

STATA FUNCTIONS REFERENCE MANUAL RELEASE 16



A Stata Press Publication
StataCorp LLC
College Station, Texas



® Copyright © 1985–2019 StataCorp LLC
All rights reserved
Version 16

Published by Stata Press, 4905 Lakeway Drive, College Station, Texas 77845
Typeset in T_EX

ISBN-10: 1-59718-278-8

ISBN-13: 978-1-59718-278-2

This manual is protected by copyright. All rights are reserved. No part of this manual may be reproduced, stored in a retrieval system, or transcribed, in any form or by any means—electronic, mechanical, photocopy, recording, or otherwise—without the prior written permission of StataCorp LLC unless permitted subject to the terms and conditions of a license granted to you by StataCorp LLC to use the software and documentation. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted by this document.

StataCorp provides this manual “as is” without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. StataCorp may make improvements and/or changes in the product(s) and the program(s) described in this manual at any time and without notice.

The software described in this manual is furnished under a license agreement or nondisclosure agreement. The software may be copied only in accordance with the terms of the agreement. It is against the law to copy the software onto DVD, CD, disk, diskette, tape, or any other medium for any purpose other than backup or archival purposes.

The automobile dataset appearing on the accompanying media is Copyright © 1979 by Consumers Union of U.S., Inc., Yonkers, NY 10703-1057 and is reproduced by permission from CONSUMER REPORTS, April 1979.

Stata, **STATA** Stata Press, Mata, **MATA** and NetCourse are registered trademarks of StataCorp LLC.

Stata and Stata Press are registered trademarks with the World Intellectual Property Organization of the United Nations.

NetCourseNow is a trademark of StataCorp LLC.

Other brand and product names are registered trademarks or trademarks of their respective companies.

For copyright information about the software, type `help copyright` within Stata.

The suggested citation for this software is

StataCorp. 2019. *Stata: Release 16*. Statistical Software. College Station, TX: StataCorp LLC.

Contents

Intro	Introduction to functions reference manual	1
Functions by category		2
Functions by name		19
Date and time functions		35
Mathematical functions		48
Matrix functions		55
Programming functions		62
Random-number functions		76
Selecting time-span functions		87
Statistical functions		89
String functions		131
Trigonometric functions		161
Subject and author index		164

Cross-referencing the documentation

When reading this manual, you will find references to other Stata manuals, for example, [U] **27 Overview of Stata estimation commands**; [R] **regress**; and [D] **reshape**. The first example is a reference to chapter 27, *Overview of Stata estimation commands*, in the *User's Guide*; the second is a reference to the **regress** entry in the *Base Reference Manual*; and the third is a reference to the **reshape** entry in the *Data Management Reference Manual*.

All the manuals in the Stata Documentation have a shorthand notation:

[GSM]	<i>Getting Started with Stata for Mac</i>
[GSU]	<i>Getting Started with Stata for Unix</i>
[GSW]	<i>Getting Started with Stata for Windows</i>
[U]	<i>Stata User's Guide</i>
[R]	<i>Stata Base Reference Manual</i>
[BAYES]	<i>Stata Bayesian Analysis Reference Manual</i>
[CM]	<i>Stata Choice Models Reference Manual</i>
[D]	<i>Stata Data Management Reference Manual</i>
[DSGE]	<i>Stata Dynamic Stochastic General Equilibrium Models Reference Manual</i>
[ERM]	<i>Stata Extended Regression Models Reference Manual</i>
[FMM]	<i>Stata Finite Mixture Models Reference Manual</i>
[FN]	<i>Stata Functions Reference Manual</i>
[G]	<i>Stata Graphics Reference Manual</i>
[IRT]	<i>Stata Item Response Theory Reference Manual</i>
[LASSO]	<i>Stata Lasso Reference Manual</i>
[XT]	<i>Stata Longitudinal-Data/Panel-Data Reference Manual</i>
[META]	<i>Stata Meta-Analysis Reference Manual</i>
[ME]	<i>Stata Multilevel Mixed-Effects Reference Manual</i>
[MI]	<i>Stata Multiple-Imputation Reference Manual</i>
[MV]	<i>Stata Multivariate Statistics Reference Manual</i>
[PSS]	<i>Stata Power, Precision, and Sample-Size Reference Manual</i>
[P]	<i>Stata Programming Reference Manual</i>
[RPT]	<i>Stata Reporting Reference Manual</i>
[SP]	<i>Stata Spatial Autoregressive Models Reference Manual</i>
[SEM]	<i>Stata Structural Equation Modeling Reference Manual</i>
[SVY]	<i>Stata Survey Data Reference Manual</i>
[ST]	<i>Stata Survival Analysis Reference Manual</i>
[TS]	<i>Stata Time-Series Reference Manual</i>
[TE]	<i>Stata Treatment-Effects Reference Manual: Potential Outcomes/Counterfactual Outcomes</i>
[I]	<i>Stata Glossary and Index</i>
[M]	<i>Mata Reference Manual</i>

Title

Intro — Introduction to functions reference manual

Description

This manual describes the functions allowed by Stata. For information on Mata functions, see [\[M-4\] Intro](#).

A quick note about missing values: Stata denotes a numeric missing value by `.`, `.a`, `.b`, ..., or `.z`. A string missing value is denoted by `"` (the empty string). Here any one of these may be referred to by *missing*. If a numeric value x is missing, then $x \geq .$ is true. If a numeric value x is not missing, then $x < .$ is true.

See [\[U\] 12.2.1 Missing values](#) for details.

Reference

Cox, N. J. 2011. [Speaking Stata: Fun and fluency with functions](#). *Stata Journal* 11: 460–471.

Also see

[\[U\] 1.3 What's new](#)

Contents

Date and time functions
Mathematical functions
Matrix functions
Programming functions
Random-number functions
Selecting time-span functions
Statistical functions
String functions
Trigonometric functions

Date and time functions

<code>bofd("cal", e_d)</code>	the e_b business date corresponding to e_d
<code>Cdhms(e_d, h, m, s)</code>	the e_{tC} datetime (ms. with leap seconds since 01jan1960 00:00:00.000) corresponding to e_d, h, m, s
<code>Chms(h, m, s)</code>	the e_{tC} datetime (ms. with leap seconds since 01jan1960 00:00:00.000) corresponding to h, m, s on 01jan1960
<code>Clock(s_1, s_2 [, Y])</code>	the e_{tC} datetime (ms. with leap seconds since 01jan1960 00:00:00.000) corresponding to s_1 based on s_2 and Y
<code>clock(s_1, s_2 [, Y])</code>	the e_{tc} datetime (ms. since 01jan1960 00:00:00.000) corresponding to s_1 based on s_2 and Y
<code>Cmdyhms(M, D, Y, h, m, s)</code>	the e_{tC} datetime (ms. with leap seconds since 01jan1960 00:00:00.000) corresponding to M, D, Y, h, m, s
<code>Cofc(e_{tc})</code>	the e_{tC} datetime (ms. with leap seconds since 01jan1960 00:00:00.000) of e_{tc} (ms. without leap seconds since 01jan1960 00:00:00.000)
<code>cofC(e_{tC})</code>	the e_{tc} datetime (ms. without leap seconds since 01jan1960 00:00:00.000) of e_{tC} (ms. with leap seconds since 01jan1960 00:00:00.000)
<code>Cofd(e_d)</code>	the e_{tC} datetime (ms. with leap seconds since 01jan1960 00:00:00.000) of date e_d at time 00:00:00.000
<code>cofd(e_d)</code>	the e_{tc} datetime (ms. since 01jan1960 00:00:00.000) of date e_d at time 00:00:00.000
<code>daily(s_1, s_2 [, Y])</code>	a synonym for <code>date(s_1, s_2 [, Y])</code>
<code>date(s_1, s_2 [, Y])</code>	the e_d date (days since 01jan1960) corresponding to s_1 based on s_2 and Y
<code>day(e_d)</code>	the numeric day of the month corresponding to e_d
<code>dhms(e_d, h, m, s)</code>	the e_{tc} datetime (ms. since 01jan1960 00:00:00.000) corresponding to e_d, h, m, s
<code>dofb($e_b, "cal"$)</code>	the e_d datetime corresponding to e_b
<code>dofC(e_{tC})</code>	the e_d date (days since 01jan1960) of datetime e_{tC} (ms. with leap seconds since 01jan1960 00:00:00.000)

<code>dofc(e_{tc})</code>	the e_d date (days since 01jan1960) of datetime e_{tc} (ms. since 01jan1960 00:00:00.000)
<code>dofh(e_h)</code>	the e_d date (days since 01jan1960) of the start of half-year e_h
<code>dofm(e_m)</code>	the e_d date (days since 01jan1960) of the start of month e_m
<code>dofq(e_q)</code>	the e_d date (days since 01jan1960) of the start of quarter e_q
<code>dofw(e_w)</code>	the e_d date (days since 01jan1960) of the start of week e_w
<code>dofy(e_y)</code>	the e_d date (days since 01jan1960) of 01jan in year e_y
<code>dow(e_d)</code>	the numeric day of the week corresponding to date e_d ; 0 = Sunday, 1 = Monday, ..., 6 = Saturday
<code>doy(e_d)</code>	the numeric day of the year corresponding to date e_d
<code>halfyear(e_d)</code>	the numeric half of the year corresponding to date e_d
<code>halfyearly(s_1, s_2 [, Y])</code>	the e_h half-yearly date (half-years since 1960h1) corresponding to s_1 based on s_2 and Y ; Y specifies <i>topyear</i> ; see date()
<code>hh(e_{tc})</code>	the hour corresponding to datetime e_{tc} (ms. since 01jan1960 00:00:00.000)
<code>hhC(e_{tC})</code>	the hour corresponding to datetime e_{tC} (ms. with leap seconds since 01jan1960 00:00:00.000)
<code>hms(h, m, s)</code>	the e_{tc} datetime (ms. since 01jan1960 00:00:00.000) corresponding to h, m, s on 01jan1960
<code>hofd(e_d)</code>	the e_h half-yearly date (half years since 1960h1) containing date e_d
<code>hours(ms)</code>	$ms/3,600,000$
<code>mdy(M, D, Y)</code>	the e_d date (days since 01jan1960) corresponding to M, D, Y
<code>mdyhms(M, D, Y, h, m, s)</code>	the e_{tc} datetime (ms. since 01jan1960 00:00:00.000) corresponding to M, D, Y, h, m, s
<code>minutes(ms)</code>	$ms/60,000$
<code>mm(e_{tc})</code>	the minute corresponding to datetime e_{tc} (ms. since 01jan1960 00:00:00.000)
<code>mmC(e_{tC})</code>	the minute corresponding to datetime e_{tC} (ms. with leap seconds since 01jan1960 00:00:00.000)
<code>mofd(e_d)</code>	the e_m monthly date (months since 1960m1) containing date e_d
<code>month(e_d)</code>	the numeric month corresponding to date e_d
<code>monthly(s_1, s_2 [, Y])</code>	the e_m monthly date (months since 1960m1) corresponding to s_1 based on s_2 and Y ; Y specifies <i>topyear</i> ; see date()
<code>msofhours(h)</code>	$h \times 3,600,000$
<code>msofminutes(m)</code>	$m \times 60,000$
<code>msofseconds(s)</code>	$s \times 1,000$
<code>qofd(e_d)</code>	the e_q quarterly date (quarters since 1960q1) containing date e_d
<code>quarter(e_d)</code>	the numeric quarter of the year corresponding to date e_d
<code>quarterly(s_1, s_2 [, Y])</code>	the e_q quarterly date (quarters since 1960q1) corresponding to s_1 based on s_2 and Y ; Y specifies <i>topyear</i> ; see date()
<code>seconds(ms)</code>	$ms/1,000$
<code>ss(e_{tc})</code>	the second corresponding to datetime e_{tc} (ms. since 01jan1960 00:00:00.000)
<code>ssC(e_{tC})</code>	the second corresponding to datetime e_{tC} (ms. with leap seconds since 01jan1960 00:00:00.000)

<code>tC(l)</code>	convenience function to make typing dates and times in expressions easier
<code>tC(l)</code>	convenience function to make typing dates and times in expressions easier
<code>td(l)</code>	convenience function to make typing dates in expressions easier
<code>th(l)</code>	convenience function to make typing half-yearly dates in expressions easier
<code>tm(l)</code>	convenience function to make typing monthly dates in expressions easier
<code>tq(l)</code>	convenience function to make typing quarterly dates in expressions easier
<code>tw(l)</code>	convenience function to make typing weekly dates in expressions easier
<code>week(e_d)</code>	the numeric week of the year corresponding to date e_d , the %td encoded date (days since 01jan1960)
<code>weekly(s₁, s₂[, Y])</code>	the e_w weekly date (weeks since 1960w1) corresponding to s_1 based on s_2 and Y ; Y specifies <i>topyear</i> ; see date()
<code>wofd(e_d)</code>	the e_w weekly date (weeks since 1960w1) containing date e_d
<code>year(e_d)</code>	the numeric year corresponding to date e_d
<code>yearly(s₁, s₂[, Y])</code>	the e_y yearly date (year) corresponding to s_1 based on s_2 and Y ; Y specifies <i>topyear</i> ; see date()
<code>yh(Y, H)</code>	the e_h half-yearly date (half-years since 1960h1) corresponding to year Y , half-year H
<code>ym(Y, M)</code>	the e_m monthly date (months since 1960m1) corresponding to year Y , month M
<code>yofd(e_d)</code>	the e_y yearly date (year) containing date e_d
<code>yq(Y, Q)</code>	the e_q quarterly date (quarters since 1960q1) corresponding to year Y , quarter Q
<code>yw(Y, W)</code>	the e_w weekly date (weeks since 1960w1) corresponding to year Y , week W

Mathematical functions

<code>abs(x)</code>	the absolute value of x
<code>ceil(x)</code>	the unique integer n such that $n - 1 < x \leq n$; x (not “.”) if x is missing, meaning that <code>ceil(.a) = .a</code>
<code>cloglog(x)</code>	the complementary log-log of x
<code>comb(n, k)</code>	the combinatorial function $n!/\{k!(n-k)!\}$
<code>digamma(x)</code>	the <code>digamma()</code> function, $d \ln \Gamma(x)/dx$
<code>exp(x)</code>	the exponential function e^x
<code>expm1(x)</code>	$e^x - 1$ with higher precision than <code>exp(x) - 1</code> for small values of $ x $
<code>floor(x)</code>	the unique integer n such that $n \leq x < n + 1$; x (not “.”) if x is missing, meaning that <code>floor(.a) = .a</code>
<code>int(x)</code>	the integer obtained by truncating x toward 0 (thus, <code>int(5.2) = 5</code> and <code>int(-5.8) = -5</code>); x (not “.”) if x is missing, meaning that <code>int(.a) = .a</code>

