **sysstate(12,prec);**

Input *prec* scalar, 64 or 80.

Output *oldprec* scalar, the original value.

Portability **Windows, UNIX**

This function has no effect under Windows or UNIX. All computations are done in 64-bit precision (except for operations done entirely within the 80x87 on Intel machines).

Remarks The precision will be changed if *prec* is either 64 or 80. Any other number will leave the precision unchanged.

See also **prcsn**

Case 13: LU Tolerance

Get and set singularity tolerance for LU decomposition.

Format *oldtol* = **sysstate(13,tol);**

Input *tol* scalar, new tolerance.

Output *oldtol* scalar, the original tolerance.

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

sysstate

Remarks The tolerance must be ≥ 0 . If *tol* is negative, the tolerance is returned and left unchanged.

See also **croutp, inv**

Case 14: Cholesky Tolerance

Get and set singularity tolerance for Cholesky decomposition.

Format *oldtol* = **sysstate**(14,*tol*);

Input *tol* scalar, new tolerance.

Output *oldtol* scalar, the original tolerance.

Remarks The tolerance must be ≥ 0 . If *tol* is negative, the tolerance is returned and left unchanged.

See also **chol, invpd, solpd**

Case 15: Screen State

Get and set window state as controlled by **screen** command.

Format *oldstate* = **sysstate**(15,*state*);

Input *state* scalar, new window state.

Output *oldstate* scalar, the original window state.

Remarks If *state* = 1, window output is turned on. If *state* = 0, window output is turned off. If *state* = -1, the state is returned unchanged.

See also **screen**

Case 16: Automatic print Mode

Get and set automatic **print** mode.

Format *oldmode* = **sysstate**(16,*mode*);

Input *mode* scalar, mode.

sysstate

Output	<i>oldmode</i>	scalar, original mode.
Remarks	If <i>mode</i> = 1, automatic print mode is turned on. If <i>mode</i> = 0, it is turned off. If <i>mode</i> = -1, the mode is returned unchanged.	
See also	print on/off	
Case 17:	Automatic lprint Mode	
	Get and set automatic lprint mode.	
Format	<i>oldmode</i> = sysstate(17,mode) ;	
Input	<i>mode</i>	scalar, mode.
Output	<i>oldmode</i>	scalar, original mode.
Remarks	If <i>mode</i> = 1, automatic lprint mode is turned on. If <i>mode</i> = 0, it is turned off. If <i>mode</i> = -1, the mode is returned unchanged.	
See also	lprint on/off	
Case 18:	Auxiliary Output	
	Get auxiliary output parameters.	
Format	{ <i>state,name</i> } = sysstate(18,dummy) ;	
Input	<i>dummy</i>	scalar, a dummy argument.
Output	<i>state</i>	scalar, auxiliary output state, 1 - on, 0 - off.
	<i>name</i>	string, auxiliary output filename.
See also	output	
Case 19:	Get/Set Format	
	Get and set format parameters.	
Format	<i>oldfmt</i> = sysstate(19,fmt) ;	

sysstate

	Input	<i>fmt</i>	scalar or 11x1 column vector containing the new format parameters. Usually this will have come from a previous <code>sysstate(19,0)</code> call. See Output for description of matrix.
	Output	<i>oldfmt</i>	11x1 vector containing the current format parameters: [1] format type. [2] justification. [3] sign. [4] leading zero. [5] trailing character. [6] row delimiter. [7] carriage line feed position. [8] automatic line feed for row vectors. [9] field. [10] precision. [11] formatted flag
	Remarks	If <i>fmt</i> is scalar 0, then the format parameters will be left unchanged.	
	See also	format	
	Case 21:	Imaginary Tolerance Get and set the imaginary tolerance.	
	Format	<i>oldtol</i> = sysstate(21,tol) ;	
	Input	<i>tol</i>	scalar, the new tolerance.
	Output	<i>oldtol</i>	scalar, the original tolerance.
	Remarks	The imaginary tolerance is used to test whether the imaginary part of a complex matrix can be treated as zero or not. Functions that are not defined for complex matrices check the imaginary part to see if it can be ignored. The default tolerance is 2.23e-16, or machine epsilon. If <i>tol</i> < 0, the current tolerance is returned.	
	See also	hasimag	

Case 22: Source Path

Get and set the path the compiler will search for source files.

Format *oldpath* = **sysstate**(22,*path*);

Input *path* scalar 0 to get path, or string containing the new path.

Output *oldpath* string, original path.

Remarks If *path* is a matrix, the current source path is returned.

This resets the **src_path** configuration variable. **src_path** is initially defined in the GAUSS configuration file, `gauss.cfg`.

path can list a sequence of directories, separated by semicolons.

Resetting **src_path** affects the path used for subsequent **run** and **compile** statements.

Case 24: Dynamic Library Directory

Get and set the path for the default dynamic library directory.

Format *oldpath* = **sysstate**(24,*path*);

Input *path* scalar 0 to get path, or string containing the new path.

Output *oldpath* string, original path.

Remarks If *path* is a matrix, the current path is returned.

path should list a single directory, not a sequence of directories.

Changing the dynamic library path does not affect the state of any DLL's currently linked to GAUSS. Rather, it determines the directory that will be searched the next time **dlibrary** is called.

UNIX

Changing the path has no effect on GAUSS's default DLL, `libgauss.so`. `libgauss.so` must always be located in the GAUSSHOME directory.

Windows

Changing the path has no effect on GAUSS' default DLL, `gauss.dll`. `gauss.dll` must always be located in the GAUSSHOME directory.

OS/2

sysstate

Changing the path has no effect on GAUSS's default DLL, `gauss.dll`. `gauss.dll` must always be located in the same directory as the GAUSS executable, `gauss.exe`.

DOS

sysstate 24 has no effect, as **dlibrary** and **dllcall** are not supported.

See also **dlibrary**, **dllcall**

Case 25: **Temporary File Path**

Get or set the path GAUSS will use for temporary files..

Format *oldpath* = **sysstate**(25,*path*) ;

Input *path* scalar 0 to get path, or string containing the new path.

Output *oldpath* string, original path.

Remarks If *path* is of type matrix, the path will be returned but not modified.

Case 26: **Interface Mode**

Returns the current interface mode.

Format *mode* = **sysstate**(26,0) ;

Output *mode* scalar, interface mode flag
0 non-X mode
1 terminal (-v) mode
2 X Windows mode

Remarks A mode of 0 indicates that you're running a non-X version of GAUSS; i.e., a version that has no X Windows capabilities. A mode of 1 indicates that you're running an X Windows version of GAUSS, but in terminal mode; i.e., you started GAUSS with the **-v** flag. A mode of 2 indicates that you're running GAUSS in X Windows mode.

Case 28: Random Number Generator Parameters

Get and set the random number generator (RNG) parameters.

sysstate

Format	<i>oldprms</i> = sysstate (28, <i>prms</i>) ;	
Input	<i>prms</i>	scalar 0 to get parameters, or 3x1 matrix of new parameters. [1] seed, 0 < seed < 2^32 [2] multiplier, 0 < mult < 2^32 [3] constant, 0 ≤ const < 2^32
Output	<i>oldprms</i>	3x1 matrix, current parameters.
Portability	Not supported for DOS.	
Remarks	If <i>prms</i> is a scalar 0, the current parameters will be returned without being changed. The modulus of the RNG cannot be changed; it is fixed at 2^32.	
See also	rndcon , rndmult , rndseed , rndns , rndus , rndn , rndu	
Case 30:	Base Year Toggle Specifies whether year value returned by date is to include base year (1900) or not.	
Format	<i>oldstate</i> = sysstate (30, <i>state</i>) ;	
Input	<i>state</i>	scalar, 1, 0, or missing value.
Output	<i>oldstate</i>	scalar, the original state.
Portability	DOS sysstate 30 has no effect. It always returns a 4-digit year.	
Remarks	Internally, date acquires the number of years since 1900. sysstate 30 specifies whether date should add the base year to that value or not. If <i>state</i> = 1, date adds 1900, returning a fully-qualified 4-digit year. If <i>state</i> = 0, date returns the number of years since 1900. If <i>state</i> is a missing value, the current state is returned. The default state is 1.	