<code>invcloglog(<i>x</i>)</code>	the inverse of the complementary log-log function of x
<code>invlogit(<i>x</i>)</code>	the inverse of the logit function of x
<code>ln(<i>x</i>)</code>	the natural logarithm, $\ln(x)$
<code>ln1m(<i>x</i>)</code>	the natural logarithm of $1 - x$ with higher precision than $\ln(1 - x)$ for small values of $ x $
<code>ln1p(<i>x</i>)</code>	the natural logarithm of $1 + x$ with higher precision than $\ln(1 + x)$ for small values of $ x $
<code>lnfactorial(<i>n</i>)</code>	the natural log of n factorial = $\ln(n!)$
<code>lngamma(<i>x</i>)</code>	$\ln\{\Gamma(x)\}$
<code>log(<i>x</i>)</code>	a synonym for <code>ln(<i>x</i>)</code>
<code>log10(<i>x</i>)</code>	the base-10 logarithm of x
<code>log1m(<i>x</i>)</code>	a synonym for <code>ln1m(<i>x</i>)</code>
<code>log1p(<i>x</i>)</code>	a synonym for <code>ln1p(<i>x</i>)</code>
<code>logit(<i>x</i>)</code>	the log of the odds ratio of x , $\text{logit}(x) = \ln\{x/(1 - x)\}$
<code>max(<i>x</i>₁, <i>x</i>₂, ..., <i>x</i>_{<i>n</i>})</code>	the maximum value of x_1, x_2, \dots, x_n
<code>min(<i>x</i>₁, <i>x</i>₂, ..., <i>x</i>_{<i>n</i>})</code>	the minimum value of x_1, x_2, \dots, x_n
<code>mod(<i>x</i>, <i>y</i>)</code>	the modulus of x with respect to y
<code>reldif(<i>x</i>, <i>y</i>)</code>	the “relative” difference $ x - y /(y + 1)$; 0 if both arguments are the same type of extended missing value; <i>missing</i> if only one argument is missing or if the two arguments are two different types of <i>missing</i>
<code>round(<i>x</i>, <i>y</i>)</code> or <code>round(<i>x</i>)</code>	x rounded in units of y or x rounded to the nearest integer if the argument y is omitted; x (not “.”) if x is missing (meaning that <code>round(.a) = .a</code> and that <code>round(.a, y) = .a</code> if y is not missing) and if y is missing, then “. ” is returned
<code>sign(<i>x</i>)</code>	the sign of x : -1 if $x < 0$, 0 if $x = 0$, 1 if $x > 0$, or <i>missing</i> if x is missing
<code>sqrt(<i>x</i>)</code>	the square root of x
<code>sum(<i>x</i>)</code>	the running sum of x , treating missing values as zero
<code>trigamma(<i>x</i>)</code>	the second derivative of <code>lngamma(<i>x</i>)</code> = $d^2 \ln\Gamma(x)/dx^2$
<code>trunc(<i>x</i>)</code>	a synonym for <code>int(<i>x</i>)</code>

Matrix functions

<code>cholesky(<i>M</i>)</code>	the Cholesky decomposition of the matrix: if $R = \text{cholesky}(S)$, then $RR^T = S$
<code>coleqnumb(<i>M</i>, <i>s</i>)</code>	the equation number of M associated with column equation s ; <i>missing</i> if the column equation cannot be found
<code>colnfreeparms(<i>M</i>)</code>	the number of free parameters in columns of M
<code>colnumb(<i>M</i>, <i>s</i>)</code>	the column number of M associated with column name s ; <i>missing</i> if the column cannot be found
<code>colsof(<i>M</i>)</code>	the number of columns of M
<code>corr(<i>M</i>)</code>	the correlation matrix of the variance matrix
<code>det(<i>M</i>)</code>	the determinant of matrix M

<code>diag(M)</code>	the square, diagonal matrix created from the row or column vector
<code>diag0cnt(M)</code>	the number of zeros on the diagonal of M
<code>el(s, i, j)</code>	$s[\text{floor}(i), \text{floor}(j)]$, the i, j element of the matrix named s ; <i>missing</i> if i or j are out of range or if matrix s does not exist
<code>get(systemname)</code>	a copy of Stata internal system matrix <i>systemname</i>
<code>hadamard(M, N)</code>	a matrix whose i, j element is $M[i, j] \cdot N[i, j]$ (if M and N are not the same size, this function reports a conformability error)
<code>I(n)</code>	an $n \times n$ identity matrix if n is an integer; otherwise, a <code>round(n)</code> \times <code>round(n)</code> identity matrix
<code>inv(M)</code>	the inverse of the matrix M
<code>invsym(M)</code>	the inverse of M if M is positive definite
<code>issymmetric(M)</code>	1 if the matrix is symmetric; otherwise, 0
<code>J(r, c, z)</code>	the $r \times c$ matrix containing elements z
<code>matmissing(M)</code>	1 if any elements of the matrix are missing; otherwise, 0
<code>matuniform(r, c)</code>	the $r \times c$ matrices containing uniformly distributed pseudorandom numbers on the interval (0, 1)
<code>mreldif(X, Y)</code>	the relative difference of X and Y , where the relative difference is defined as $\max_{i,j} \{ x_{ij} - y_{ij} / (y_{ij} + 1)\}$
<code>nullmat(matname)</code>	use with the row-join (,) and column-join (\) operators
<code>roweqnumb(M, s)</code>	the equation number of M associated with row equation s ; <i>missing</i> if the row equation cannot be found
<code>rownfreeparms(M)</code>	the number of free parameters in rows of M
<code>rownumb(M, s)</code>	the row number of M associated with row name s ; <i>missing</i> if the row cannot be found
<code>rowsof(M)</code>	the number of rows of M
<code>sweep(M, i)</code>	matrix M with i th row/column swept
<code>trace(M)</code>	the trace of matrix M
<code>vec(M)</code>	a column vector formed by listing the elements of M , starting with the first column and proceeding column by column
<code>vecdiag(M)</code>	the row vector containing the diagonal of matrix M

Programming functions

<code>autocode(x, n, x_0, x_1)</code>	partitions the interval from x_0 to x_1 into n equal-length intervals and returns the upper bound of the interval that contains x
<code>byteorder()</code>	1 if your computer stores numbers by using a hilo byte order and evaluates to 2 if your computer stores numbers by using a lohi byte order
<code>c(name)</code>	the value of the system or constant result $c(name)$ (see [P] creturn)
<code>_caller()</code>	version of the program or session that invoked the currently running program; see [P] version
<code>chop(x, ϵ)</code>	<code>round(x)</code> if $\text{abs}(x - \text{round}(x)) < \epsilon$; otherwise, x ; or x if x is missing
<code>clip(x, a, b)</code>	x if $a < x < b$, b if $x \geq b$, a if $x \leq a$, or <i>missing</i> if x is missing or if $a > b$; x if x is missing

<code>cond(x,a,b[,c])</code>	<i>a</i> if <i>x</i> is <i>true</i> and nonmissing, <i>b</i> if <i>x</i> is <i>false</i> , and <i>c</i> if <i>x</i> is <i>missing</i> ; <i>a</i> if <i>c</i> is not specified and <i>x</i> evaluates to <i>missing</i>
<code>e(name)</code>	the value of stored result <code>e(name)</code> ; see [U] 18.8 Accessing results calculated by other programs
<code>e(sample)</code>	1 if the observation is in the estimation sample and 0 otherwise
<code>epsdouble()</code>	the machine precision of a double-precision number
<code>epsfloat()</code>	the machine precision of a floating-point number
<code>fileexists(f)</code>	1 if the file specified by <i>f</i> exists; otherwise, 0
<code>fileread(f)</code>	the contents of the file specified by <i>f</i>
<code>filereaderror(s)</code>	0 or positive integer, said value having the interpretation of a return code
<code>filewrite(f,s[,r])</code>	writes the string specified by <i>s</i> to the file specified by <i>f</i> and returns the number of bytes in the resulting file
<code>float(x)</code>	the value of <i>x</i> rounded to <code>float</code> precision
<code>fmtwidth(fmtstr)</code>	the output length of the <code>%fmt</code> contained in <i>fmtstr</i> ; <i>missing</i> if <i>fmtstr</i> does not contain a valid <code>%fmt</code>
<code>frval()</code>	returns values of variables stored in other frames
<code>_frval()</code>	programmer's version of <code>frval()</code>
<code>has_eprop(name)</code>	1 if <i>name</i> appears as a word in <code>e(properties)</code> ; otherwise, 0
<code>inlist(z,a,b,...)</code>	1 if <i>z</i> is a member of the remaining arguments; otherwise, 0
<code>inrange(z,a,b)</code>	1 if it is known that $a \leq z \leq b$; otherwise, 0
<code>irecode(x,x₁,...,x_n)</code>	<i>missing</i> if <i>x</i> is <i>missing</i> or x_1, \dots, x_n is not weakly increasing; 0 if $x \leq x_1$; 1 if $x_1 < x \leq x_2$; 2 if $x_2 < x \leq x_3$; ...; <i>n</i> if $x > x_n$
<code>matrix(exp)</code>	restricts name interpretation to scalars and matrices; see <code>scalar()</code>
<code>maxbyte()</code>	the largest value that can be stored in storage type <code>byte</code>
<code>maxdouble()</code>	the largest value that can be stored in storage type <code>double</code>
<code>maxfloat()</code>	the largest value that can be stored in storage type <code>float</code>
<code>maxint()</code>	the largest value that can be stored in storage type <code>int</code>
<code>maxlong()</code>	the largest value that can be stored in storage type <code>long</code>
<code>mi(x₁,x₂,...,x_n)</code>	a synonym for <code>missing(x₁,x₂,...,x_n)</code>
<code>minbyte()</code>	the smallest value that can be stored in storage type <code>byte</code>
<code>mindouble()</code>	the smallest value that can be stored in storage type <code>double</code>
<code>minfloat()</code>	the smallest value that can be stored in storage type <code>float</code>
<code>minint()</code>	the smallest value that can be stored in storage type <code>int</code>
<code>minlong()</code>	the smallest value that can be stored in storage type <code>long</code>
<code>missing(x₁,x₂,...,x_n)</code>	1 if any <i>x_i</i> evaluates to <i>missing</i> ; otherwise, 0
<code>r(name)</code>	the value of the stored result <code>r(name)</code> ; see [U] 18.8 Accessing results calculated by other programs
<code>recode(x,x₁,...,x_n)</code>	<i>missing</i> if x_1, x_2, \dots, x_n is not weakly increasing; <i>x</i> if <i>x</i> is <i>missing</i> ; <i>x₁</i> if $x \leq x_1$; <i>x₂</i> if $x \leq x_2, \dots$; otherwise, <i>x_n</i> if $x > x_1, x_2, \dots, x_{n-1}$. <i>x_i ≥ .</i> is interpreted as <i>x_i = +∞</i>
<code>replay()</code>	1 if the first nonblank character of local macro '0' is a comma, or if '0' is empty

<code>return(name)</code>	the value of the to-be-stored result <code>r(name)</code> ; see [P] return
<code>s(name)</code>	the value of stored result <code>s(name)</code> ; see [U] 18.8 Accessing results calculated by other programs
<code>scalar(exp)</code>	restricts name interpretation to scalars and matrices
<code>smallestdouble()</code>	the smallest double-precision number greater than zero

Random-number functions

<code>rbeta(a,b)</code>	<code>beta(a,b)</code> random variates, where a and b are the beta distribution shape parameters
<code>rbinomial(n,p)</code>	<code>binomial(n,p)</code> random variates, where n is the number of trials and p is the success probability
<code>rcauchy(a,b)</code>	<code>Cauchy(a,b)</code> random variates, where a is the location parameter and b is the scale parameter
<code>rchi2(df)</code>	chi-squared, with df degrees of freedom, random variates
<code>rexponential(b)</code>	exponential random variates with scale b
<code>rgamma(a,b)</code>	<code>gamma(a,b)</code> random variates, where a is the gamma shape parameter and b is the scale parameter
<code>rhypergeometric(N,K,n)</code>	hypergeometric random variates
<code>rigaussian(m,a)</code>	inverse Gaussian random variates with mean m and shape parameter a
<code>rlaplace(m,b)</code>	<code>Laplace(m,b)</code> random variates with mean m and scale parameter b
<code>rlogistic()</code>	logistic variates with mean 0 and standard deviation $\pi/\sqrt{3}$
<code>rlogistic(s)</code>	logistic variates with mean 0, scale s , and standard deviation $s\pi/\sqrt{3}$
<code>rlogistic(m,s)</code>	logistic variates with mean m , scale s , and standard deviation $s\pi/\sqrt{3}$
<code>rnbinomial(n,p)</code>	negative binomial random variates
<code>rnormal()</code>	standard normal (Gaussian) random variates, that is, variates from a normal distribution with a mean of 0 and a standard deviation of 1
<code>rnormal(m)</code>	<code>normal(m,1)</code> (Gaussian) random variates, where m is the mean and the standard deviation is 1
<code>rnormal(m,s)</code>	<code>normal(m,s)</code> (Gaussian) random variates, where m is the mean and s is the standard deviation
<code>rpoisson(m)</code>	<code>Poisson(m)</code> random variates, where m is the distribution mean
<code>rt(df)</code>	Student's t random variates, where df is the degrees of freedom
<code>runiform()</code>	uniformly distributed random variates over the interval (0,1)
<code>runiform(a,b)</code>	uniformly distributed random variates over the interval (a,b)
<code>runiformint(a,b)</code>	uniformly distributed random integer variates on the interval $[a,b]$
<code>rweibull(a,b)</code>	Weibull variates with shape a and scale b
<code>rweibull(a,b,g)</code>	Weibull variates with shape a , scale b , and location g
<code>rweibullph(a,b)</code>	Weibull (proportional hazards) variates with shape a and scale b
<code>rweibullph(a,b,g)</code>	Weibull (proportional hazards) variates with shape a , scale b , and location g

Selecting time-span functions

<code>tin(d_1, d_2)</code>	<i>true</i> if $d_1 \leq t \leq d_2$, where t is the time variable previously <code>tsset</code>
<code>twithin(d_1, d_2)</code>	<i>true</i> if $d_1 < t < d_2$, where t is the time variable previously <code>tsset</code>