system

system

a b c d e f g h i j k l m n o p q r s t u v w x y z	Purpose	Quits GAUSS and returns to the operating system.	
	Format	system; system <i>c</i> ;	
	Input	<i>c</i>	scalar, an optional exit code that can be recovered by the program that invoked GAUSS. The default is 0. Valid arguments are 0-255.
	Remarks	The system command always returns an exit code to the operating system or invoking program. If you don't supply one, it returns 0. This is usually interpreted as indicating success.	
	See also	exec	

tab

Purpose	Tab the cursor to a specified text column.
Format	tab (<i>col</i>); print <i>expr1</i> <i>expr2</i> tab (<i>col1</i>) <i>expr3</i> tab (<i>col2</i>) <i>expr4</i> ...;
Input	<i>col</i> scalar, the column position to tab to.
Remarks	<p><i>col</i> specifies an absolute column position. If <i>col</i> is not an integer, it will be truncated.</p> <p>tab can be called alone or embedded in a print statement. You cannot embed it within a parenthesized expression in a print statement, though. For example:</p> <pre>print (tab(20) c + d * e);</pre> <p>will not give the results you expect. If you have to use parenthesized expressions, write it like this instead:</p> <pre>print tab(20) (c + d * e);</pre>

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

tan

tan

Purpose	Returns the tangent of its argument.
Format	$y = \mathbf{tan}(x);$
Input	x NxK matrix.
Output	y NxK matrix.
Remarks	For real matrices, x should contain angles measured in radians. To convert degrees to radians, multiply the degrees by $\frac{\pi}{180}$.
Example	$\mathbf{let\ x = 0\ .5\ 1\ 1.5;}$ $y = \mathbf{tan(x);}$ $y =$ 0.00000000 0.54630249 1.55740772 14.10141995
See also	atan, pi

tanh

Purpose	Computes the hyperbolic tangent.		
Format	$y = \mathbf{tanh}(x);$		
Input	x	NxK matrix.	
Output	y	NxK matrix containing the hyperbolic tangents of the elements of x .	
Example	<pre>let x = -0.5 -0.25 0 0.25 0.5 1; x = x * pi; y = tanh(x); -1.570796 -0.785398 x = 0.000000 0.785398 1.570796 3.141593 -0.917152 -0.655794 y = 0.000000 0.655794 0.917152 0.996272</pre>		
Source	trig.src		

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

tempname

tempname

a	Purpose	Creates a temporary file with a unique name.
b	Format	<i>tname</i> = tempname (<i>path</i> , <i>pre</i> , <i>suf</i>) ;
c	Input	<i>path</i> string, path where the file will reside. <i>pre</i> string, a prefix to begin the file name with. <i>suf</i> string, a suffix to end the file name with.
d	Output	<i>tname</i> string, unique temporary file name of the form <i>path/pre</i> XXXXnnnnn <i>suf</i> , where XXXX are 4 letters, and nnnnn is the process id of the calling process.
e	Remarks	Any or all of the inputs may be a null string or 0. If <i>path</i> is not specified, the current working directory is used. If unable to create a unique file name of the form requested, tempname returns a null string. WARNING: GAUSS does not remove temporary files created by tempname . It is left to the user to remove them when they are no longer needed.
f		
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

time

Purpose	Returns the current system time.
Format	<code>y = time;</code>
Output	<code>y</code> 4x1 numeric vector, the current time in the order: hours, minutes, seconds, and hundredths of a second.
Example	<pre>print time; 7.000000 31.000000 46.000000 33.000000</pre>
See also	<code>date</code> , <code>datestr</code> , <code>datestring</code> , <code>datestrymd</code> , <code>hsec</code> , <code>timestr</code>

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

timedt

timedt

a	Purpose	Returns system date and time in DT scalar format.
b	Format	<i>dt</i> = timedt ;
c	Output	<i>dt</i> scalar, system date and time in DT scalar format.
d	Remarks	The DT scalar format is a double precision representation of the date and time. In the DT scalar format, the number
e		20010421183207
f		represents 18:32:07 or 6:32:07 PM on April 21, 2001.
g	Source	time.src
h	See also	todaydt , timeutc , dtdate
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

timestr

Purpose	Formats a time in a vector to a string.
Format	<code>ts = timestr(t);</code>
Input	<i>t</i> 4x1 vector from the time function, or a zero. If the input is 0, the time function will be called to return the current system time.
Output	<i>ts</i> 8 character string containing current time in the format hr:mn:sc
Example	<pre>t = { 7, 31, 46, 33 }; ts = timestr(t); print ts;</pre> <p>Produces:</p> <pre>7:31:46</pre>
Source	<code>time.src</code>
See also	date, datestr, datestring, datestrymd, ethsec, etstr, time

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

timeutc

timeutc

a		
b	Purpose	Returns the number of seconds since January 1, 1970 Greenwich Mean Time.
c	Format	<code>tc = timeutc;</code>
d	Output	<code>tc</code> scalar, number of seconds since January 1, 1970 Greenwich Mean Time.
e		
f	Example	<code>tc = timeutc;</code> <code>utv = utctodtv(tc);</code> <code>tc = 939235033</code> <code>utv = 1999 10 6 11 37 13 3 278</code>
g		
h	See also	<code>dtvnormal</code> , <code>utctodtv</code>
i		
j		
k		

title

Purpose	Sets the title for the graph.
Library	pgraph
Format	title(<i>str</i>);
Input	<i>str</i> string, the title to display above the graph.
Global Output	_ptitle
Remarks	Up to three lines of title may be produced by embedding a line feed character (“\L”) in the title string.
Example	<pre>title("First title line\L Second title line\L Third title line");</pre> <p>Fonts may be specified in the title string. For instructions on using fonts, see “Publication Quality Graphics” in the <i>User Guide</i>.</p>
Source	<code>pgraph.src</code>
See also	xlabel, ylabel, fonts

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

tkf2eps

tkf2eps

a	
b	
c	
d	
e	
f	
g	
h	
i	
j	
k	
l	
m	
n	
o	
p	
q	
r	
s	
t	
u	
v	
w	
x y z	

Purpose	Converts a .tkf file to an Encapsulated PostScript file.
Library	pgraph
Format	<i>ret</i> = tkf2eps (<i>tekfile</i> , <i>epsfile</i>);
Input	<i>tekfile</i> string, name of .tkf file <i>epsfile</i> string, name of Encapsulated PostScript file
Output	<i>ret</i> scalar, 0 if successful
Remarks	The conversion is done using the global parameters in peps.dec . You can modify these globally by editing the .dec file, or locally by setting them in your program before calling tkf2eps . See the header of the output Encapsulated PostScript file and a PostScript manual if you want to modify these parameters.

tkf2ps

Purpose	Converts a .tkf file to a PostScript file.
Library	pgraph
Format	<i>ret</i> = tkf2ps (<i>tekfile</i> , <i>psfile</i>);
Input	<i>tekfile</i> string, name of .tkf file <i>epsfile</i> string, name of Encapsulated PostScript file
Output	<i>ret</i> scalar, 0 if successful
Remarks	<p>The conversion is done using the global parameters in peps.dec. You can modify these globally by editing the .dec file, or locally by setting them in your program before calling tkf2ps.</p> <p>See the header of the output Encapsulated PostScript file and a PostScript manual if you want to modify these parameters.</p>

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

tocart

tocart

Purpose	Converts from polar to cartesian coordinates.		
Format	$xy = \text{tocart}(r,theta);$		
Input	r	NxK real matrix, radius.	
	$theta$	LxM real matrix, ExE conformable with r , angle in radians.	
Output	xy	max(N,L) by max(K,M) complex matrix containing the X coordinate in the real part and the Y coordinate in the imaginary part.	
Source	coord.src		

todaydt

Purpose	Returns system date in DT scalar format. The time returned is always midnight (00:00:00), the beginning of the returned day.
Format	<i>dt</i> = todaydt ;
Output	<i>dt</i> scalar, system date in DT scalar format.
Remarks	The DT scalar format is a double precision representation of the date and time. In the DT scalar format, the number 20010421183207 represents 18:32:07 or 6:32:07 PM on April 21, 2001.
Source	time.src
See also	timedt , timeutc , dtdate

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

toeplitz

toeplitz

a	
b	
c	
d	
e	
f	
g	
h	
i	
j	
k	
l	
m	
n	
o	
p	
q	
r	
s	
t	
u	
v	
w	
x y z	

Purpose Creates a Toeplitz matrix from a column vector.