Statistical functions

<code>betaden(a, b, x)</code>	the probability density of the beta distribution, where a and b are the shape parameters; 0 if $x < 0$ or $x > 1$
<code>binomial(n, k, θ)</code>	the probability of observing <code>floor(k)</code> or fewer successes in <code>floor(n)</code> trials when the probability of a success on one trial is θ ; 0 if $k < 0$; or 1 if $k > n$
<code>binomialp(n, k, p)</code>	the probability of observing <code>floor(k)</code> successes in <code>floor(n)</code> trials when the probability of a success on one trial is p
<code>binomialtail(n, k, θ)</code>	the probability of observing <code>floor(k)</code> or more successes in <code>floor(n)</code> trials when the probability of a success on one trial is θ ; 1 if $k < 0$; or 0 if $k > n$
<code>binormal(h, k, ρ)</code>	the joint cumulative distribution $\Phi(h, k, \rho)$ of bivariate normal with correlation ρ
<code>cauchy(a, b, x)</code>	the cumulative Cauchy distribution with location parameter a and scale parameter b
<code>cauchyden(a, b, x)</code>	the probability density of the Cauchy distribution with location parameter a and scale parameter b
<code>cauchytail(a, b, x)</code>	the reverse cumulative (upper tail or survivor) Cauchy distribution with location parameter a and scale parameter b
<code>chi2(df, x)</code>	the cumulative χ^2 distribution with df degrees of freedom; 0 if $x < 0$
<code>chi2den(df, x)</code>	the probability density of the chi-squared distribution with df degrees of freedom; 0 if $x < 0$
<code>chi2tail(df, x)</code>	the reverse cumulative (upper tail or survivor) χ^2 distribution with df degrees of freedom; 1 if $x < 0$
<code>dgammapda(a, x)</code>	$\frac{\partial P(a, x)}{\partial a}$, where $P(a, x) = \text{gammap}(a, x)$; 0 if $x < 0$
<code>dgammapdada(a, x)</code>	$\frac{\partial^2 P(a, x)}{\partial a^2}$, where $P(a, x) = \text{gammap}(a, x)$; 0 if $x < 0$
<code>dgammapdadx(a, x)</code>	$\frac{\partial^2 P(a, x)}{\partial a \partial x}$, where $P(a, x) = \text{gammap}(a, x)$; 0 if $x < 0$
<code>dgammapdx(a, x)</code>	$\frac{\partial P(a, x)}{\partial x}$, where $P(a, x) = \text{gammap}(a, x)$; 0 if $x < 0$
<code>dgammapdxdx(a, x)</code>	$\frac{\partial^2 P(a, x)}{\partial x^2}$, where $P(a, x) = \text{gammap}(a, x)$; 0 if $x < 0$
<code>dunnettprob(k, df, x)</code>	the cumulative multiple range distribution that is used in Dunnett's multiple-comparison method with k ranges and df degrees of freedom; 0 if $x < 0$
<code>exponential(b, x)</code>	the cumulative exponential distribution with scale b
<code>exponentialden(b, x)</code>	the probability density function of the exponential distribution with scale b
<code>exponentialtail(b, x)</code>	the reverse cumulative exponential distribution with scale b

<code>F(df₁,df₂,f)</code>	the cumulative F distribution with df_1 numerator and df_2 denominator degrees of freedom: $F(df_1,df_2,f) = \int_0^f \text{Fden}(df_1,df_2,t) dt$; 0 if $f < 0$
<code>Fden(df₁,df₂,f)</code>	the probability density function of the F distribution with df_1 numerator and df_2 denominator degrees of freedom; 0 if $f < 0$
<code>Ftail(df₁,df₂,f)</code>	the reverse cumulative (upper tail or survivor) F distribution with df_1 numerator and df_2 denominator degrees of freedom; 1 if $f < 0$
<code>gammaden(a,b,g,x)</code>	the probability density function of the gamma distribution; 0 if $x < g$
<code>gammap(a,x)</code>	the cumulative gamma distribution with shape parameter a ; 0 if $x < 0$
<code>gammaptail(a,x)</code>	the reverse cumulative (upper tail or survivor) gamma distribution with shape parameter a ; 1 if $x < 0$
<code>hypergeometric(N,K,n,k)</code>	the cumulative probability of the hypergeometric distribution
<code>hypergeometricp(N,K,n,k)</code>	the hypergeometric probability of k successes out of a sample of size n , from a population of size N containing K elements that have the attribute of interest
<code>ibeta(a,b,x)</code>	the cumulative beta distribution with shape parameters a and b ; 0 if $x < 0$; or 1 if $x > 1$
<code>ibetatail(a,b,x)</code>	the reverse cumulative (upper tail or survivor) beta distribution with shape parameters a and b ; 1 if $x < 0$; or 0 if $x > 1$
<code>igaussian(m,a,x)</code>	the cumulative inverse Gaussian distribution with mean m and shape parameter a ; 0 if $x \leq 0$
<code>igaussianden(m,a,x)</code>	the probability density of the inverse Gaussian distribution with mean m and shape parameter a ; 0 if $x \leq 0$
<code>igaussiantail(m,a,x)</code>	the reverse cumulative (upper tail or survivor) inverse Gaussian distribution with mean m and shape parameter a ; 1 if $x \leq 0$
<code>invbinomial(n,k,p)</code>	the inverse of the cumulative binomial; that is, θ (θ = probability of success on one trial) such that the probability of observing <code>floor(k)</code> or fewer successes in <code>floor(n)</code> trials is p
<code>invbinomialtail(n,k,p)</code>	the inverse of the right cumulative binomial; that is, θ (θ = probability of success on one trial) such that the probability of observing <code>floor(k)</code> or more successes in <code>floor(n)</code> trials is p
<code>invcauchy(a,b,p)</code>	the inverse of <code>cauchy()</code> : if <code>cauchy(a,b,x) = p</code> , then <code>invcauchy(a,b,p) = x</code>
<code>invcauchytail(a,b,p)</code>	the inverse of <code>cauchytail()</code> : if <code>cauchytail(a,b,x) = p</code> , then <code>invcauchytail(a,b,p) = x</code>
<code>invchi2(df,p)</code>	the inverse of <code>chi2()</code> : if <code>chi2(df,x) = p</code> , then <code>invchi2(df,p) = x</code>
<code>invchi2tail(df,p)</code>	the inverse of <code>chi2tail()</code> : if <code>chi2tail(df,x) = p</code> , then <code>invchi2tail(df,p) = x</code>
<code>invdunnettprob(k,df,p)</code>	the inverse cumulative multiple range distribution that is used in Dunnett's multiple-comparison method with k ranges and df degrees of freedom
<code>invexponential(b,p)</code>	the inverse cumulative exponential distribution with scale b : if <code>exponential(b,x) = p</code> , then <code>invexponential(b,p) = x</code>

<code>invexponentialtail(b,p)</code>	the inverse reverse cumulative exponential distribution with scale b : if <code>exponentialtail(b,x) = p</code> , then <code>invexponentialtail(b,p) = x</code>
<code>invF(df₁,df₂,p)</code>	the inverse cumulative F distribution: if <code>F(df₁,df₂,f) = p</code> , then <code>invF(df₁,df₂,p) = f</code>
<code>invFtail(df₁,df₂,p)</code>	the inverse reverse cumulative (upper tail or survivor) F distribution: if <code>Ftail(df₁,df₂,f) = p</code> , then <code>invFtail(df₁,df₂,p) = f</code>
<code>invgammap(a,p)</code>	the inverse cumulative gamma distribution: if <code>gammap(a,x) = p</code> , then <code>invgammap(a,p) = x</code>
<code>invgammaptail(a,p)</code>	the inverse reverse cumulative (upper tail or survivor) gamma distribution: if <code>gammaptail(a,x) = p</code> , then <code>invgammaptail(a,p) = x</code>
<code>invibeta(a,b,p)</code>	the inverse cumulative beta distribution: if <code>ibeta(a,b,x) = p</code> , then <code>invibeta(a,b,p) = x</code>
<code>invibetatail(a,b,p)</code>	the inverse reverse cumulative (upper tail or survivor) beta distribution: if <code>ibetatail(a,b,x) = p</code> , then <code>invibetatail(a,b,p) = x</code>
<code>invgaussian(m,a,p)</code>	the inverse of <code>igaussian()</code> : if <code>igaussian(m,a,x) = p</code> , then <code>invgaussian(m,a,p) = x</code>
<code>invgaussiantail(m,a,p)</code>	the inverse of <code>igaussiantail()</code> : if <code>igaussiantail(m,a,x) = p</code> , then <code>invgaussiantail(m,a,p) = x</code>
<code>invlaplace(m,b,p)</code>	the inverse of <code>laplace()</code> : if <code>laplace(m,b,x) = p</code> , then <code>invlaplace(m,b,p) = x</code>
<code>invlaplacetail(m,b,p)</code>	the inverse of <code>laplacetail()</code> : if <code>laplacetail(m,b,x) = p</code> , then <code>invlaplacetail(m,b,p) = x</code>
<code>invlogistic(p)</code>	the inverse cumulative logistic distribution: if <code>logistic(x) = p</code> , then <code>invlogistic(p) = x</code>
<code>invlogistic(s,p)</code>	the inverse cumulative logistic distribution: if <code>logistic(s,x) = p</code> , then <code>invlogistic(s,p) = x</code>
<code>invlogistic(m,s,p)</code>	the inverse cumulative logistic distribution: if <code>logistic(m,s,x) = p</code> , then <code>invlogistic(m,s,p) = x</code>
<code>invlogistictail(p)</code>	the inverse reverse cumulative logistic distribution: if <code>logistictail(x) = p</code> , then <code>invlogistictail(p) = x</code>
<code>invlogistictail(s,p)</code>	the inverse reverse cumulative logistic distribution: if <code>logistictail(s,x) = p</code> , then <code>invlogistictail(s,p) = x</code>
<code>invlogistictail(m,s,p)</code>	the inverse reverse cumulative logistic distribution: if <code>logistictail(m,s,x) = p</code> , then <code>invlogistictail(m,s,p) = x</code>
<code>invnbinomial(n,k,q)</code>	the value of the negative binomial parameter, p , such that $q =$ <code>nbinomial(n,k,p)</code>
<code>invnbinomialtail(n,k,q)</code>	the value of the negative binomial parameter, p , such that $q =$ <code>nbinomialtail(n,k,p)</code>
<code>invnchi2(df,np,p)</code>	the inverse cumulative noncentral χ^2 distribution: if <code>nchi2(df,np,x) = p</code> , then <code>invnchi2(df,np,p) = x</code>
<code>invnchi2tail(df,np,p)</code>	the inverse reverse cumulative (upper tail or survivor) non- central χ^2 distribution: if <code>nchi2tail(df,np,x) = p</code> , then <code>invnchi2tail(df,np,p) = x</code>

<code>invnF(df₁,df₂,np,p)</code>	the inverse cumulative noncentral F distribution: if $\text{nF}(df_1,df_2,np,f) = p$, then $\text{invnF}(df_1,df_2,np,p) = f$
<code>invnFtail(df₁,df₂,np,p)</code>	the inverse reverse cumulative (upper tail or survivor) noncentral F distribution: if $\text{nFtail}(df_1,df_2,np,f) = p$, then $\text{invnFtail}(df_1,df_2,np,p) = f$
<code>invnibeta(a,b,np,p)</code>	the inverse cumulative noncentral beta distribution: if $\text{nibeta}(a,b,np,x) = p$, then $\text{invnibeta}(a,b,np,p) = x$
<code>invnormal(p)</code>	the inverse cumulative standard normal distribution: if $\text{normal}(z) = p$, then $\text{invnormal}(p) = z$
<code>invnt(df,np,p)</code>	the inverse cumulative noncentral Student's t distribution: if $\text{nt}(df,np,t) = p$, then $\text{invnt}(df,np,p) = t$
<code>invnttail(df,np,p)</code>	the inverse reverse cumulative (upper tail or survivor) noncentral Student's t distribution: if $\text{nttail}(df,np,t) = p$, then $\text{invnttail}(df,np,p) = t$
<code>invpoisson(k,p)</code>	the Poisson mean such that the cumulative Poisson distribution evaluated at k is p : if $\text{poisson}(m,k) = p$, then $\text{invpoisson}(k,p) = m$
<code>invpoissontail(k,q)</code>	the Poisson mean such that the reverse cumulative Poisson distribution evaluated at k is q : if $\text{poissontail}(m,k) = q$, then $\text{invpoissontail}(k,q) = m$
<code>invt(df,p)</code>	the inverse cumulative Student's t distribution: if $\text{t}(df,t) = p$, then $\text{invt}(df,p) = t$
<code>invttail(df,p)</code>	the inverse reverse cumulative (upper tail or survivor) Student's t distribution: if $\text{ttail}(df,t) = p$, then $\text{invttail}(df,p) = t$
<code>invtukeyprob(k,df,p)</code>	the inverse cumulative Tukey's Studentized range distribution with k ranges and df degrees of freedom
<code>invweibull(a,b,p)</code>	the inverse cumulative Weibull distribution with shape a and scale b : if $\text{weibull}(a,b,x) = p$, then $\text{invweibull}(a,b,p) = x$
<code>invweibull(a,b,g,p)</code>	the inverse cumulative Weibull distribution with shape a , scale b , and location g : if $\text{weibull}(a,b,g,x) = p$, then $\text{invweibull}(a,b,g,p) = x$
<code>invweibullph(a,b,p)</code>	the inverse cumulative Weibull (proportional hazards) distribution with shape a and scale b : if $\text{weibullph}(a,b,x) = p$, then $\text{invweibullph}(a,b,p) = x$
<code>invweibullph(a,b,g,p)</code>	the inverse cumulative Weibull (proportional hazards) distribution with shape a , scale b , and location g : if $\text{weibullph}(a,b,g,x) = p$, then $\text{invweibullph}(a,b,g,p) = x$
<code>invweibullphtail(a,b,p)</code>	the inverse reverse cumulative Weibull (proportional hazards) distribution with shape a and scale b : if $\text{weibullphtail}(a,b,x) = p$, then $\text{invweibullphtail}(a,b,p) = x$
<code>invweibullphtail(a,b,g,p)</code>	the inverse reverse cumulative Weibull (proportional hazards) distribution with shape a , scale b , and location g : if $\text{weibullphtail}(a,b,g,x) = p$, then $\text{invweibullphtail}(a,b,g,p) = x$
<code>invweibulltail(a,b,p)</code>	the inverse reverse cumulative Weibull distribution with shape a and scale b : if $\text{weibulltail}(a,b,x) = p$, then $\text{invweibulltail}(a,b,p) = x$
<code>invweibulltail(a,b,g,p)</code>	the inverse reverse cumulative Weibull distribution with shape a , scale b , and location g : if $\text{weibulltail}(a,b,g,x) = p$, then $\text{invweibulltail}(a,b,g,p) = x$