Format `t = toeplitz(x);`

Input `x` Kx1 vector.

Output `t` KxK Toeplitz matrix.

Example `x = seqa(1,1,5);`
 `y = toeplitz(x);`

```

      1
      2
x =   3
      4
      5

      1 2 3 4 5
      2 1 2 3 4
y =   3 2 1 2 3
      4 3 2 1 2
      5 4 3 2 1
```

Source `toeplitz.src`

token

Purpose	Extracts the leading token from a string.		
Format	{ <i>token</i> , <i>str_left</i> } = token (<i>str</i>);		
Input	<i>str</i>	string, the string to parse.	
Output	<i>token</i>	string, the first token in <i>str</i> .	
	<i>str_left</i>	string, str minus <i>token</i> .	
Remarks	<i>str</i> can be delimited with commas or spaces.		
	The advantage of token over parse is that parse is limited to tokens of 8 characters or less; token can extract tokens of any length.		

Example Here is a keyword that uses token to parse its string parameter.

```
keyword add(s);
    local tok,sum;
    sum = 0;
    do until s $== "";
        { tok, s } = token(s);
        sum = sum + stof(tok);
    endo;
    format /rd 1,2;
    print "Sum is: " sum;
endp;
```

If you type:

```
add 1 2 3 4 5 6;
```

add will respond:

```
Sum is: 21.00
```

Source token.src

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

token

See also **parse**

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

topolar

Purpose	Converts from cartesian to polar coordinates.	
Format	$\{ r, \theta \} = \text{topolar}(xy);$	
Input	xy	NxK complex matrix containing the X coordinate in the real part and the Y coordinate in the imaginary part.
Output	r	NxK real matrix, radius.
	θ	NxK real matrix, angle in radians.
Source	coord.src	

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

trace

trace

Purpose	Allows the user to trace program execution for debugging purposes.		
Format	trace <i>new</i> ; trace <i>new</i> , <i>mask</i> ;		
Input	<i>new</i>	scalar,	new value for trace flag.
	<i>mask</i>	scalar,	optional mask to allow leaving some bits of the trace flag unchanged.
Remarks	The trace command has no effect unless you are running your program under GAUSS's source level debugger. Setting the trace flag will not generate any debugging output during normal execution of a program.		
	The argument is converted to a binary integer with the following meanings:		
	bit	decimal	meaning
	ones	1	trace calls/returns
	twos	2	trace line numbers
	fours	4	verbose trace
	eights	8	output to window
	sixteens	16	output to print
	thirty-twos	32	output to auxiliary output
	sixty-fours	64	output to error log
	You must set one or more of the output bits to get any output from trace . If you set trace to 4, you'll be doing a verbose trace of your program, but the output won't be displayed anywhere.		
	The trace output as a program executes will be as follows:		
	(+GRAD)	calling function or procedure GRAD	
	(-GRAD)	returning from GRAD	
	[47]	executing line 47	
	Note that the line number trace will only produce output if the program was compiled with line number records.		

trace

To set a single bit use two arguments:

trace 16,16; turn on output to printer

trace 0,16; turn off output to printer

Example	<code>trace 1+8;</code>	trace fn/proc calls/returns to standard output
	<code>trace 2+8;</code>	trace line numbers to standard output
	<code>trace 1+2+8;</code>	trace line numbers and fn/proc calls/returns to standard output
	<code>trace 1+16;</code>	trace fn/proc calls/returns to printer
	<code>trace 2+16;</code>	trace line numbers to printer
	<code>trace 1+2+16;</code>	trace line numbers and fn/proc calls/returns to printer
	<code>trace 4+8;</code>	verbose trace to screen

See also **#lineson**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

trap

trap

a		
b		
c		
d		
e		
f		
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

Purpose	Sets the trap flag to enable or disable trapping of numerical errors.	
Format	trap <i>new</i> ; trap <i>new</i> , <i>mask</i> ;	
Input	<i>new</i>	scalar, new trap value.
	<i>mask</i>	scalar, optional mask to allow leaving some bits of the trap flag unchanged.
Remarks	<p>The trap flag is examined by some functions to control error handling. There are 16 bits in the trap flag, but most GAUSS functions will examine only the lowest order bit:</p> <p>trap 1; turn trapping on trap 0; turn trapping off</p> <p>If we extend the use of the trap flag, we will use the lower order bits of the trap flag. It would be wise for you to use the highest 8 bits of the trap flag if you create some sort of user-defined trap mechanism for use in your programs. (See the function trapchk for detailed instructions on testing the state of the trap flag; see error for generating user-defined error codes.)</p> <p>To set only one bit and leave the others unchanged use two arguments:</p> <p>trap 1,1; set the ones bit trap 0,1; clear the ones bit</p>	
Example	<pre>x = eye(3); oldval = trapchk(1); trap 1,1; y = inv(x); trap oldval,1;</pre>	

trap

```
if scalerr(y);  
    errorlog "WARNING: x is singular";  
else;  
    print "y" y;  
endif;
```

In this example the result of **inv** is trapped in case **x** is singular. The trap state is reset to the original value after the call to **inv**.

Run the example

```
x = eye(3);
```

It is inverted.

Now try

```
ones(3,3);
```

It isn't.

See also **scalerr**, **trapchk**, **error**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

trapchk

trapchk

a	Purpose	Tests the value of the trap flag.
b	Format	$y = \text{trapchk}(m);$
c	Input	m scalar mask value.
d	Output	y scalar which is the result of the bitwise logical AND of the trap flag and the mask.
e	Remarks	To check the various bits in the trap flag, add the decimal values for the bits you wish to check according to the chart below and pass the sum in as the argument to the trapchk function:
f		
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

bit	decimal value
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024
11	2048
12	4096
13	8192
14	16384
15	32768

If you want to test if either bit 0 or bit 8 is set, then pass an argument of 1+256 or 257 to **trapchk**. The following table demonstrates values that will be returned for:

```
y=trapchk(257);
```

trapchk

	0	1	value of bit 0 in trap flag
0	0	1	
1	256	257	
value of bit 8 in trap flag			

GAUSS functions that test the trap flag currently test only bits 0 and 1.

See also **scalerr, trap, error**

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

trimr

trimr

a		
b	Purpose	Trims rows from the top and/or bottom of a matrix.
c	Format	$y = \text{trimr}(x,t,b);$
d	Input	x NxK matrix from which rows are to be trimmed.
e		t scalar containing the number of rows which are to be removed from the top of x .
f		b scalar containing the number of rows which are to be removed from the bottom of x .
g		
h	Output	y RxK matrix where $R=N-(t+b)$ containing the rows left after the trim.
i		
j	Remarks	If either t or b is zero, then no rows will be trimmed from that end of the matrix.
k	Example	$x = \text{rndu}(5,3);$
l		$y = \text{trimr}(x,2,1);$
m		0.76042751 0.33841579 0.01844780
n		0.05334503 0.38939785 0.65029973
o		$x =$ 0.93077511 0.06961078 0.04207563
p		0.53640701 0.06640062 0.07222560
q		0.14084669 0.06033813 0.69449247
r		$y =$ 0.93077511 0.06961078 0.04207563
s		0.53640701 0.06640062 0.07222560
t	See also	submat, rotater, shiftr
u		
v		
w		
x y z		

trunc

Purpose Converts numbers to integers by truncating the fractional portion.