<code>laplace(m, b, x)</code>	the cumulative Laplace distribution with mean m and scale parameter b
<code>laplaceden(m, b, x)</code>	the probability density of the Laplace distribution with mean m and scale parameter b
<code>laplacetail(m, b, x)</code>	the reverse cumulative (upper tail or survivor) Laplace distribution with mean m and scale parameter b
<code>lncauchyden(a, b, x)</code>	the natural logarithm of the density of the Cauchy distribution with location parameter a and scale parameter b
<code>lnigammaden(a, b, x)</code>	the natural logarithm of the inverse gamma density, where a is the shape parameter and b is the scale parameter
<code>lnigaussianden(m, a, x)</code>	the natural logarithm of the inverse Gaussian density with mean m and shape parameter a
<code>lniwishartden(df, V, X)</code>	the natural logarithm of the density of the inverse Wishart distribution; missing if $df \leq n - 1$
<code>lnlaplaceden(m, b, x)</code>	the natural logarithm of the density of the Laplace distribution with mean m and scale parameter b
<code>lnmvnormalden(M, V, X)</code>	the natural logarithm of the multivariate normal density
<code>lnnormal(z)</code>	the natural logarithm of the cumulative standard normal distribution
<code>lnnormalden(z)</code>	the natural logarithm of the standard normal density, $N(0, 1)$
<code>lnnormalden(x, σ)</code>	the natural logarithm of the normal density with mean 0 and standard deviation σ
<code>lnnormalden(x, μ, σ)</code>	the natural logarithm of the normal density with mean μ and standard deviation σ , $N(\mu, \sigma^2)$
<code>lnwishartden(df, V, X)</code>	the natural logarithm of the density of the Wishart distribution; missing if $df \leq n - 1$
<code>logistic(x)</code>	the cumulative logistic distribution with mean 0 and standard deviation $\pi/\sqrt{3}$
<code>logistic(s, x)</code>	the cumulative logistic distribution with mean 0, scale s , and standard deviation $s\pi/\sqrt{3}$
<code>logistic(m, s, x)</code>	the cumulative logistic distribution with mean m , scale s , and standard deviation $s\pi/\sqrt{3}$
<code>logisticden(x)</code>	the density of the logistic distribution with mean 0 and standard deviation $\pi/\sqrt{3}$
<code>logisticden(s, x)</code>	the density of the logistic distribution with mean 0, scale s , and standard deviation $s\pi/\sqrt{3}$
<code>logisticden(m, s, x)</code>	the density of the logistic distribution with mean m , scale s , and standard deviation $s\pi/\sqrt{3}$
<code>logistictail(x)</code>	the reverse cumulative logistic distribution with mean 0 and standard deviation $\pi/\sqrt{3}$
<code>logistictail(s, x)</code>	the reverse cumulative logistic distribution with mean 0, scale s , and standard deviation $s\pi/\sqrt{3}$
<code>logistictail(m, s, x)</code>	the reverse cumulative logistic distribution with mean m , scale s , and standard deviation $s\pi/\sqrt{3}$
<code>nbetaden(a, b, np, x)</code>	the probability density function of the noncentral beta distribution; 0 if $x < 0$ or $x > 1$
<code>nbinomial(n, k, p)</code>	the cumulative probability of the negative binomial distribution

<code>nbinomialp(<i>n</i>,<i>k</i>,<i>p</i>)</code>	the negative binomial probability
<code>nbinomialtail(<i>n</i>,<i>k</i>,<i>p</i>)</code>	the reverse cumulative probability of the negative binomial distribution
<code>nchi2(<i>df</i>,<i>np</i>,<i>x</i>)</code>	the cumulative noncentral χ^2 distribution; 0 if $x < 0$
<code>nchi2den(<i>df</i>,<i>np</i>,<i>x</i>)</code>	the probability density of the noncentral χ^2 distribution; 0 if $x < 0$
<code>nchi2tail(<i>df</i>,<i>np</i>,<i>x</i>)</code>	the reverse cumulative (upper tail or survivor) noncentral χ^2 distribution; 1 if $x < 0$
<code>nF(<i>df</i>₁,<i>df</i>₂,<i>np</i>,<i>f</i>)</code>	the cumulative noncentral F distribution with <i>df</i> ₁ numerator and <i>df</i> ₂ denominator degrees of freedom and noncentrality parameter <i>np</i> ; 0 if $f < 0$
<code>nFden(<i>df</i>₁,<i>df</i>₂,<i>np</i>,<i>f</i>)</code>	the probability density function of the noncentral F distribution with <i>df</i> ₁ numerator and <i>df</i> ₂ denominator degrees of freedom and noncentrality parameter <i>np</i> ; 0 if $f < 0$
<code>nFtail(<i>df</i>₁,<i>df</i>₂,<i>np</i>,<i>f</i>)</code>	the reverse cumulative (upper tail or survivor) noncentral F distribution with <i>df</i> ₁ numerator and <i>df</i> ₂ denominator degrees of freedom and noncentrality parameter <i>np</i> ; 1 if $f < 0$
<code>nibeta(<i>a</i>,<i>b</i>,<i>np</i>,<i>x</i>)</code>	the cumulative noncentral beta distribution; 0 if $x < 0$; or 1 if $x > 1$
<code>normal(<i>z</i>)</code>	the cumulative standard normal distribution
<code>normalden(<i>z</i>)</code>	the standard normal density, $N(0, 1)$
<code>normalden(<i>x</i>,σ)</code>	the normal density with mean 0 and standard deviation σ
<code>normalden(<i>x</i>,μ,σ)</code>	the normal density with mean μ and standard deviation σ , $N(\mu, \sigma^2)$
<code>npnchi2(<i>df</i>,<i>x</i>,<i>p</i>)</code>	the noncentrality parameter, <i>np</i> , for noncentral χ^2 : if <code>nchi2(<i>df</i>,<i>np</i>,<i>x</i>) = <i>p</i></code> , then <code>npnchi2(<i>df</i>,<i>x</i>,<i>p</i>) = <i>np</i></code>
<code>npnF(<i>df</i>₁,<i>df</i>₂,<i>f</i>,<i>p</i>)</code>	the noncentrality parameter, <i>np</i> , for the noncentral F : if <code>nF(<i>df</i>₁,<i>df</i>₂,<i>np</i>,<i>f</i>) = <i>p</i></code> , then <code>npnF(<i>df</i>₁,<i>df</i>₂,<i>f</i>,<i>p</i>) = <i>np</i></code>
<code>npnt(<i>df</i>,<i>t</i>,<i>p</i>)</code>	the noncentrality parameter, <i>np</i> , for the noncentral Student's t distribution: if <code>nt(<i>df</i>,<i>np</i>,<i>t</i>) = <i>p</i></code> , then <code>npnt(<i>df</i>,<i>t</i>,<i>p</i>) = <i>np</i></code>
<code>nt(<i>df</i>,<i>np</i>,<i>t</i>)</code>	the cumulative noncentral Student's t distribution with <i>df</i> degrees of freedom and noncentrality parameter <i>np</i>
<code>ntden(<i>df</i>,<i>np</i>,<i>t</i>)</code>	the probability density function of the noncentral Student's t distribution with <i>df</i> degrees of freedom and noncentrality parameter <i>np</i>
<code>nttail(<i>df</i>,<i>np</i>,<i>t</i>)</code>	the reverse cumulative (upper tail or survivor) noncentral Student's t distribution with <i>df</i> degrees of freedom and noncentrality parameter <i>np</i>
<code>poisson(<i>m</i>,<i>k</i>)</code>	the probability of observing <code>floor(<i>k</i>)</code> or fewer outcomes that are distributed as Poisson with mean <i>m</i>
<code>poissonp(<i>m</i>,<i>k</i>)</code>	the probability of observing <code>floor(<i>k</i>)</code> outcomes that are distributed as Poisson with mean <i>m</i>
<code>poisontail(<i>m</i>,<i>k</i>)</code>	the probability of observing <code>floor(<i>k</i>)</code> or more outcomes that are distributed as Poisson with mean <i>m</i>
<code>t(<i>df</i>,<i>t</i>)</code>	the cumulative Student's t distribution with <i>df</i> degrees of freedom
<code>tden(<i>df</i>,<i>t</i>)</code>	the probability density function of Student's t distribution
<code>ttail(<i>df</i>,<i>t</i>)</code>	the reverse cumulative (upper tail or survivor) Student's t distribution; the probability $T > t$
<code>tukeyprob(<i>k</i>,<i>df</i>,<i>x</i>)</code>	the cumulative Tukey's Studentized range distribution with <i>k</i> ranges and <i>df</i> degrees of freedom; 0 if $x < 0$

<code>weibull(a,b,x)</code>	the cumulative Weibull distribution with shape a and scale b
<code>weibull(a,b,g,x)</code>	the cumulative Weibull distribution with shape a , scale b , and location g
<code>weibullden(a,b,x)</code>	the probability density function of the Weibull distribution with shape a and scale b
<code>weibullden(a,b,g,x)</code>	the probability density function of the Weibull distribution with shape a , scale b , and location g
<code>weibullph(a,b,x)</code>	the cumulative Weibull (proportional hazards) distribution with shape a and scale b
<code>weibullph(a,b,g,x)</code>	the cumulative Weibull (proportional hazards) distribution with shape a , scale b , and location g
<code>weibullphden(a,b,x)</code>	the probability density function of the Weibull (proportional hazards) distribution with shape a and scale b
<code>weibullphden(a,b,g,x)</code>	the probability density function of the Weibull (proportional hazards) distribution with shape a , scale b , and location g
<code>weibullphtail(a,b,x)</code>	the reverse cumulative Weibull (proportional hazards) distribution with shape a and scale b
<code>weibullphtail(a,b,g,x)</code>	the reverse cumulative Weibull (proportional hazards) distribution with shape a , scale b , and location g
<code>weibulltail(a,b,x)</code>	the reverse cumulative Weibull distribution with shape a and scale b
<code>weibulltail(a,b,g,x)</code>	the reverse cumulative Weibull distribution with shape a , scale b , and location g

String functions

<code>abbrev(s,n)</code>	name s , abbreviated to a length of n
<code>char(n)</code>	the character corresponding to ASCII or extended ASCII code n ; "" if n is not in the domain
<code>collatorlocale(loc,type)</code>	the most closely related locale supported by ICU from loc if $type$ is 1; the actual locale where the collation data comes from if $type$ is 2
<code>collatorversion(loc)</code>	the version string of a collator based on locale loc
<code>indexnot(s₁,s₂)</code>	the position in ASCII string s_1 of the first character of s_1 not found in ASCII string s_2 , or 0 if all characters of s_1 are found in s_2
<code>plural(n,s)</code>	the plural of s if $n \neq \pm 1$
<code>plural(n,s₁,s₂)</code>	the plural of s_1 , as modified by or replaced with s_2 , if $n \neq \pm 1$
<code>real(s)</code>	s converted to numeric or <i>missing</i>
<code>regexm(s,re)</code>	performs a match of a regular expression and evaluates to 1 if regular expression re is satisfied by the ASCII string s ; otherwise, 0
<code>regexpr(s₁,re,s₂)</code>	replaces the first substring within ASCII string s_1 that matches re with ASCII string s_2 and returns the resulting string
<code>regexs(n)</code>	subexpression n from a previous <code>regexm()</code> match, where $0 \leq n < 10$
<code>soundex(s)</code>	the soundex code for a string, s
<code>soundex_nara(s)</code>	the U.S. Census soundex code for a string, s

<code>strcat(<i>s</i>₁,<i>s</i>₂)</code>	there is no <code>strcat()</code> function; instead the addition operator is used to concatenate strings
<code>strdup(<i>s</i>₁,<i>n</i>)</code>	there is no <code>strdup()</code> function; instead the multiplication operator is used to create multiple copies of strings
<code>string(<i>n</i>)</code>	a synonym for <code>strofreal(<i>n</i>)</code>
<code>string(<i>n</i>,<i>s</i>)</code>	a synonym for <code>strofreal(<i>n</i>,<i>s</i>)</code>
<code>stritrim(<i>s</i>)</code>	<i>s</i> with multiple, consecutive internal blanks (ASCII space character <code>char(32)</code>) collapsed to one blank
<code>strlen(<i>s</i>)</code>	the number of characters in ASCII <i>s</i> or length in bytes
<code>strlower(<i>s</i>)</code>	lowercase ASCII characters in string <i>s</i>
<code>strltrim(<i>s</i>)</code>	<i>s</i> without leading blanks (ASCII space character <code>char(32)</code>)
<code>strmatch(<i>s</i>₁,<i>s</i>₂)</code>	1 if <i>s</i> ₁ matches the pattern <i>s</i> ₂ ; otherwise, 0
<code>strofreal(<i>n</i>)</code>	<i>n</i> converted to a string
<code>strofreal(<i>n</i>,<i>s</i>)</code>	<i>n</i> converted to a string using the specified display format
<code>strpos(<i>s</i>₁,<i>s</i>₂)</code>	the position in <i>s</i> ₁ at which <i>s</i> ₂ is first found; otherwise, 0
<code>strproper(<i>s</i>)</code>	a string with the first ASCII letter and any other letters immediately following characters that are not letters capitalized; all other ASCII letters converted to lowercase
<code>strreverse(<i>s</i>)</code>	reverses the ASCII string <i>s</i>
<code>strrpos(<i>s</i>₁,<i>s</i>₂)</code>	the position in <i>s</i> ₁ at which <i>s</i> ₂ is last found; otherwise, 0
<code>strrtrim(<i>s</i>)</code>	<i>s</i> without trailing blanks (ASCII space character <code>char(32)</code>)
<code>strtoname(<i>s</i>[,<i>p</i>])</code>	<i>s</i> translated into a Stata 13 compatible name
<code>strtrim(<i>s</i>)</code>	<i>s</i> without leading and trailing blanks (ASCII space character <code>char(32)</code>); equivalent to <code>strltrim(strrtrim(<i>s</i>))</code>
<code>strupper(<i>s</i>)</code>	uppercase ASCII characters in string <i>s</i>
<code>subinstr(<i>s</i>₁,<i>s</i>₂,<i>s</i>₃,<i>n</i>)</code>	<i>s</i> ₁ , where the first <i>n</i> occurrences in <i>s</i> ₁ of <i>s</i> ₂ have been replaced with <i>s</i> ₃
<code>subinword(<i>s</i>₁,<i>s</i>₂,<i>s</i>₃,<i>n</i>)</code>	<i>s</i> ₁ , where the first <i>n</i> occurrences in <i>s</i> ₁ of <i>s</i> ₂ as a word have been replaced with <i>s</i> ₃
<code>substr(<i>s</i>,<i>n</i>₁,<i>n</i>₂)</code>	the substring of <i>s</i> , starting at <i>n</i> ₁ , for a length of <i>n</i> ₂
<code>tobytes(<i>s</i>[,<i>n</i>])</code>	escaped decimal or hex digit strings of up to 200 bytes of <i>s</i>
<code>uchar(<i>n</i>)</code>	the Unicode character corresponding to Unicode code point <i>n</i> or an empty string if <i>n</i> is beyond the Unicode code-point range
<code>udstrlen(<i>s</i>)</code>	the number of display columns needed to display the Unicode string <i>s</i> in the Stata Results window
<code>udsubstr(<i>s</i>,<i>n</i>₁,<i>n</i>₂)</code>	the Unicode substring of <i>s</i> , starting at character <i>n</i> ₁ , for <i>n</i> ₂ display columns
<code>uisdigit(<i>s</i>)</code>	1 if the first Unicode character in <i>s</i> is a Unicode decimal digit; otherwise, 0
<code>uisletter(<i>s</i>)</code>	1 if the first Unicode character in <i>s</i> is a Unicode letter; otherwise, 0
<code>ustrcompare(<i>s</i>₁,<i>s</i>₂[,<i>loc</i>])</code>	compares two Unicode strings
<code>ustrcompareex(<i>s</i>₁,<i>s</i>₂,<i>loc</i>,<i>st</i>,<i>case</i>,<i>cslv</i>,<i>norm</i>,<i>num</i>,<i>alt</i>,<i>fr</i>)</code>	compares two Unicode strings
<code>ustrfix(<i>s</i>[,<i>rep</i>])</code>	replaces each invalid UTF-8 sequence with a Unicode character

<code>ustrfrom(<i>s,enc,mode</i>)</code>	converts the string <i>s</i> in encoding <i>enc</i> to a UTF-8 encoded Unicode string
<code>ustrinvalidcnt(<i>s</i>)</code>	the number of invalid UTF-8 sequences in <i>s</i>
<code>ustrleft(<i>s,n</i>)</code>	the first <i>n</i> Unicode characters of the Unicode string <i>s</i>
<code>ustrlen(<i>s</i>)</code>	the number of characters in the Unicode string <i>s</i>
<code>ustrlower(<i>s</i>[,<i>loc</i>])</code>	lowercase all characters of Unicode string <i>s</i> under the given locale <i>loc</i>
<code>ustrltrim(<i>s</i>)</code>	removes the leading Unicode whitespace characters and blanks from the Unicode string <i>s</i>
<code>ustrnormalize(<i>s,norm</i>)</code>	normalizes Unicode string <i>s</i> to one of the five normalization forms specified by <i>norm</i>
<code>ustrpos(<i>s</i>₁,<i>s</i>₂[,<i>n</i>])</code>	the position in <i>s</i> ₁ at which <i>s</i> ₂ is first found; otherwise, 0
<code>ustrregexm(<i>s,re</i>[,<i>noc</i>])</code>	performs a match of a regular expression and evaluates to 1 if regular expression <i>re</i> is satisfied by the Unicode string <i>s</i> ; otherwise, 0
<code>ustrregexra(<i>s</i>₁,<i>re</i>,<i>s</i>₂[,<i>noc</i>])</code>	replaces all substrings within the Unicode string <i>s</i> ₁ that match <i>re</i> with <i>s</i> ₂ and returns the resulting string
<code>ustrregexrf(<i>s</i>₁,<i>re</i>,<i>s</i>₂[,<i>noc</i>])</code>	replaces the first substring within the Unicode string <i>s</i> ₁ that matches <i>re</i> with <i>s</i> ₂ and returns the resulting string
<code>ustrregexs(<i>n</i>)</code>	subexpression <i>n</i> from a previous <code>ustrregexm()</code> match
<code>ustrreverse(<i>s</i>)</code>	reverses the Unicode string <i>s</i>
<code>ustrright(<i>s,n</i>)</code>	the last <i>n</i> Unicode characters of the Unicode string <i>s</i>
<code>ustrrpos(<i>s</i>₁,<i>s</i>₂[,<i>n</i>])</code>	the position in <i>s</i> ₁ at which <i>s</i> ₂ is last found; otherwise, 0
<code>ustrrtrim(<i>s</i>)</code>	remove trailing Unicode whitespace characters and blanks from the Unicode string <i>s</i>
<code>ustrsortkey(<i>s</i>[,<i>loc</i>])</code>	generates a null-terminated byte array that can be used by the <code>sort</code> command to produce the same order as <code>ustrcompare()</code>
<code>ustrsortkeyex(<i>s,loc,st,case,cslv,norm,num,alt,fr</i>)</code>	generates a null-terminated byte array that can be used by the <code>sort</code> command to produce the same order as <code>ustrcompare()</code>
<code>ustrtitle(<i>s</i>[,<i>loc</i>])</code>	a string with the first characters of Unicode words titlecased and other characters lowercased
<code>ustrto(<i>s,enc,mode</i>)</code>	converts the Unicode string <i>s</i> in UTF-8 encoding to a string in encoding <i>enc</i>
<code>ustrtohex(<i>s</i>[,<i>n</i>])</code>	escaped hex digit string of <i>s</i> up to 200 Unicode characters
<code>ustrtoname(<i>s</i>[,<i>p</i>])</code>	string <i>s</i> translated into a Stata name
<code>ustrtrim(<i>s</i>)</code>	removes leading and trailing Unicode whitespace characters and blanks from the Unicode string <i>s</i>
<code>ustrunescape(<i>s</i>)</code>	the Unicode string corresponding to the escaped sequences of <i>s</i>
<code>ustrupper(<i>s</i>[,<i>loc</i>])</code>	uppercase all characters in string <i>s</i> under the given locale <i>loc</i>
<code>ustrword(<i>s,n</i>[,<i>loc</i>])</code>	the <i>n</i> th Unicode word in the Unicode string <i>s</i>
<code>ustrwordcount(<i>s</i>[,<i>loc</i>])</code>	the number of nonempty Unicode words in the Unicode string <i>s</i>
<code>usubinstr(<i>s</i>₁,<i>s</i>₂,<i>s</i>₃,<i>n</i>)</code>	replaces the first <i>n</i> occurrences of the Unicode string <i>s</i> ₂ with the Unicode string <i>s</i> ₃ in <i>s</i> ₁
<code>usubstr(<i>s,n</i>₁,<i>n</i>₂)</code>	the Unicode substring of <i>s</i> , starting at <i>n</i> ₁ , for a length of <i>n</i> ₂

<code>word(s, n)</code>	the n th word in s ; <i>missing</i> ("") if n is missing
<code>wordbreaklocale(loc, type)</code>	the most closely related locale supported by ICU from loc if $type$ is 1, the actual locale where the word-boundary analysis data come from if $type$ is 2; or an empty string is returned for any other $type$
<code>wordcount(s)</code>	the number of words in s

Trigonometric functions

<code>acos(x)</code>	the radian value of the arccosine of x
<code>acosh(x)</code>	the inverse hyperbolic cosine of x
<code>asin(x)</code>	the radian value of the arcsine of x
<code>asinh(x)</code>	the inverse hyperbolic sine of x
<code>atan(x)</code>	the radian value of the arctangent of x
<code>atan2(y, x)</code>	the radian value of the arctangent of y/x , where the signs of the parameters y and x are used to determine the quadrant of the answer
<code>atanh(x)</code>	the inverse hyperbolic tangent of x
<code>cos(x)</code>	the cosine of x , where x is in radians
<code>cosh(x)</code>	the hyperbolic cosine of x
<code>sin(x)</code>	the sine of x , where x is in radians
<code>sinh(x)</code>	the hyperbolic sine of x
<code>tan(x)</code>	the tangent of x , where x is in radians
<code>tanh(x)</code>	the hyperbolic tangent of x

Also see

[FN] **Functions by name**

[D] **egen** — Extensions to generate

[D] **generate** — Create or change contents of variable

[M-4] **Intro** — Categorical guide to Mata functions

[U] **13.3 Functions**

Functions by name

<code>abbrev(s, n)</code>	name s , abbreviated to a length of n
<code>abs(x)</code>	the absolute value of x
<code>acos(x)</code>	the radian value of the arccosine of x
<code>acosh(x)</code>	the inverse hyperbolic cosine of x
<code>asin(x)</code>	the radian value of the arcsine of x
<code>asinh(x)</code>	the inverse hyperbolic sine of x
<code>atan(x)</code>	the radian value of the arctangent of x
<code>atan2(y, x)</code>	the radian value of the arctangent of y/x , where the signs of the parameters y and x are used to determine the quadrant of the answer
<code>atanh(x)</code>	the inverse hyperbolic tangent of x
<code>autocode(x, n, x₀, x₁)</code>	partitions the interval from x_0 to x_1 into n equal-length intervals and returns the upper bound of the interval that contains x
<code>betaden(a, b, x)</code>	the probability density of the beta distribution, where a and b are the shape parameters; 0 if $x < 0$ or $x > 1$
<code>binomial(n, k, θ)</code>	the probability of observing <code>floor(k)</code> or fewer successes in <code>floor(n)</code> trials when the probability of a success on one trial is θ ; 0 if $k < 0$; or 1 if $k > n$
<code>binomialp(n, k, p)</code>	the probability of observing <code>floor(k)</code> successes in <code>floor(n)</code> trials when the probability of a success on one trial is p
<code>binomialtail(n, k, θ)</code>	the probability of observing <code>floor(k)</code> or more successes in <code>floor(n)</code> trials when the probability of a success on one trial is θ ; 1 if $k < 0$; or 0 if $k > n$
<code>binormal(h, k, ρ)</code>	the joint cumulative distribution $\Phi(h, k, \rho)$ of bivariate normal with correlation ρ
<code>bofd("cal", e_d)</code>	the e_b business date corresponding to e_d
<code>byteorder()</code>	1 if your computer stores numbers by using a hilo byte order and evaluates to 2 if your computer stores numbers by using a lohi byte order
<code>c(name)</code>	the value of the system or constant result $c(name)$ (see [P] <code>creturn</code>)
<code>_caller()</code>	version of the program or session that invoked the currently running program; see [P] <code>version</code>
<code>cauchy(a, b, x)</code>	the cumulative Cauchy distribution with location parameter a and scale parameter b
<code>cauchyden(a, b, x)</code>	the probability density of the Cauchy distribution with location parameter a and scale parameter b
<code>cauchytail(a, b, x)</code>	the reverse cumulative (upper tail or survivor) Cauchy distribution with location parameter a and scale parameter b
<code>Cdhms(e_d, h, m, s)</code>	the e_{tC} datetime (ms. with leap seconds since 01jan1960 00:00:00.000) corresponding to e_d, h, m, s
<code>ceil(x)</code>	the unique integer n such that $n - 1 < x \leq n$; x (not ".") if x is missing, meaning that <code>ceil(.a) = .a</code>

<code>char(<i>n</i>)</code>	the character corresponding to ASCII or extended ASCII code <i>n</i> ; "" if <i>n</i> is not in the domain
<code>chi2(<i>df</i>, <i>x</i>)</code>	the cumulative χ^2 distribution with <i>df</i> degrees of freedom; 0 if <i>x</i> < 0
<code>chi2den(<i>df</i>, <i>x</i>)</code>	the probability density of the chi-squared distribution with <i>df</i> degrees of freedom; 0 if <i>x</i> < 0
<code>chi2tail(<i>df</i>, <i>x</i>)</code>	the reverse cumulative (upper tail or survivor) χ^2 distribution with <i>df</i> degrees of freedom; 1 if <i>x</i> < 0
<code>Chms(<i>h</i>, <i>m</i>, <i>s</i>)</code>	the e_{tC} datetime (ms. with leap seconds since 01jan1960 00:00:00.000) corresponding to <i>h</i> , <i>m</i> , <i>s</i> on 01jan1960
<code>chop(<i>x</i>, ϵ)</code>	round(<i>x</i>) if abs(<i>x</i> - round(<i>x</i>)) < ϵ ; otherwise, <i>x</i> ; or <i>x</i> if <i>x</i> is missing
<code>cholesky(<i>M</i>)</code>	the Cholesky decomposition of the matrix: if $R = \text{cholesky}(S)$, then $RR^T = S$
<code>clip(<i>x</i>, <i>a</i>, <i>b</i>)</code>	<i>x</i> if $a < x < b$, <i>b</i> if $x \geq b$, <i>a</i> if $x \leq a$, or <i>missing</i> if <i>x</i> is missing or if $a > b$; <i>x</i> if <i>x</i> is missing
<code>Clock(<i>s</i>₁, <i>s</i>₂[, <i>Y</i>])</code>	the e_{tC} datetime (ms. with leap seconds since 01jan1960 00:00:00.000) corresponding to <i>s</i> ₁ based on <i>s</i> ₂ and <i>Y</i>
<code>clock(<i>s</i>₁, <i>s</i>₂[, <i>Y</i>])</code>	the e_{tc} datetime (ms. since 01jan1960 00:00:00.000) corresponding to <i>s</i> ₁ based on <i>s</i> ₂ and <i>Y</i>
<code>cloglog(<i>x</i>)</code>	the complementary log-log of <i>x</i>
<code>Cmdyhms(<i>M</i>, <i>D</i>, <i>Y</i>, <i>h</i>, <i>m</i>, <i>s</i>)</code>	the e_{tC} datetime (ms. with leap seconds since 01jan1960 00:00:00.000) corresponding to <i>M</i> , <i>D</i> , <i>Y</i> , <i>h</i> , <i>m</i> , <i>s</i>
<code>Cofc(e_{tc})</code>	the e_{tC} datetime (ms. with leap seconds since 01jan1960 00:00:00.000) of e_{tc} (ms. without leap seconds since 01jan1960 00:00:00.000)
<code>cofC(e_{tC})</code>	the e_{tc} datetime (ms. without leap seconds since 01jan1960 00:00:00.000) of e_{tC} (ms. with leap seconds since 01jan1960 00:00:00.000)
<code>Cofd(e_d)</code>	the e_{tC} datetime (ms. with leap seconds since 01jan1960 00:00:00.000) of date e_d at time 00:00:00.000
<code>cofd(e_d)</code>	the e_{tc} datetime (ms. since 01jan1960 00:00:00.000) of date e_d at time 00:00:00.000
<code>coleqnumb(<i>M</i>, <i>s</i>)</code>	the equation number of <i>M</i> associated with column equation <i>s</i> ; <i>missing</i> if the column equation cannot be found
<code>collatorlocale(<i>loc</i>, <i>type</i>)</code>	the most closely related locale supported by ICU from <i>loc</i> if <i>type</i> is 1; the actual locale where the collation data comes from if <i>type</i> is 2
<code>collatorversion(<i>loc</i>)</code>	the version string of a collator based on locale <i>loc</i>
<code>colnfreeparms(<i>M</i>)</code>	the number of free parameters in columns of <i>M</i>
<code>colnumb(<i>M</i>, <i>s</i>)</code>	the column number of <i>M</i> associated with column name <i>s</i> ; <i>missing</i> if the column cannot be found
<code>colsof(<i>M</i>)</code>	the number of columns of <i>M</i>
<code>comb(<i>n</i>, <i>k</i>)</code>	the combinatorial function $n!/\{k!(n-k)!\}$
<code>cond(<i>x</i>, <i>a</i>, <i>b</i>[, <i>c</i>])</code>	<i>a</i> if <i>x</i> is <i>true</i> and nonmissing, <i>b</i> if <i>x</i> is <i>false</i> , and <i>c</i> if <i>x</i> is <i>missing</i> ; <i>a</i> if <i>c</i> is not specified and <i>x</i> evaluates to <i>missing</i>
<code>corr(<i>M</i>)</code>	the correlation matrix of the variance matrix