Format $y = \text{trunc}(x);$

Input x NxK matrix.

Output y NxK matrix containing the truncated elements of x .

Example $x = 100 * \text{rndn}(2, 2);$

$$x = \begin{bmatrix} 77.68 & -14.10 \\ 4.73 & -158.88 \end{bmatrix}$$

$$y = \text{trunc}(x);$$

$$y = \begin{bmatrix} 77.00 & -14.00 \\ 4.00 & -158.00 \end{bmatrix}$$

See also **ceil, floor, round**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

type

type

a	Purpose	Returns the symbol table type of the argument.
b	Format	<code>t = type(x);</code>
c	Input	<code>x</code> matrix or string, can be an expression.
d	Output	<code>t</code> scalar. 6 matrix 13 string 15 string array
e	Remarks	type returns the type of a single symbol. The related function typecv will take a character vector of symbol names and return a vector of either their types or the missing value code for any that are undefined. type works for matrices, strings, and string arrays; typecv works for user-defined procedures, keywords and functions as well. type works for global or local symbols; typecv works only for global symbols.
f	Example	<code>k = {"CHARS"};</code> <code>print k;</code> <code>if type(k) == 6;</code> <code> k = "\$"+k;</code> <code>endif;</code> <code>print k;</code> produces +DEN CHARS
g	See also	typecv , typef
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

typecv

Purpose	Returns the symbol table type of objects whose names are given as a string or as elements of a character vector.
Format	<code>y = typecv(x);</code>
Input	<i>x</i> string or Nx1 character vector which contains the names of variables whose type is to be determined.
Output	<i>y</i> scalar or Nx1 vector containing the types of the respective symbols in <i>x</i> .
Remarks	<p>The values returned by typecv for the various variable types are as follows:</p> <ul style="list-style-type: none"> 6 Matrix (Numeric, Character, or Mixed) 8 Procedure (proc) 9 Function (fn) 5 Keyword (keyword) 13 String 15 String Array <p>It will return the GAUSS missing value code if the symbol is not found, so typecv may be used to determine if a symbol is defined or not.</p>
Example	<pre> xvar = sqrt(5); yvar = "Montana"; fn area(r) = pi*r*r; let names = xvar yvar area; y = typecv(names); XVAR names = YVAR AREA </pre>

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

typecv

$$y = \begin{matrix} 6 \\ 13 \\ 9 \end{matrix}$$

See also **type, typeof, varput, varget**

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

typedef

Purpose	Returns the type of data (the number of bytes per element) in a GAUSS data set.						
Format	<code>y = typedef(fp);</code>						
Input	<i>fp</i> scalar, file handle of an open file.						
Output	<i>y</i> scalar, type of data in GAUSS data set.						
Remarks	<p>If <i>fp</i> is a valid GAUSS file handle, then <i>y</i> will be set to the type of the data in the file as follows:</p> <table> <tr> <td>2</td><td>2-byte signed integer</td></tr> <tr> <td>4</td><td>4-byte IEEE floating point</td></tr> <tr> <td>8</td><td>8-byte IEEE floating point</td></tr> </table>	2	2-byte signed integer	4	4-byte IEEE floating point	8	8-byte IEEE floating point
2	2-byte signed integer						
4	4-byte IEEE floating point						
8	8-byte IEEE floating point						
Example	<pre>infile = "dat1"; outfile = "dat2"; open fin = ^infile; names = getname(infile); create fout = ^outfile with ^names,0,typedef(fin);</pre> <p>In this example a file <code>dat2.dat</code> is created which has the same variables and variable type as the input file, <code>dat1.dat</code>. typedef is used to return the type of the input file for the create statement.</p>						
See also	colsf , rowsf						

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

union

union

a		
b	Purpose	Returns the union of two vectors with duplicates removed.
c	Format	$y = \text{union}(v1,v2,flag);$
d	Input	$v1$ Nx1 vector. $v2$ Mx1 vector. $flag$ scalar, 1 if numeric data, 0 if character.
e	Output	y Lx1 vector containing all unique values that are in $v1$ and $v2$, sorted in ascending order.
f	Remarks	The combined elements of $v1$ and $v2$ must fit into a single vector.
g	Example	<pre>let v1 = mary jane linda john; let v2 = mary sally; x = union(v1,v2,0); JANE JOHN x = LINDA MARY SALLY</pre>
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

uniqindx

Purpose	Computes the sorted index of x , leaving out duplicate elements.		
Format	$index = \text{uniqindx}(x, flag);$		
Input	x	Nx1 or 1xN vector.	
	$flag$	scalar, 1 if numeric data, 0 if character.	
Output	$index$	Mx1 vector, indices corresponding to the elements of x sorted in ascending order with duplicates removed.	
Remarks	Among sets of duplicates it is unpredictable which elements will be indexed.		
Example	<pre>let x = 5 4 4 3 3 2 1;</pre>		
	<pre>ind = uniqindx(x,1);</pre>		
	<pre>y = x[ind];</pre>		
		7	
		6	
	$ind =$	5	
		2	
		1	
		1	
		2	
	$y =$	3	
		4	
		5	

unique

unique

a		
b	Purpose	Sorts and removes duplicate elements from a vector.
c	Format	$y = \text{unique}(x, flag);$
d	Input	x Nx1 or 1xN vector.
e		$flag$ scalar, 1 if numeric data, 0 if character.
f	Output	y Mx1 vector, sorted x with the duplicates removed.
g	Example	$let\ x = 5\ 4\ 4\ 3\ 3\ 2\ 1;$
h		$y = unique(x, 1);$
i		1
j		2
k	$y =$	3
l		4
m		5
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

upmat, upmat1

Purpose Returns the upper portion of a matrix. **upmat** returns the main diagonal and every element above. **upmat1** is the same except it replaces the main diagonal with ones.

Format $u = \text{upmat}(x);$
 $u = \text{upmat1}(x);$

Input x NxK matrix.

Output u NxK matrix containing the upper elements of the matrix. The lower elements are replaced with zeros. **upmat** returns the main diagonal intact. **upmat1** replaces the main diagonal with ones.

Example $x = \begin{Bmatrix} 1 & 2 & -1, \\ 2 & 3 & -2, \\ 1 & -2 & 1 \end{Bmatrix};$

$u = \text{upmat}(x);$

$u1 = \text{upmat1}(x);$

The resulting matrices are

$u = \begin{Bmatrix} 1 & 2 & -1 \\ 0 & 3 & -2 \\ 0 & 0 & 1 \end{Bmatrix}$

$u1 = \begin{Bmatrix} 1 & 2 & -1 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{Bmatrix}$

Source diag.src

See also **lowmat, lowmat1, diag, diagrv, crout**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

upper

upper

a	Purpose	Converts a string or matrix of character data to uppercase.
b	Format	$y = \text{upper}(x);$
c	Input	x string or NxK matrix containing the character data to be converted to uppercase.
d	Output	y string or NxK matrix containing the uppercase equivalent of data in x .
e	Remarks	If x is a numeric matrix, y will contain garbage. No error message will be generated since GAUSS does not distinguish between numeric and character data in matrices.
f	Example	$x = \text{"uppercase"};$ $y = \text{upper}(x);$ $y = \text{UPPERCASE}$
g	See also	lower
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

use

Purpose	Loads a compiled file at the beginning of the compilation of a source program.
Format	use <i>fname</i> ;
Input	<i>fname</i> literal or ^string, the name of a compiled file created using the compile or the saveall command.
Remarks	<p>The use command can be used ONCE at the TOP of a program to load in a compiled file which the rest of the program will be added to. In other words, if <i>xy.e</i> had the following lines:</p> <pre>library pgraph; external proc xy; x = seqa(0.1,0.1,100);</pre> <p>It could be compiled to <i>xy.gcg</i>. Then the following program could be run:</p> <pre>use xy; xy(x,sin(x));</pre> <p>Which would be equivalent to:</p> <pre>new; x = seqa(0.1,0.1,100); xy(x,sin(x));</pre> <p>The use command can be used at the top of files that are to be compiled with the compile command. This can greatly shorten compile time for a set of closely related programs. For example:</p> <pre>library pgraph; external proc xy,logx,logy,loglog,hist; saveall pgraph;</pre> <p>This would create a file called <i>pgraph.gcg</i> containing all the procedures, strings and matrices needed to run PQG programs. Other</p>

use

programs could be compiled very quickly with the following statement at the top of each:

```
use pgraph;
```

or the same statement could be executed once, for instance from the command prompt, to instantly load all the procedures for PQG.

When the compiled file is loaded with **use**, all previous symbols and procedures are deleted before the program is loaded. It is therefore unnecessary to execute a **new** before **use**'ing a compiled file.

use can appear only ONCE at the TOP of a program.

See also **compile, run, saveall**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

utctodt

Purpose	Converts UTC scalar format to DT scalar format.
Format	<code>dt = utctodt(utc);</code>
Input	<i>utc</i> Nx1 vector, UTC scalar format.
Output	<i>dt</i> Nx1 matrix, DT scalar format.
Remarks	A UTC scalar gives the number of seconds since or before January 1, 1970 Greenwich Mean Time. In DT scalar format, 11:06:47 on March 15, 2001 is 20010315110647.
Example	<pre>tc = 985633642; print "tc = " tc; dt = utctodt(tc); print "dt = " dt;</pre> <p>produces:</p> <pre>tc = 985633642 dt = 20010326110722</pre>
Source	time.src
See also	dtvnormal , timeutc , utctodtv , dttdodtv , dtvtodt , dttoutc , dtvtodt , strtodt , dttostr

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

utctodtv

utctodtv

a	Purpose	Converts UTC scalar format to DTV vector format.
b	Format	<i>dtv</i> = utctodtv (<i>utc</i>) ;
c	Input	<i>utc</i> Nx1 vector, UTC scalar format.
d	Output	<i>dtv</i> Nx8 matrix, DTV vector format.
e	Remarks	A UTC scalar gives the number of seconds since or before January 1, 1970 Greenwich Mean Time.
f		Each row of <i>dtv</i> , in DTV vector format, contains:
g		[N,1] Year
h		[N,2] Month in Year, 1-12
i		[N,3] Day of month, 1-31
j		[N,4] Hours since midnight, 0-23
k		[N,5] Minutes, 0-59
l		[N,6] Seconds, 0-59
m		[N,7] Day of week, 0-6, 0 = Sunday
n		[N,8] Days since Jan 1 of current year, 0-365
o	Example	<pre>tc = timeutc; print "tc = " tc; dtv = utctodtv(tc); print "dtv = " dtv;</pre> <p>produces:</p> <pre>tc = 985633642 dtv = 2001 3 26 11 7 22 1 84</pre>
p	Source	time.src
q	See also	dtvnormal , timeutc , utctodt , dttodtv , dttoutc , dtvtodt , dtvtoutc , strtodt , dttostr
r		
s		
t		
u		
v		
w		
x y z		

utrisol

Purpose	Computes the solution of $Ux = b$ where U is an upper triangular matrix.	
Format	$x = \text{utrisol}(b,U);$	
Input	b	PxK matrix.
	U	PxP upper triangular matrix.
Output	x	PxK matrix.
Remarks	utrisol applies a back solve to $Ux = b$ to solve for x . If b has more than one column, each column is solved for separately, i.e., utrisol applies a back solve to $Ux[:,i] = b[:,i]$.	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

vals

vals

a	Purpose	Converts a string into a matrix of its ASCII values.
b	Format	<code>y = vals(s);</code>
c	Input	<code>s</code> string of length N where N > 0.
d	Output	<code>y</code> Nx1 matrix containing the ASCII values of the characters in the string <code>s</code> .
e	Remarks	If the string is null, the function will fail and an error message will be given.
f	Example	<code>k0:</code> <code> k = key;</code> <code> if not k;</code> <code> goto k0;</code> <code> endif;</code> <code> if k == vals("Y") or k == vals("y");</code> <code> goto doit;</code> <code> else;</code> <code> end;</code> <code> endif;</code> <code>doit:</code> In this example the key function is used to read the keyboard. When key returns a nonzero value, meaning a key has been pressed, the ASCII value it returns is tested to see if it is an uppercase or lowercase "Y". If it is, the program will jump to the label doit , otherwise the program will end.
g	See also	chrs, ftos, stof
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

varget

Purpose	Accesses a global variable whose name is given as a string argument.	
Format	<code>y = varget(s);</code>	
Input	<i>s</i>	string containing the name of the global symbol you wish to access.
Output	<i>y</i>	contents of the matrix or string whose name is in <i>s</i> .
Remarks	This function searches the global symbol table for the symbol whose name is in <i>s</i> and returns the contents of the variable if it exists. If the symbol does not exist, the function will terminate with an Undefined symbol error message. If you want to check to see if a variable exists before using this function, use typecv .	
Example	<pre>dog = rndn(2,2); y = varget("dog"); dog = -0.83429985 0.34782433 0.91032546 1.75446391 y = -0.83429985 0.34782433 0.91032546 1.75446391</pre>	
See also	typecv, varput	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

varget1

varget1

Purpose	Accesses a local variable whose name is given as a string argument.	
Format	<code>y = varget1(s);</code>	
Input	<i>s</i>	string containing the name of the local symbol you wish to access.
Output	<i>y</i>	contents of the matrix or string whose name is in <i>s</i> .
Remarks	This function searches the local symbol list for the symbol whose name is in <i>s</i> and returns the contents of the variable if it exists. If the symbol does not exist, the function will terminate with an Undefined symbol error message.	
Example	<pre>proc dog; local x,y; x = rndn(2,2); y = varget1("x"); print "x" x; print "y" y; retp(y); endp; z = dog; print "z" z;</pre>	

varget1

Produces:

x	-0.543851	-0.181701
	-0.108873	0.0648738
y	-0.543851	-0.181701
	-0.108873	0.0648738
z	-0.543851	-0.181701
	-0.108873	0.0648738

See also **varput1**

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

varmall

varmall

Purpose	Computes log-likelihood of a Vector ARMA model.		
Format	<i>res</i> = varmall (<i>w</i> , <i>phi</i> , <i>theta</i> , <i>vc</i>) ;		
Input	<i>w</i>	NxK matrix, time series.	
	<i>phi</i>	K*PxK matrix, AR coefficient matrices.	
	<i>theta</i>	K*QxK matrix, MA coefficient matrices.	
	<i>vc</i>	KxK matrix, covariance matrix.	
Output	<i>ll</i>	scalar, log-likelihood. If the calculation fails <i>res</i> is set to missing value with error code:	
		Error Code	Reason for Failure
		1	$M < 1$
		2	$N < 1$
		3	$P < 0$
		4	$Q < 0$
		5	$P = 0$ and $Q = 0$
		7	floating point work space too small
		8	integer work space too small
		9	qq is not positive definite
		10	AR parameters too close to stationarity boundary
		11	model not stationary
		12	model not invertible
		13	$I+M'H'HM$ not positive definite
Remarks	varmall is adapted from code developed by Jose Alberto Mauricio of the Universidad Complutense de Madrid. It was published as Algorithm AS311 in Applied Statistics. Also described in “Exact Maximum Likelihood Estimation of Stationary Vector ARMA Models,” JASA, 90:282-264.		

varmares

Purpose	Computes residuals of a Vector ARMA model.	
Format	<i>res</i> = varmares (<i>w</i> , <i>phi</i> , <i>theta</i>) ;	
Input	<i>w</i>	NxK matrix, time series.
	<i>phi</i>	K*PxK matrix, AR coefficient matrices.
	<i>theta</i>	K*QxK matrix, MA coefficient matrices.
Output	<i>res</i>	NxK matrix, residuals. If the calculation fails <i>res</i> is set to missing value with error code:

Error Code	Reason for Failure
1	M < 1
2	N < 1
3	P < 0
4	Q < 0
5	P = 0 and Q = 0
7	floating point work space too small
8	integer work space too small
9	qq is not positive definite
10	AR parameters too close to stationarity boundary
11	model not stationary
12	model not invertible
13	I+M'H'HM not positive definite

Remarks	varmares is adapted from code developed by Jose Alberto Mauricio of the Universidad Complutense de Madrid. It was published as Algorithm AS311 in Applied Statistics. Also described in “Exact Maximum Likelihood Estimation of Stationary Vector ARMA Models,” JASA, 90:282-264.
---------	--

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

varput

varput

Purpose	Allows a matrix or string to be assigned to a global symbol whose name is given as a string argument.		
Format	$y = \text{varput}(x,n);$		
Input	x	NxK matrix or string which is to be assigned to the target variable.	
	n	string containing the name of the global symbol which will be the target variable.	
Output	y	scalar, 1 if the operation is successful and 0 if the operation fails.	
Remarks	x and n may be global or local. The variable, whose name is in n , that x is assigned to is always a global.		
	If the function fails, it will be because the global symbol table is full.		
	This function is useful for returning values generated in local variables within a procedure to the global symbol table.		
Example	$\text{source} = \text{rndn}(2,2);$		
	$\text{targname} = \text{"target"};$		
	$\text{if not varput}(\text{source},\text{targname});$		
	$\text{print "Symbol table full"};$		
	$\text{end};$		
	$\text{endif};$		
	$\text{source} =$		
	$-0.93519984 \quad 0.40642598$		
	$-0.36867581 \quad 2.57623519$		
	$\text{target} =$		
	$-0.93519984 \quad 0.40642598$		
	$-0.36867581 \quad 2.57623519$		
See also	varget, typecv		

varputl

Purpose	Allows a matrix or string to be assigned to a local symbol whose name is given as a string argument.	
Format	<code>y = varputl(x,n);</code>	
Input	<i>x</i>	NxK matrix or string which is to be assigned to the target variable.
	<i>n</i>	string containing the name of the local symbol which will be the target variable.
Output	<i>y</i>	scalar, 1 if the operation is successful and 0 if the operation fails.
Remarks	<i>x</i> and <i>n</i> may be global or local. The variable, whose name is in <i>n</i> , that <i>x</i> is assigned to is always a local.	
Example	<pre> proc dog(x); local a,b,c,d,e,vars,putvar; a=1;b=2;c=3;d=5;e=7; vars = { a b c d e }; putvar = 0; do while putvar \$/= vars; print "Assign x (" \$vars ")": " ;; putvar = upper(cons); print; endo; call varputl(x,putvar); retp(a+b*c-d/e); endp; format /rds 2,1; i = 0; </pre>	

varput1

```
do until i ≥ 5;
    z = dog(17);
    print " z is " z;
    i = i + 1;
endo;
```

Produces:

```
Assign x ( A B C D E ): a
    z is 22.3
Assign x ( A B C D E ): b
    z is 51.3
Assign x ( A B C D E ): c
    z is 34.3
Assign x ( A B C D E ): d
    z is 4.6
Assign x ( A B C D E ): e
    z is 6.7
```

See also **varget1**

vartype

Purpose	Returns a vector of ones and zeros that indicate whether variables in a data set are character or numeric.
Format	<code>y = vartype(names);</code>
Input	<i>names</i> Nx1 character vector of variable names retrieved from a data set header file with the getname function.
Output	<i>y</i> Nx1 vector of ones and zeros, 1 if variable is numeric, 0 if character.
Remarks	This function is being obsoleted. See vartypef . If a variable name in <i>names</i> is lowercase, a 0 will be returned in the corresponding element of the returned vector.
Example	<pre>names = getname("freq"); y = vartype(names); print \$names; print y;</pre> <div> <div>AGE</div> <div>PAY</div> <div>sex</div> <div>WT</div> <div>1.0000000</div> <div>1.0000000</div> <div>0.0000000</div> <div>1.0000000</div> </div>
Source	<code>vartype.src</code>

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

vartypef

vartypef

Purpose	Returns a vector of ones and zeros that indicate whether variables in a data set are character or numeric.		
Format	$y = \text{vartypef}(f);$		
Input	f	file handle of an open file.	
Output	y	Nx1 vector of ones and zeros, 1 if variable is numeric, 0 if character.	
Remarks	This function should be used in place of older functions that are based on the case of the variable names. You should also use the v96 data set format.		

vcm, vcx

Purpose	Computes a variance-covariance matrix.	
Format	<code>vc = vcm(m);</code> <code>vc = vcx(x);</code>	
Input	<i>m</i>	KxK moment (x'x) matrix. A constant term MUST have been the first variable when the moment matrix was computed.
	<i>x</i>	NxK matrix of data.
Output	<i>vc</i>	KxK variance-covariance matrix.
Source	<code>corr.src</code>	
See also	momentd	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

vec, vecr

vec, vecr

a	Purpose	Creates a column vector by appending the columns/rows of a matrix to each other.	
b			
c	Format	$yc = \text{vec}(x);$	
d		$yr = \text{vecr}(x);$	
e	Input	x NxK matrix.	
f			
g	Output	yc (N*K)x1 vector, the columns of x appended to each other.	
h		yr (N*K)x1 vector, the rows of x appended to each other and the result transposed.	
i	Remarks	vecr is much faster.	
j	Example	$x = \begin{Bmatrix} 1 & 2, \\ 3 & 4 \end{Bmatrix};$	
k		$yc = \text{vec}(x);$	
l		$yr = \text{vecr}(x);$	
m			
n		$x = \begin{matrix} 1.000000 & 2.000000 \\ 3.000000 & 4.000000 \end{matrix}$	
o			
p		$yc = \begin{matrix} 1.000000 \\ 3.000000 \\ 2.000000 \\ 4.000000 \end{matrix}$	
q			
r			
s		$yr = \begin{matrix} 1.000000 \\ 2.000000 \\ 3.000000 \\ 4.000000 \end{matrix}$	
t			
u			
v			

vech

Purpose	Vectorizes a symmetric matrix by retaining only the lower triangular portion of the matrix.	
Format	$v = \mathbf{vech}(x);$	
Input	x	$N \times N$ symmetric matrix.
Output	v	$(N*(N+1)/2) \times 1$ vector, the lower triangular portion of the matrix x .
Remarks	As you can see from the example below, vech will not check to see if x is symmetric. It just packs the lower triangular portion of the matrix into a column vector in row-wise order.	
Example	<pre> x = seqa(10,10,3) + seqa(1,1,3)'; v = vech(x); sx = xpnd(v); 11 12 13 x = 21 22 23 31 32 33 11 21 v = 22 31 32 33 11 21 31 sx = 21 22 32 31 32 33 </pre>	

See also **xpnd**

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

vector (dataloop)

vector (dataloop)

a		
b		
c		
d		
e		
f		
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		
	Purpose	Specifies the creation of a new variable within a data loop.
	Format	vector <code>[[#]] numvar = numeric_expression;</code> vector <code>\$ charvar = character_expression;</code>
	Remarks	<p>A <i>numeric_expression</i> is any valid expression returning a numeric value. A <i>character_expression</i> is any valid expression returning a character value. If neither ‘\$’ nor ‘#’ is specified, ‘#’ is assumed.</p> <p>vector is used in place of make when the expression returns a scalar rather than a vector. vector forces the result of such an expression to a vector of the correct length. vector could actually be used anywhere that make is used, but would generate slower code for expressions that already return vectors.</p> <p>Any variables referenced must already exist, either as elements of the source data set, as externs, or as the result of a previous make, vector, or code statement.</p>
	Example	<code>vector const = 1;</code>
	See also	make