<code>cos(<i>x</i>)</code>	the cosine of <i>x</i> , where <i>x</i> is in radians
<code>cosh(<i>x</i>)</code>	the hyperbolic cosine of <i>x</i>
<code>daily(<i>s</i>₁,<i>s</i>₂[<i>Y</i>])</code>	a synonym for <code>date(<i>s</i>₁,<i>s</i>₂[<i>Y</i>])</code>
<code>date(<i>s</i>₁,<i>s</i>₂[<i>Y</i>])</code>	the <i>e_d</i> date (days since 01jan1960) corresponding to <i>s</i> ₁ based on <i>s</i> ₂ and <i>Y</i>
<code>day(<i>e_d</i>)</code>	the numeric day of the month corresponding to <i>e_d</i>
<code>det(<i>M</i>)</code>	the determinant of matrix <i>M</i>
<code>dgamma_{pda}(<i>a</i>,<i>x</i>)</code>	$\frac{\partial P(a,x)}{\partial a}$, where $P(a,x) = \text{gamma}(a,x)$; 0 if $x < 0$
<code>dgamma_{pdda}(<i>a</i>,<i>x</i>)</code>	$\frac{\partial^2 P(a,x)}{\partial a^2}$, where $P(a,x) = \text{gamma}(a,x)$; 0 if $x < 0$
<code>dgamma_{pddx}(<i>a</i>,<i>x</i>)</code>	$\frac{\partial^2 P(a,x)}{\partial a \partial x}$, where $P(a,x) = \text{gamma}(a,x)$; 0 if $x < 0$
<code>dgamma_{pdx}(<i>a</i>,<i>x</i>)</code>	$\frac{\partial P(a,x)}{\partial x}$, where $P(a,x) = \text{gamma}(a,x)$; 0 if $x < 0$
<code>dgamma_{pdx}_{dx}(<i>a</i>,<i>x</i>)</code>	$\frac{\partial^2 P(a,x)}{\partial x^2}$, where $P(a,x) = \text{gamma}(a,x)$; 0 if $x < 0$
<code>dhms(<i>e_d</i>,<i>h</i>,<i>m</i>,<i>s</i>)</code>	the <i>e_{tc}</i> datetime (ms. since 01jan1960 00:00:00.000) corresponding to <i>e_d</i> , <i>h</i> , <i>m</i> , and <i>s</i>
<code>diag(<i>M</i>)</code>	the square, diagonal matrix created from the row or column vector
<code>diag0cnt(<i>M</i>)</code>	the number of zeros on the diagonal of <i>M</i>
<code>digamma(<i>x</i>)</code>	the <code>digamma()</code> function, $d \ln \Gamma(x)/dx$
<code>dofb(<i>e_b</i>, "cal")</code>	the <i>e_d</i> datetime corresponding to <i>e_b</i>
<code>dofC(<i>e_{tC}</i>)</code>	the <i>e_d</i> date (days since 01jan1960) of datetime <i>e_{tC}</i> (ms. with leap seconds since 01jan1960 00:00:00.000)
<code>dofc(<i>e_{tc}</i>)</code>	the <i>e_d</i> date (days since 01jan1960) of datetime <i>e_{tc}</i> (ms. since 01jan1960 00:00:00.000)
<code>dofh(<i>e_h</i>)</code>	the <i>e_d</i> date (days since 01jan1960) of the start of half-year <i>e_h</i>
<code>dofm(<i>e_m</i>)</code>	the <i>e_d</i> date (days since 01jan1960) of the start of month <i>e_m</i>
<code>dofq(<i>e_q</i>)</code>	the <i>e_d</i> date (days since 01jan1960) of the start of quarter <i>e_q</i>
<code>dofw(<i>e_w</i>)</code>	the <i>e_d</i> date (days since 01jan1960) of the start of week <i>e_w</i>
<code>dofy(<i>e_y</i>)</code>	the <i>e_d</i> date (days since 01jan1960) of 01jan in year <i>e_y</i>
<code>dow(<i>e_d</i>)</code>	the numeric day of the week corresponding to date <i>e_d</i> ; 0 = Sunday, 1 = Monday, ..., 6 = Saturday
<code>doy(<i>e_d</i>)</code>	the numeric day of the year corresponding to date <i>e_d</i>
<code>dunnettprob(<i>k</i>,<i>df</i>,<i>x</i>)</code>	the cumulative multiple range distribution that is used in Dunnett's multiple-comparison method with <i>k</i> ranges and <i>df</i> degrees of freedom; 0 if $x < 0$
<code>e(name)</code>	the value of stored result <code>e(name)</code> ; see [U] 18.8 Accessing results calculated by other programs
<code>el(<i>s</i>,<i>i</i>,<i>j</i>)</code>	<i>s</i> [<code>floor(i)</code> , <code>floor(j)</code>], the <i>i</i> , <i>j</i> element of the matrix named <i>s</i> ; <i>missing</i> if <i>i</i> or <i>j</i> are out of range or if matrix <i>s</i> does not exist
<code>e(sample)</code>	1 if the observation is in the estimation sample and 0 otherwise
<code>epsdouble()</code>	the machine precision of a double-precision number
<code>epsfloat()</code>	the machine precision of a floating-point number
<code>exp(<i>x</i>)</code>	the exponential function e^x
<code>expm1(<i>x</i>)</code>	$e^x - 1$ with higher precision than <code>exp(<i>x</i>) - 1</code> for small values of $ x $

<code>exponential(<i>b</i>,<i>x</i>)</code>	the cumulative exponential distribution with scale <i>b</i>
<code>exponentialden(<i>b</i>,<i>x</i>)</code>	the probability density function of the exponential distribution with scale <i>b</i>
<code>exponentialtail(<i>b</i>,<i>x</i>)</code>	the reverse cumulative exponential distribution with scale <i>b</i>
<code>F(<i>df</i>₁,<i>df</i>₂,<i>f</i>)</code>	the cumulative <i>F</i> distribution with <i>df</i> ₁ numerator and <i>df</i> ₂ denominator degrees of freedom: $F(df_1, df_2, f) = \int_0^f Fden(df_1, df_2, t) dt$; 0 if <i>f</i> < 0
<code>Fden(<i>df</i>₁,<i>df</i>₂,<i>f</i>)</code>	the probability density function of the <i>F</i> distribution with <i>df</i> ₁ numerator and <i>df</i> ₂ denominator degrees of freedom; 0 if <i>f</i> < 0
<code>fileexists(<i>f</i>)</code>	1 if the file specified by <i>f</i> exists; otherwise, 0
<code>fileread(<i>f</i>)</code>	the contents of the file specified by <i>f</i>
<code>filereaderror(<i>s</i>)</code>	0 or positive integer, said value having the interpretation of a return code
<code>filewrite(<i>f</i>,<i>s</i>[,<i>r</i>])</code>	writes the string specified by <i>s</i> to the file specified by <i>f</i> and returns the number of bytes in the resulting file
<code>float(<i>x</i>)</code>	the value of <i>x</i> rounded to <code>float</code> precision
<code>floor(<i>x</i>)</code>	the unique integer <i>n</i> such that $n \leq x < n + 1$; <i>x</i> (not “.”) if <i>x</i> is missing, meaning that <code>floor(.a) = .a</code>
<code>fmtwidth(<i>fmtstr</i>)</code>	the output length of the <code>%fmt</code> contained in <i>fmtstr</i> ; <i>missing</i> if <i>fmtstr</i> does not contain a valid <code>%fmt</code>
<code>frval()</code>	returns values of variables stored in other frames
<code>_frval()</code>	programmer’s version of <code>frval()</code>
<code>Ftail(<i>df</i>₁,<i>df</i>₂,<i>f</i>)</code>	the reverse cumulative (upper tail or survivor) <i>F</i> distribution with <i>df</i> ₁ numerator and <i>df</i> ₂ denominator degrees of freedom; 1 if <i>f</i> < 0
<code>gammaden(<i>a</i>,<i>b</i>,<i>g</i>,<i>x</i>)</code>	the probability density function of the gamma distribution; 0 if <i>x</i> < <i>g</i>
<code>gammap(<i>a</i>,<i>x</i>)</code>	the cumulative gamma distribution with shape parameter <i>a</i> ; 0 if <i>x</i> < 0
<code>gammaptail(<i>a</i>,<i>x</i>)</code>	the reverse cumulative (upper tail or survivor) gamma distribution with shape parameter <i>a</i> ; 1 if <i>x</i> < 0
<code>get(<i>systemname</i>)</code>	a copy of Stata internal system matrix <i>systemname</i>
<code>hadamard(<i>M</i>,<i>N</i>)</code>	a matrix whose <i>i</i> , <i>j</i> element is $M[i, j] \cdot N[i, j]$ (if <i>M</i> and <i>N</i> are not the same size, this function reports a conformability error)
<code>halfyear(<i>e</i>_{<i>d</i>})</code>	the numeric half of the year corresponding to date <i>e</i> _{<i>d</i>}
<code>halfyearly(<i>s</i>₁,<i>s</i>₂[,<i>Y</i>])</code>	the <i>e</i> _{<i>h</i>} half-yearly date (half-years since 1960h1) corresponding to <i>s</i> ₁ based on <i>s</i> ₂ and <i>Y</i> ; <i>Y</i> specifies <i>topyear</i> ; see date()
<code>has_eprop(<i>name</i>)</code>	1 if <i>name</i> appears as a word in <code>e(properties)</code> ; otherwise, 0
<code>hh(<i>e</i>_{<i>tc</i>})</code>	the hour corresponding to datetime <i>e</i> _{<i>tc</i>} (ms. since 01jan1960 00:00:00.000)
<code>hhC(<i>e</i>_{<i>tC</i>})</code>	the hour corresponding to datetime <i>e</i> _{<i>tC</i>} (ms. with leap seconds since 01jan1960 00:00:00.000)
<code>hms(<i>h</i>,<i>m</i>,<i>s</i>)</code>	the <i>e</i> _{<i>tc</i>} datetime (ms. since 01jan1960 00:00:00.000) corresponding to <i>h</i> , <i>m</i> , <i>s</i> on 01jan1960
<code>hofd(<i>e</i>_{<i>d</i>})</code>	the <i>e</i> _{<i>h</i>} half-yearly date (half years since 1960h1) containing date <i>e</i> _{<i>d</i>}
<code>hours(<i>ms</i>)</code>	<i>ms</i> /3,600,000

<code>hypergeometric(N, K, n, k)</code>	the cumulative probability of the hypergeometric distribution
<code>hypergeometricp(N, K, n, k)</code>	the hypergeometric probability of k successes out of a sample of size n , from a population of size N containing K elements that have the attribute of interest
<code>I(n)</code>	an $n \times n$ identity matrix if n is an integer; otherwise, a <code>round(n)</code> \times <code>round(n)</code> identity matrix
<code>ibeta(a, b, x)</code>	the cumulative beta distribution with shape parameters a and b ; 0 if $x < 0$; or 1 if $x > 1$
<code>ibetatail(a, b, x)</code>	the reverse cumulative (upper tail or survivor) beta distribution with shape parameters a and b ; 1 if $x < 0$; or 0 if $x > 1$
<code>igaussian(m, a, x)</code>	the cumulative inverse Gaussian distribution with mean m and shape parameter a ; 0 if $x \leq 0$
<code>igaussianden(m, a, x)</code>	the probability density of the inverse Gaussian distribution with mean m and shape parameter a ; 0 if $x \leq 0$
<code>igaussiantail(m, a, x)</code>	the reverse cumulative (upper tail or survivor) inverse Gaussian distribution with mean m and shape parameter a ; 1 if $x \leq 0$
<code>indexnot(s_1, s_2)</code>	the position in ASCII string s_1 of the first character of s_1 not found in ASCII string s_2 , or 0 if all characters of s_1 are found in s_2
<code>inlist(z, a, b, \dots)</code>	1 if z is a member of the remaining arguments; otherwise, 0
<code>inrange(z, a, b)</code>	1 if it is known that $a \leq z \leq b$; otherwise, 0
<code>int(x)</code>	the integer obtained by truncating x toward 0 (thus, <code>int(5.2)</code> = 5 and <code>int(-5.8)</code> = -5); x (not ".") if x is missing, meaning that <code>int(.a)</code> = .a
<code>inv(M)</code>	the inverse of the matrix M
<code>invbinomial(n, k, p)</code>	the inverse of the cumulative binomial; that is, θ (θ = probability of success on one trial) such that the probability of observing <code>floor(k)</code> or fewer successes in <code>floor(n)</code> trials is p
<code>invbinomialtail(n, k, p)</code>	the inverse of the right cumulative binomial; that is, θ (θ = probability of success on one trial) such that the probability of observing <code>floor(k)</code> or more successes in <code>floor(n)</code> trials is p
<code>invcauchy(a, b, p)</code>	the inverse of <code>cauchy()</code> : if <code>cauchy(a, b, x)</code> = p , then <code>invcauchy(a, b, p)</code> = x
<code>invcauchytail(a, b, p)</code>	the inverse of <code>cauchytail()</code> : if <code>cauchytail(a, b, x)</code> = p , then <code>invcauchytail(a, b, p)</code> = x
<code>invchi2(df, p)</code>	the inverse of <code>chi2()</code> : if <code>chi2(df, x)</code> = p , then <code>invchi2(df, p)</code> = x
<code>invchi2tail(df, p)</code>	the inverse of <code>chi2tail()</code> : if <code>chi2tail(df, x)</code> = p , then <code>invchi2tail(df, p)</code> = x
<code>invcloglog(x)</code>	the inverse of the complementary log-log function of x
<code>invdunnettprob(k, df, p)</code>	the inverse cumulative multiple range distribution that is used in Dunnett's multiple-comparison method with k ranges and df degrees of freedom
<code>invexponential(b, p)</code>	the inverse cumulative exponential distribution with scale b : if <code>exponential(b, x)</code> = p , then <code>invexponential(b, p)</code> = x
<code>invexponentialtail(b, p)</code>	the inverse reverse cumulative exponential distribution with scale b : if <code>exponentialtail(b, x)</code> = p , then <code>invexponentialtail(b, p)</code> = x

<code>invF(df₁,df₂,p)</code>	the inverse cumulative F distribution: if $F(df_1,df_2,f) = p$, then $invF(df_1,df_2,p) = f$
<code>invFtail(df₁,df₂,p)</code>	the inverse reverse cumulative (upper tail or survivor) F distribution: if $Ftail(df_1,df_2,f) = p$, then $invFtail(df_1,df_2,p) = f$
<code>invgammap(a,p)</code>	the inverse cumulative gamma distribution: if $gammap(a,x) = p$, then $invgammap(a,p) = x$
<code>invgammaptail(a,p)</code>	the inverse reverse cumulative (upper tail or survivor) gamma distribution: if $gammaptail(a,x) = p$, then $invgammaptail(a,p) = x$
<code>invibeta(a,b,p)</code>	the inverse cumulative beta distribution: if $ibeta(a,b,x) = p$, then $invibeta(a,b,p) = x$
<code>invibetatail(a,b,p)</code>	the inverse reverse cumulative (upper tail or survivor) beta distribution: if $ibetatail(a,b,x) = p$, then $invibetatail(a,b,p) = x$
<code>invgaussian(m,a,p)</code>	the inverse of <code>igaussian()</code> : if $igaussian(m,a,x) = p$, then $invgaussian(m,a,p) = x$
<code>invgaussiantail(m,a,p)</code>	the inverse of <code>igaussiantail()</code> : if $igaussiantail(m,a,x) = p$, then $invgaussiantail(m,a,p) = x$
<code>invlaplace(m,b,p)</code>	the inverse of <code>laplace()</code> : if $laplace(m,b,x) = p$, then $invlaplace(m,b,p) = x$
<code>invlaplacetail(m,b,p)</code>	the inverse of <code>laplacetail()</code> : if $laplacetail(m,b,x) = p$, then $invlaplacetail(m,b,p) = x$
<code>invlogistic(p)</code>	the inverse cumulative logistic distribution: if $logistic(x) = p$, then $invlogistic(p) = x$
<code>invlogistic(s,p)</code>	the inverse cumulative logistic distribution: if $logistic(s,x) = p$, then $invlogistic(s,p) = x$
<code>invlogistic(m,s,p)</code>	the inverse cumulative logistic distribution: if $logistic(m,s,x) = p$, then $invlogistic(m,s,p) = x$
<code>invlogistictail(p)</code>	the inverse reverse cumulative logistic distribution: if $logistictail(x) = p$, then $invlogistictail(p) = x$
<code>invlogistictail(s,p)</code>	the inverse cumulative logistic distribution: if $logistic(s,x) = p$, then $invlogistic(s,p) = x$
<code>invlogistictail(m,s,p)</code>	the inverse cumulative logistic distribution: if $logistic(m,s,x) = p$, then $invlogistic(m,s,p) = x$
<code>invlogit(x)</code>	the inverse of the logit function of x
<code>invnbinomial(n,k,q)</code>	the value of the negative binomial parameter, p , such that $q = nbinomial(n,k,p)$
<code>invnbinomialtail(n,k,q)</code>	the value of the negative binomial parameter, p , such that $q = nbinomialtail(n,k,p)$
<code>invnchi2(df,np,p)</code>	the inverse cumulative noncentral χ^2 distribution: if $nchi2(df,np,x) = p$, then $invnchi2(df,np,p) = x$
<code>invnchi2tail(df,np,p)</code>	the inverse reverse cumulative (upper tail or survivor) noncentral χ^2 distribution: if $nchi2tail(df,np,x) = p$, then $invnchi2tail(df,np,p) = x$
<code>invnF(df₁,df₂,np,p)</code>	the inverse cumulative noncentral F distribution: if $nF(df_1,df_2,np,f) = p$, then $invnF(df_1,df_2,np,p) = f$

<code>invnFtail(df₁,df₂,np,p)</code>	the inverse reverse cumulative (upper tail or survivor) noncentral F distribution: if <code>nFtail(df₁,df₂,np,f) = p</code> , then <code>invnFtail(df₁,df₂,np,p) = f</code>
<code>invnibeta(a,b,np,p)</code>	the inverse cumulative noncentral beta distribution: if <code>nibeta(a,b,np,x) = p</code> , then <code>invnibeta(a,b,np,p) = x</code>
<code>invnormal(p)</code>	the inverse cumulative standard normal distribution: if <code>normal(z) = p</code> , then <code>invnormal(p) = z</code>
<code>invnt(df,np,p)</code>	the inverse cumulative noncentral Student's t distribution: if <code>nt(df,np,t) = p</code> , then <code>invnt(df,np,p) = t</code>
<code>invnttail(df,np,p)</code>	the inverse reverse cumulative (upper tail or survivor) noncentral Student's t distribution: if <code>nttail(df,np,t) = p</code> , then <code>invnttail(df,np,p) = t</code>
<code>invpoisson(k,p)</code>	the Poisson mean such that the cumulative Poisson distribution evaluated at k is p : if <code>poisson(m,k) = p</code> , then <code>invpoisson(k,p) = m</code>
<code>invpoisontail(k,q)</code>	the Poisson mean such that the reverse cumulative Poisson distribution evaluated at k is q : if <code>poisontail(m,k) = q</code> , then <code>invpoisontail(k,q) = m</code>
<code>invsym(M)</code>	the inverse of M if M is positive definite
<code>invt(df,p)</code>	the inverse cumulative Student's t distribution: if <code>t(df,t) = p</code> , then <code>invt(df,p) = t</code>
<code>invttail(df,p)</code>	the inverse reverse cumulative (upper tail or survivor) Student's t distribution: if <code>ttail(df,t) = p</code> , then <code>invttail(df,p) = t</code>
<code>invtukeyprob(k,df,p)</code>	the inverse cumulative Tukey's Studentized range distribution with k ranges and df degrees of freedom
<code>invweibull(a,b,p)</code>	the inverse cumulative Weibull distribution with shape a and scale b : if <code>weibull(a,b,x) = p</code> , then <code>invweibull(a,b,p) = x</code>
<code>invweibull(a,b,g,p)</code>	the inverse cumulative Weibull distribution with shape a , scale b , and location g : if <code>weibull(a,b,g,x) = p</code> , then <code>invweibull(a,b,g,p) = x</code>
<code>invweibullph(a,b,p)</code>	the inverse cumulative Weibull (proportional hazards) distribution with shape a and scale b : if <code>weibullph(a,b,x) = p</code> , then <code>invweibullph(a,b,p) = x</code>
<code>invweibullph(a,b,g,p)</code>	the inverse cumulative Weibull (proportional hazards) distribution with shape a , scale b , and location g : if <code>weibullph(a,b,g,x) = p</code> , then <code>invweibullph(a,b,g,p) = x</code>
<code>invweibullphtail(a,b,p)</code>	the inverse reverse cumulative Weibull (proportional hazards) distribution with shape a and scale b : if <code>weibullphtail(a,b,x) = p</code> , then <code>invweibullphtail(a,b,p) = x</code>
<code>invweibullphtail(a,b,g,p)</code>	the inverse reverse cumulative Weibull (proportional hazards) distribution with shape a , scale b , and location g : if <code>weibullphtail(a,b,g,x) = p</code> , then <code>invweibullphtail(a,b,g,p) = x</code>
<code>invweibulltail(a,b,p)</code>	the inverse reverse cumulative Weibull distribution with shape a and scale b : if <code>weibulltail(a,b,x) = p</code> , then <code>invweibulltail(a,b,p) = x</code>
<code>invweibulltail(a,b,g,p)</code>	the inverse reverse cumulative Weibull distribution with shape a , scale b , and location g : if <code>weibulltail(a,b,g,x) = p</code> , then <code>invweibulltail(a,b,g,p) = x</code>

<code>irecode(x, x_1, \dots, x_n)</code>	<i>missing</i> if x is missing or x_1, \dots, x_n is not weakly increasing; 0 if $x \leq x_1$; 1 if $x_1 < x \leq x_2$; 2 if $x_2 < x \leq x_3$; ...; n if $x > x_n$
<code>issymmetric(M)</code>	1 if the matrix is symmetric; otherwise, 0
<code>J(r, c, z)</code>	the $r \times c$ matrix containing elements z
<code>laplace(m, b, x)</code>	the cumulative Laplace distribution with mean m and scale parameter b
<code>laplaceden(m, b, x)</code>	the probability density of the Laplace distribution with mean m and scale parameter b
<code>laplacetail(m, b, x)</code>	the reverse cumulative (upper tail or survivor) Laplace distribution with mean m and scale parameter b
<code>ln(x)</code>	the natural logarithm, $\ln(x)$
<code>ln1m(x)</code>	the natural logarithm of $1 - x$ with higher precision than $\ln(1 - x)$ for small values of $ x $
<code>ln1p(x)</code>	the natural logarithm of $1 + x$ with higher precision than $\ln(1 + x)$ for small values of $ x $
<code>lncauchyden(a, b, x)</code>	the natural logarithm of the density of the Cauchy distribution with location parameter a and scale parameter b
<code>lnfactorial(n)</code>	the natural log of n factorial = $\ln(n!)$
<code>lngamma(x)</code>	$\ln\{\Gamma(x)\}$
<code>lnigammaden(a, b, x)</code>	the natural logarithm of the inverse gamma density, where a is the shape parameter and b is the scale parameter
<code>lnigaussianden(m, a, x)</code>	the natural logarithm of the inverse Gaussian density with mean m and shape parameter a
<code>lniwishartden(df, V, X)</code>	the natural logarithm of the density of the inverse Wishart distribution; missing if $df \leq n - 1$
<code>lnlaplaceden(m, b, x)</code>	the natural logarithm of the density of the Laplace distribution with mean m and scale parameter b
<code>lnmvnormalden(M, V, X)</code>	the natural logarithm of the multivariate normal density
<code>lnnormal(z)</code>	the natural logarithm of the cumulative standard normal distribution
<code>lnnormalden(z)</code>	the natural logarithm of the standard normal density, $N(0, 1)$
<code>lnnormalden(x, σ)</code>	the natural logarithm of the normal density with mean 0 and standard deviation σ
<code>lnnormalden(x, μ, σ)</code>	the natural logarithm of the normal density with mean μ and standard deviation σ , $N(\mu, \sigma^2)$
<code>lnwishartden(df, V, X)</code>	the natural logarithm of the density of the Wishart distribution; missing if $df \leq n - 1$
<code>log(x)</code>	a synonym for <code>ln(x)</code>
<code>log10(x)</code>	the base-10 logarithm of x
<code>log1m(x)</code>	a synonym for <code>ln1m(x)</code>
<code>log1p(x)</code>	a synonym for <code>ln1p(x)</code>
<code>logistic(x)</code>	the cumulative logistic distribution with mean 0 and standard deviation $\pi/\sqrt{3}$
<code>logistic(s, x)</code>	the cumulative logistic distribution with mean 0, scale s , and standard deviation $s\pi/\sqrt{3}$

<code>logistic(m, s, x)</code>	the cumulative logistic distribution with mean m , scale s , and standard deviation $s\pi/\sqrt{3}$
<code>logisticden(x)</code>	the density of the logistic distribution with mean 0 and standard deviation $\pi/\sqrt{3}$
<code>logisticden(s, x)</code>	the density of the logistic distribution with mean 0, scale s , and standard deviation $s\pi/\sqrt{3}$
<code>logisticden(m, s, x)</code>	the density of the logistic distribution with mean m , scale s , and standard deviation $s\pi/\sqrt{3}$
<code>logistictail(x)</code>	the reverse cumulative logistic distribution with mean 0 and standard deviation $\pi/\sqrt{3}$
<code>logistictail(s, x)</code>	the reverse cumulative logistic distribution with mean 0, scale s , and standard deviation $s\pi/\sqrt{3}$
<code>logistictail(m, s, x)</code>	the reverse cumulative logistic distribution with mean m , scale s , and standard deviation $s\pi/\sqrt{3}$
<code>logit(x)</code>	the log of the odds ratio of x , $\text{logit}(x) = \ln\{x/(1-x)\}$
<code>matmissing(M)</code>	1 if any elements of the matrix are missing; otherwise, 0
<code>matrix(exp)</code>	restricts name interpretation to scalars and matrices; see <code>scalar()</code>
<code>matuniform(r, c)</code>	the $r \times c$ matrices containing uniformly distributed pseudorandom numbers on the interval (0, 1)
<code>max(x_1, x_2, \dots, x_n)</code>	the maximum value of x_1, x_2, \dots, x_n
<code>maxbyte()</code>	the largest value that can be stored in storage type <code>byte</code>
<code>maxdouble()</code>	the largest value that can be stored in storage type <code>double</code>
<code>maxfloat()</code>	the largest value that can be stored in storage type <code>float</code>
<code>maxint()</code>	the largest value that can be stored in storage type <code>int</code>
<code>maxlong()</code>	the largest value that can be stored in storage type <code>long</code>
<code>mdy(M, D, Y)</code>	the e_d date (days since 01jan1960) corresponding to M, D, Y
<code>mdyhms(M, D, Y, h, m, s)</code>	the e_{tc} datetime (ms. since 01jan1960 00:00:00.000) corresponding to M, D, Y, h, m, s
<code>mi(x_1, x_2, \dots, x_n)</code>	a synonym for <code>missing(x_1, x_2, \dots, x_n)</code>
<code>min(x_1, x_2, \dots, x_n)</code>	the minimum value of x_1, x_2, \dots, x_n
<code>minbyte()</code>	the smallest value that can be stored in storage type <code>byte</code>
<code>mindouble()</code>	the smallest value that can be stored in storage type <code>double</code>
<code>minfloat()</code>	the smallest value that can be stored in storage type <code>float</code>
<code>minint()</code>	the smallest value that can be stored in storage type <code>int</code>
<code>minlong()</code>	the smallest value that can be stored in storage type <code>long</code>
<code>minutes(ms)</code>	$ms/60,000$
<code>missing(x_1, x_2, \dots, x_n)</code>	1 if any x_i evaluates to <i>missing</i> ; otherwise, 0
<code>mm(e_{tc})</code>	the minute corresponding to datetime e_{tc} (ms. since 01jan1960 00:00:00.000)
<code>mmC(e_{tC})</code>	the minute corresponding to datetime e_{tC} (ms. with leap seconds since 01jan1960 00:00:00.000)
<code>mod(x, y)</code>	the modulus of x with respect to y
<code>mofd(e_d)</code>	the e_m monthly date (months since 1960m1) containing date e_d