vget

Purpose	Extracts a matrix or string from a data buffer constructed with vput .	
Format	{ <i>x</i>,<i>dbufnew</i> } = vget(<i>dbuf</i>,<i>name</i>);	
Input	<i>dbuf</i>	Nx1 vector, a data buffer containing various strings and matrices.
	<i>name</i>	string, the name of the string or matrix to extract from <i>dbuf</i> .
Output	<i>x</i>	LxM matrix or string, the item extracted from <i>dbuf</i> .
	<i>dbufnew</i>	Kx1 vector, the remainder of <i>dbuf</i> after <i>x</i> has been extracted.
Source	vpack.src	
See also	vlist, vput, vread	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

view

view

a		
b	Purpose	Sets the position of the observer in workbox units for 3-D plots.
c	Library	pgraph
d	Format	view(x,y,z) ;
e	Input	<i>x</i> scalar, the X position in workbox units.
f		<i>y</i> scalar, the Y position in workbox units.
g		<i>z</i> scalar, the Z position in workbox units.
h	Remarks	The size of the workbox is set with volume . The viewer must be outside of the workbox. The closer the position of the observer, the more perspective distortion there will be. If $x = y = z$, the projection will be isometric.
i		If view is not called, a default position will be calculated.
j		Use viewxyz to locate the observer in plot coordinates.
k		
l	Source	<code>pgraph.src</code>
m	See also	volume, viewxyz
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

viewxyz

Purpose	Sets the position of the observer in plot coordinates for 3-D plots.		
Library	pgraph		
Format	viewxyz (<i>x,y,z</i>) ;		
Input	<i>x</i>	scalar, the X position in plot coordinates.	
	<i>y</i>	scalar, the Y position in plot coordinates.	
	<i>z</i>	scalar, the Z position in plot coordinates.	
Remarks	The viewer must be outside of the workbox. The closer the observer, the more perspective distortion there will be.		
	If viewxyz is not called, a default position will be calculated.		
	Use view to locate the observer in workbox units.		
Source	pgraph.src		
See also	volume , view		

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

vlist

vlist

a	Purpose	Lists the contents of a data buffer constructed with vput .
b	Format	vlist (<i>dbuf</i>);
c	Input	<i>dbuf</i> Nx1 vector, a data buffer containing various strings and matrices.
d	Remarks	vlist lists the names of all the strings and matrices stored in <i>dbuf</i> .
e	Source	vpack.src
f	See also	vget , vput , vread
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

vnamecv

Purpose	Returns the names of the elements of a data buffer constructed with vput .	
Format	<i>cv</i> = vnamecv (<i>dbuf</i>) ;	
Input	<i>dbuf</i>	Nx1 vector, a data buffer containing various strings and matrices.
Output	<i>cv</i>	Kx1 character vector containing the names of the elements of <i>dbuf</i> .
See also	vget , vput , vread , vtypecv	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

volume

volume

a		
b	Purpose	Sets the length, width, and height ratios of the 3-D workbox.
c	Library	pgraph
d	Format	volume (<i>x,y,z</i>) ;
e	Input	<i>x</i> scalar, the X length of the 3-D workbox.
f		<i>y</i> scalar, the Y length of the 3-D workbox.
g		<i>z</i> scalar, the Z length of the 3-D workbox.
h	Remarks	The ratio between these values is what is important. If volume is not called, a default workbox will be calculated.
i	Source	<code>pgraph.src</code>
j	See also	view
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

vput

Purpose	Inserts a matrix or string into a data buffer.	
Format	<i>dbufnew</i> = vput (<i>dbuf</i> , <i>x</i> , <i>xname</i>) ;	
Input	<i>dbuf</i>	Nx1 vector, a data buffer containing various strings and matrices. If <i>dbuf</i> is a scalar 0, a new data buffer will be created.
	<i>x</i>	LXM matrix or string, item to be inserted into <i>dbuf</i> .
	<i>xname</i>	string, the name of <i>x</i> , will be inserted with <i>x</i> into <i>dbuf</i> .
Output	<i>dbufnew</i>	Kx1 vector, the data buffer after <i>x</i> and <i>xname</i> have been inserted.
Remarks	If <i>dbuf</i> already contains <i>x</i> , the new value of <i>x</i> will replace the old one.	
Source	vpack.src	
See also	vget , vlist , vread	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

vread

vread

Purpose	Reads a string or matrix from a data buffer constructed with vput .	
Format	$x = \mathbf{vread}(dbuf,xname) ;$	
Input	$dbuf$	Nx1 vector, a data buffer containing various strings and matrices.
	$xname$	string, the name of the matrix or string to read from $dbuf$.
Output	x	LxM matrix or string, the item read from $dbuf$.
Remarks	vread , unlike vget , does not change the contents of $dbuf$. Reading x from $dbuf$ does not remove it from $dbuf$.	
Source	vpack.src	
See also	vget , vlist , vput	

vtypecv

Purpose	Returns the types of the elements of a data buffer constructed with vput .	
Format	<i>cv</i> = vtypecv (<i>dbuf</i>);	
Input	<i>dbuf</i>	Nx1 vector, a data buffer containing various strings and matrices.
Output	<i>cv</i>	Kx1 character vector containing the types of the elements of <i>dbuf</i> .
See also	vget , vput , vread , vnamecv	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

wait, waitc

wait, waitc

a	Purpose	Waits until any key is pressed.
b		
c	Format	wait; waitc;
d		
e	Remarks	If you are working in terminal mode, they don't "see" any keystrokes until ENTER is pressed. waitc clears any pending keystrokes before waiting until another key is pressed.
f		
g	Source	wait.src, waitc.src
h	See also	pause

window

Purpose	Partitions the window into tiled regions of equal size.	
Library	pgraph	
Format	window (<i>row,col,typ</i>) ;	
Input	<i>row</i>	scalar, number of rows of graphic panels.
	<i>col</i>	scalar, number of columns of graphic panels.
	<i>typ</i>	scalar, graphic panel attribute type. If 1, the graphic panels will be transparent, if 0, the graphic panels will be nontransparent (blanked).
Remarks	<p>The graphic panels will be numbered from 1 to (<i>row</i>)x(<i>col</i>) starting from the left topmost graphic panel and moving right.</p> <p>See makewind for creating graphic panels of a specific size and position. (For more information, see “Publication Quality Graphics” in the <i>User Guide</i>.)</p>	
Source	pwindow.src	
See also	endwind , begwind , setwind , nextwind , getwind , makewind	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

writer

writer

a		
b		
c		
d		
e		
f		
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		
	Purpose	Writes a matrix to a GAUSS data set.
	Format	$y = \text{writer}(fh,x);$
	Input	fh handle of the file that data is to be written to. x NxK matrix.
	Output	y scalar specifying the number of rows of data actually written to the data set.
	Remarks	<p>The file must have been opened with create, open for append, or open for update.</p> <p>The data in x will be written to the data set whose handle is fh starting at the current pointer position in the file. The pointer position in the file will be updated so the next call to writer will put the next block of data after the first block. (See open and create for the initial pointer positions in the file for reading and writing.)</p> <p>x must have the same number of columns as the data set. colsf returns the number of columns in a data set.</p> <p>writer returns the number of rows actually written to the data set. If y does not equal rows(x), the disk is probably full.</p> <p>If the data set is not double precision, the data will be rounded to nearest as it is written out.</p> <p>If the data contain character elements, the file must be double precision or the character information will be lost.</p> <p>If the file being written to is the 2-byte integer data type, then missing values will be written out as -32768. These will not automatically be converted to missings on input. They can be converted with the miss function:</p> <p>$x = \text{miss}(x,-32768);$</p> <p>Trying to write complex data to a data set that was originally created to store real data will cause a program to abort with an error message. (See create for details on creating a complex data set.)</p>

Example `create fp = data with x,10,8;`
 `if fp == -1;`
 `errorlog "Can't create output file";`
 `end;`
 `endif;`
 `c = 0;`
 `do until c ≥ 10000;`
 `y = rndn(100,10);`
 `k = writer(fp,y);`
 `if k /= rows(y);`
 `errorlog "Disk Full";`
 `fp = close(fp);`
 `end;`
 `endif;`
 `c = c+k;`
 `endo;`
 `fp = close(fp);`

In this example, a 10000x10 data set of Normal random numbers is written to a data set called `data.dat`. The variable names are X01-X10.

See also **`open, close, create, readr, saved, seekr`**

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

xlabel

xlabel

	Purpose	Sets a label for the X axis.
	Library	pgraph
	Format	xlabel (<i>str</i>);
	Input	<i>str</i> string, the label for the X axis.
	Source	pgraph.src
	See also	title, ylabel, zlabel

xpnd

Purpose	Expands a column vector into a symmetric matrix.	
Format	$x = \text{xpnd}(v);$	
Input	v	$K \times 1$ vector, to be expanded into a symmetric matrix.
Output	x	$M \times M$ matrix, the results of taking v and filling in a symmetric matrix with its elements. $M = ((-1 + \text{sqrt}(1+8*K))/2)$
Remarks	If v does not contain the right number of elements, (that is, if sqrt(1 + 8*K) is not integral), then an error message is generated. This function is particularly useful for hard-coding symmetric matrices, because only about half of the matrix needs to be entered.	
Example	$x = \{ \begin{matrix} 1, \\ 2, \ 3, \\ 4, \ 5, \ 6, \\ 7, \ 8, \ 9, \ 10 \end{matrix} \};$ $y = \text{xpnd}(x);$ $x = \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \end{matrix}$	

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

xpnd

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x y z

$$y = \begin{matrix} & 1 & 2 & 4 & 7 \\ 2 & 3 & 5 & & 8 \\ 4 & 5 & 6 & & 9 \\ 7 & 8 & 9 & 10 & \end{matrix}$$

See also **vech**

xtics

Purpose	Sets and fixes scaling, axes numbering and tick marks for the X axis.	
Library	pgraph	
Format	xtics (<i>min,max,step,minordiv</i>) ;	
Input	<i>min</i>	scalar, the minimum value.
	<i>max</i>	scalar, the maximum value.
	<i>step</i>	scalar, the value between major tick marks.
	<i>minordiv</i>	scalar, the number of minor subdivisions.
Remarks	<p>This routine fixes the scaling for all subsequent graphs until graphset is called.</p> <p>This gives you direct control over the axes endpoints and tick marks. If xtics is called after a call to scale, it will override scale.</p> <p>X and Y axes numbering may be reversed for xy, logx, logy, and loglog graphs. This may be accomplished by using a negative step value in the xtics and ytics functions.</p>	
Source	pscale.src	
See also	scale , ytics , ztics	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

xy

xy

a	Purpose	Graphs X vs. Y using Cartesian coordinates.	
b	Library	pggraph	
c	Format	xy(x,y);	
d	Input	x	Nx1 or NxM matrix. Each column contains the X values for a particular line.
e		y	Nx1 or NxM matrix. Each column contains the Y values for a particular line.
f	Remarks	Missing values are ignored when plotting symbols. If missing values are encountered while plotting a curve, the curve will end and a new curve will begin plotting at the next non-missing value.	
g	Source	pxy.src	
h	See also	xyz, logx, logy, loglog	
i			
j			
k			
l			
m			
n			
o			
p			
q			
r			
s			
t			
u			
v			
w			
x y z			

xyz

Purpose	Graphs X vs. Y vs. Z using Cartesian coordinates.	
Library	pgraph	
Format	xyz (<i>x,y,z</i>) ;	
Input	<i>x</i>	Nx1 or NxK matrix. Each column contains the X values for a particular line.
	<i>y</i>	Nx1 or NxK matrix. Each column contains the Y values for a particular line.
	<i>z</i>	Nx1 or NxK matrix. Each column contains the Z values for a particular line.
Remarks	Missing values are ignored when plotting symbols. If missing values are encountered while plotting a curve, the curve will end and a new curve will begin plotting at the next non-missing value.	
Source	pxyz.src	
See also	xy, surface, volume, view	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

ylabel

ylabel

a	Purpose	Sets a label for the Y axis.
b	Library	pgraph
c	Format	ylabel(<i>str</i>) ;
d	Input	<i>str</i> string, the label for the Y axis.
e	Source	pgraph.src
f	See also	title, xlabel, ylabel
g		
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

yticks

Purpose	Sets and fixes scaling, axes numbering and tick marks for the Y axis.	
Library	pgraph	
Format	yticks (<i>min,max,step,minordiv</i>) ;	
Input	<i>min</i>	scalar, the minimum value.
	<i>max</i>	scalar, the maximum value.
	<i>step</i>	scalar, the value between major tick marks.
	<i>minordiv</i>	scalar, the number of minor subdivisions.
Remarks	<p>This routine fixes the scaling for all subsequent graphs until graphset is called.</p> <p>This gives you direct control over the axes endpoints and tick marks. If yticks is called after a call to scale, it will override scale.</p> <p>X and Y axes numbering may be reversed for xy, logx, logy and loglog graphs. This may be accomplished by using a negative step value in the xticks and yticks functions.</p>	
Source	pscale.src	
See also	scale , xticks , zticks	

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

zeros

zeros

a	Purpose	Creates a matrix of zeros.
b	Format	$y = \text{zeros}(r,c);$
c	Input	r scalar, the number of rows. c scalar, the number of columns.
d	Output	y $R \times C$ matrix of zeros.
e	Remarks	This is faster than ones . Noninteger arguments will be truncated to an integer.
f	Example	$y = \text{zeros}(3,2);$ 0.000000 0.000000 $y =$ 0.000000 0.000000 0.000000 0.000000
g	See also	ones, eye
h		
i		
j		
k		
l		
m		
n		
o		
p		
q		
r		
s		
t		
u		
v		
w		
x y z		

zlabel

Purpose	Sets a label for the Z axis.
Library	pgraph
Format	zlabel (<i>str</i>);
Input	<i>str</i> string, the label for the Z axis.
Source	pgraph.src
See also	title , xlabel , ylabel

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x y z

ztics

ztics

a	Purpose	Sets and fixes scaling, axes numbering and tick marks for the Z axis.	
b	Library	pgraph	
c	Format	ztics (<i>min,max,step,minordiv</i>) ;	
d	Input	<i>min</i>	scalar, the minimum value.
e		<i>max</i>	scalar, the maximum value.
f		<i>step</i>	scalar, the value between major tick marks.
g		<i>minordiv</i>	scalar, the number of minor subdivisions. If this function is used with contour , contour labels will be placed every <i>minordiv</i> levels. If 0, there will be no labels.
h	Remarks	This routine fixes the scaling for all subsequent graphs until graphset is called.	
i		This gives you direct control over the axes endpoints and tick marks. If ztics is called after a call to scale3d , it will override scale3d .	
j	Source	pscale.src	
k	See also	scale3d, xtics, ytics, contour	
l			
m			
n			
o			
p			
q			
r			
s			
t			
u			
v			
w			
x y z			

Obsolete Commands **A**

The following commands will no longer be supported, therefore should not be used when creating new programs.

disable	setvmode
editm	WinClear
enable	WinClearArea
files	WinClearTTYlog
font	WinClose
FontLoad	WinCloseAll
FontUnload	WinGetActive
FontUnloadAll	WinGetAttributes
graph	WinGetColorCells
line	WinGetCursor
lprint on/off	WinMove
medit	WinOpenPQG
ndpchk	WinOpenText
ndpclex	WinOpenTTY
ndpctrl	WinPan
plot	WinPrint
plotsym	WinPrintPQG
print on/off	WinRefresh
rndmod	WinRefreshArea
rndns/rndus	WinResize

Obsolete Commands

WinSetActive
WinSetBackground
WinSetColorCells
WinSetColormap
WinSetCursor
WinSetForeground
WinSetRefresh
WinSetTextWrap
WinZoomPQG

Colors Appendix **B**

0 Black	8 Dark Grey
1 Blue	9 Light Blue
2 Green	10 Light Green
3 Cyan	11 Light Cyan
4 Red	12 Light Red
5 Magenta	13 Light Magenta
6 Brown	14 Yellow
7 Grey	15 White