<code>month(e_d)</code>	the numeric month corresponding to date e_d
<code>monthly($s_1, s_2[, Y]$)</code>	the e_m monthly date (months since 1960m1) corresponding to s_1 based on s_2 and Y ; Y specifies <i>topyear</i> ; see <code>date()</code>
<code>mreldif(X, Y)</code>	the relative difference of X and Y , where the relative difference is defined as $\max_{i,j} \{ x_{ij} - y_{ij} / (y_{ij} + 1)\}$
<code>msofhours(h)</code>	$h \times 3,600,000$
<code>msofminutes(m)</code>	$m \times 60,000$
<code>msofseconds(s)</code>	$s \times 1,000$
<code>nbetaden(a, b, np, x)</code>	the probability density function of the noncentral beta distribution; 0 if $x < 0$ or $x > 1$
<code>nbinomial(n, k, p)</code>	the cumulative probability of the negative binomial distribution
<code>nbinomialp(n, k, p)</code>	the negative binomial probability
<code>nbinomialtail(n, k, p)</code>	the reverse cumulative probability of the negative binomial distribution
<code>nchi2(df, np, x)</code>	the cumulative noncentral χ^2 distribution; 0 if $x < 0$
<code>nchi2den(df, np, x)</code>	the probability density of the noncentral χ^2 distribution; 0 if $x < 0$
<code>nchi2tail(df, np, x)</code>	the reverse cumulative (upper tail or survivor) noncentral χ^2 distribution; 1 if $x < 0$
<code>nF(df_1, df_2, np, f)</code>	the cumulative noncentral F distribution with df_1 numerator and df_2 denominator degrees of freedom and noncentrality parameter np ; 0 if $f < 0$
<code>nFden(df_1, df_2, np, f)</code>	the probability density function of the noncentral F distribution with df_1 numerator and df_2 denominator degrees of freedom and noncentrality parameter np ; 0 if $f < 0$
<code>nFtail(df_1, df_2, np, f)</code>	the reverse cumulative (upper tail or survivor) noncentral F distribution with df_1 numerator and df_2 denominator degrees of freedom and noncentrality parameter np ; 1 if $f < 0$
<code>nibeta(a, b, np, x)</code>	the cumulative noncentral beta distribution; 0 if $x < 0$; or 1 if $x > 1$
<code>normal(z)</code>	the cumulative standard normal distribution
<code>normalden(z)</code>	the standard normal density, $N(0, 1)$
<code>normalden(x, σ)</code>	the normal density with mean 0 and standard deviation σ
<code>normalden(x, μ, σ)</code>	the normal density with mean μ and standard deviation σ , $N(\mu, \sigma^2)$
<code>npnchi2(df, x, p)</code>	the noncentrality parameter, np , for noncentral χ^2 : if $\text{nchi2}(df, np, x) = p$, then $\text{npnchi2}(df, x, p) = np$
<code>npnF(df_1, df_2, f, p)</code>	the noncentrality parameter, np , for the noncentral F : if $\text{nF}(df_1, df_2, np, f) = p$, then $\text{npnF}(df_1, df_2, f, p) = np$
<code>npnt(df, t, p)</code>	the noncentrality parameter, np , for the noncentral Student's t distribution: if $\text{nt}(df, np, t) = p$, then $\text{npnt}(df, t, p) = np$
<code>nt(df, np, t)</code>	the cumulative noncentral Student's t distribution with df degrees of freedom and noncentrality parameter np
<code>ntden(df, np, t)</code>	the probability density function of the noncentral Student's t distribution with df degrees of freedom and noncentrality parameter np
<code>nttail(df, np, t)</code>	the reverse cumulative (upper tail or survivor) noncentral Student's t distribution with df degrees of freedom and noncentrality parameter np

<code>nullmat(matname)</code>	use with the row-join (,) and column-join (\) operators
<code>plural(n,s)</code>	the plural of s if $n \neq \pm 1$
<code>plural(n,s₁,s₂)</code>	the plural of s_1 , as modified by or replaced with s_2 , if $n \neq \pm 1$
<code>poisson(m,k)</code>	the probability of observing <code>floor(k)</code> or fewer outcomes that are distributed as Poisson with mean m
<code>poissonp(m,k)</code>	the probability of observing <code>floor(k)</code> outcomes that are distributed as Poisson with mean m
<code>poisontail(m,k)</code>	the probability of observing <code>floor(k)</code> or more outcomes that are distributed as Poisson with mean m
<code>qofd(e_d)</code>	the e_q quarterly date (quarters since 1960q1) containing date e_d
<code>quarter(e_d)</code>	the numeric quarter of the year corresponding to date e_d
<code>quarterly(s₁,s₂[,Y])</code>	the e_q quarterly date (quarters since 1960q1) corresponding to s_1 based on s_2 and Y ; Y specifies <i>topyear</i> ; see <code>date()</code>
<code>r(name)</code>	the value of the stored result <code>r(name)</code> ; see [U] 18.8 Accessing results calculated by other programs
<code>rbeta(a,b)</code>	<code>beta(a,b)</code> random variates, where a and b are the beta distribution shape parameters
<code>rbinomial(n,p)</code>	<code>binomial(n,p)</code> random variates, where n is the number of trials and p is the success probability
<code>rcauchy(a,b)</code>	<code>Cauchy(a,b)</code> random variates, where a is the location parameter and b is the scale parameter
<code>rchi2(df)</code>	chi-squared, with df degrees of freedom, random variates
<code>recode(x,x₁,...,x_n)</code>	<i>missing</i> if x_1, x_2, \dots, x_n is not weakly increasing; x if x is missing; x_1 if $x \leq x_1$; x_2 if $x \leq x_2, \dots$; otherwise, x_n if $x > x_1, x_2, \dots, x_{n-1}$. $x_i \geq .$ is interpreted as $x_i = +\infty$
<code>real(s)</code>	s converted to numeric or <i>missing</i>
<code>regexm(s,re)</code>	performs a match of a regular expression and evaluates to 1 if regular expression re is satisfied by the ASCII string s ; otherwise, 0
<code>regexr(s₁,re,s₂)</code>	replaces the first substring within ASCII string s_1 that matches re with ASCII string s_2 and returns the resulting string
<code>regexs(n)</code>	subexpression n from a previous <code>regexm()</code> match, where $0 \leq n < 10$
<code>reldif(x,y)</code>	the “relative” difference $ x - y /(y + 1)$; 0 if both arguments are the same type of extended missing value; <i>missing</i> if only one argument is missing or if the two arguments are two different types of <i>missing</i>
<code>replay()</code>	1 if the first nonblank character of local macro ‘0’ is a comma, or if ‘0’ is empty
<code>return(name)</code>	the value of the to-be-stored result <code>r(name)</code> ; see [P] return
<code>rexponential(b)</code>	exponential random variates with scale b
<code>rgamma(a,b)</code>	<code>gamma(a,b)</code> random variates, where a is the gamma shape parameter and b is the scale parameter
<code>rhypergeometric(N,K,n)</code>	hypergeometric random variates
<code>rigaussian(m,a)</code>	inverse Gaussian random variates with mean m and shape parameter a
<code>rlaplace(m,b)</code>	<code>Laplace(m,b)</code> random variates with mean m and scale parameter b
<code>rlogistic()</code>	logistic variates with mean 0 and standard deviation $\pi/\sqrt{3}$

<code>rlogistic(s)</code>	logistic variates with mean 0, scale s , and standard deviation $s\pi/\sqrt{3}$
<code>rlogistic(m,s)</code>	logistic variates with mean m , scale s , and standard deviation $s\pi/\sqrt{3}$
<code>rnbinomial(n,p)</code>	negative binomial random variates
<code>rnormal()</code>	standard normal (Gaussian) random variates, that is, variates from a normal distribution with a mean of 0 and a standard deviation of 1
<code>rnormal(m)</code>	normal($m,1$) (Gaussian) random variates, where m is the mean and the standard deviation is 1
<code>rnormal(m,s)</code>	normal(m,s) (Gaussian) random variates, where m is the mean and s is the standard deviation
<code>round(x,y)</code> or <code>round(x)</code>	x rounded in units of y or x rounded to the nearest integer if the argument y is omitted; x (not “.”) if x is missing (meaning that <code>round(.a) = .a</code> and that <code>round(.a,y) = .a</code> if y is not missing) and if y is missing, then “.”) is returned
<code>roweqnumb(M,s)</code>	the equation number of M associated with row equation s ; <i>missing</i> if the row equation cannot be found
<code>rownfreeparms(M)</code>	the number of free parameters in rows of M
<code>rownumb(M,s)</code>	the row number of M associated with row name s ; <i>missing</i> if the row cannot be found
<code>rowsof(M)</code>	the number of rows of M
<code>rpoisson(m)</code>	Poisson(m) random variates, where m is the distribution mean
<code>rt(df)</code>	Student’s t random variates, where df is the degrees of freedom
<code>runiform()</code>	uniformly distributed random variates over the interval (0,1)
<code>runiform(a,b)</code>	uniformly distributed random variates over the interval (a,b)
<code>runiformint(a,b)</code>	uniformly distributed random integer variates on the interval [a,b]
<code>rweibull(a,b)</code>	Weibull variates with shape a and scale b
<code>rweibull(a,b,g)</code>	Weibull variates with shape a , scale b , and location g
<code>rweibullph(a,b)</code>	Weibull (proportional hazards) variates with shape a and scale b
<code>rweibullph(a,b,g)</code>	Weibull (proportional hazards) variates with shape a , scale b , and location g
<code>s(name)</code>	the value of stored result <code>s(name)</code> ; see [U] 18.8 Accessing results calculated by other programs
<code>scalar(exp)</code>	restricts name interpretation to scalars and matrices
<code>seconds(ms)</code>	$ms/1,000$
<code>sign(x)</code>	the sign of x : -1 if $x < 0$, 0 if $x = 0$, 1 if $x > 0$, or <i>missing</i> if x is missing
<code>sin(x)</code>	the sine of x , where x is in radians
<code>sinh(x)</code>	the hyperbolic sine of x
<code>smallestdouble()</code>	the smallest double-precision number greater than zero
<code>soundex(s)</code>	the soundex code for a string, s
<code>soundex_nara(s)</code>	the U.S. Census soundex code for a string, s
<code>sqrt(x)</code>	the square root of x
<code>ss(e_{tc})</code>	the second corresponding to datetime e_{tc} (ms. since 01jan1960 00:00:00.000)

<code>ssC(e_{tC})</code>	the second corresponding to datetime e_{tC} (ms. with leap seconds since 01jan1960 00:00:00.000)
<code>strcat(s_1, s_2)</code>	there is no <code>strcat()</code> function; instead the addition operator is used to concatenate strings
<code>strdup(s_1, n)</code>	there is no <code>strdup()</code> function; instead the multiplication operator is used to create multiple copies of strings
<code>string(n)</code>	a synonym for <code>strofreal(n)</code>
<code>string(n, s)</code>	a synonym for <code>strofreal(n, s)</code>
<code>stritrim(s)</code>	s with multiple, consecutive internal blanks (ASCII space character <code>char(32)</code>) collapsed to one blank
<code>strlen(s)</code>	the number of characters in ASCII s or length in bytes
<code>strlower(s)</code>	lowercase ASCII characters in string s
<code>strltrim(s)</code>	s without leading blanks (ASCII space character <code>char(32)</code>)
<code>strmatch(s_1, s_2)</code>	1 if s_1 matches the pattern s_2 ; otherwise, 0
<code>strofreal(n)</code>	n converted to a string
<code>strofreal(n, s)</code>	n converted to a string using the specified display format
<code>strpos(s_1, s_2)</code>	the position in s_1 at which s_2 is first found; otherwise, 0
<code>strproper(s)</code>	a string with the first ASCII letter and any other letters immediately following characters that are not letters capitalized; all other ASCII letters converted to lowercase
<code>strreverse(s)</code>	reverses the ASCII string s
<code>strrpos(s_1, s_2)</code>	the position in s_1 at which s_2 is last found; otherwise, 0
<code>strrtrim(s)</code>	s without trailing blanks (ASCII space character <code>char(32)</code>)
<code>strtoname(s [, p])</code>	s translated into a Stata 13 compatible name
<code>strtrim(s)</code>	s without leading and trailing blanks (ASCII space character <code>char(32)</code>); equivalent to <code>strltrim(strrtrim(s))</code>
<code>strupper(s)</code>	uppercase ASCII characters in string s
<code>subinstr(s_1, s_2, s_3, n)</code>	s_1 , where the first n occurrences in s_1 of s_2 have been replaced with s_3
<code>subinword(s_1, s_2, s_3, n)</code>	s_1 , where the first n occurrences in s_1 of s_2 as a word have been replaced with s_3
<code>substr(s, n_1, n_2)</code>	the substring of s , starting at n_1 , for a length of n_2
<code>sum(x)</code>	the running sum of x , treating missing values as zero
<code>sweep(M, i)</code>	matrix M with i th row/column swept
<code>t(df, t)</code>	the cumulative Student's t distribution with df degrees of freedom
<code>tan(x)</code>	the tangent of x , where x is in radians
<code>tanh(x)</code>	the hyperbolic tangent of x
<code>tC(l)</code>	convenience function to make typing dates and times in expressions easier
<code>tc(l)</code>	convenience function to make typing dates and times in expressions easier
<code>td(l)</code>	convenience function to make typing dates in expressions easier
<code>tden(df, t)</code>	the probability density function of Student's t distribution
<code>th(l)</code>	convenience function to make typing half-yearly dates in expressions easier

<code>tin(d_1, d_2)</code>	<i>true</i> if $d_1 \leq t \leq d_2$, where t is the time variable previously <code>tsset</code>
<code>tm(l)</code>	convenience function to make typing monthly dates in expressions easier
<code>tobytes($s[,n]$)</code>	escaped decimal or hex digit strings of up to 200 bytes of s
<code>tq(l)</code>	convenience function to make typing quarterly dates in expressions easier
<code>trace(M)</code>	the trace of matrix M
<code>trigamma(x)</code>	the second derivative of <code>lngamma(x)</code> = $d^2 \ln \Gamma(x) / dx^2$
<code>trunc(x)</code>	a synonym for <code>int(x)</code>
<code>ttail(df, t)</code>	the reverse cumulative (upper tail or survivor) Student's t distribution; the probability $T > t$
<code>tukeyprob(k, df, x)</code>	the cumulative Tukey's Studentized range distribution with k ranges and df degrees of freedom; 0 if $x < 0$
<code>tw(l)</code>	convenience function to make typing weekly dates in expressions easier
<code>twithin(d_1, d_2)</code>	<i>true</i> if $d_1 < t < d_2$, where t is the time variable previously <code>tsset</code>
<code>uchar(n)</code>	the Unicode character corresponding to Unicode code point n or an empty string if n is beyond the Unicode code-point range
<code>udstrlen(s)</code>	the number of display columns needed to display the Unicode string s in the Stata Results window
<code>udsubstr(s, n_1, n_2)</code>	the Unicode substring of s , starting at character n_1 , for n_2 display columns
<code>uisdigit(s)</code>	1 if the first Unicode character in s is a Unicode decimal digit; otherwise, 0
<code>uisletter(s)</code>	1 if the first Unicode character in s is a Unicode letter; otherwise, 0
<code>ustrcompare($s_1, s_2[,loc]$)</code>	compares two Unicode strings
<code>ustrcompareex($s_1, s_2, loc, st, case, cslv, norm, num, alt, fr$)</code>	compares two Unicode strings
<code>ustrfix($s[,rep]$)</code>	replaces each invalid UTF-8 sequence with a Unicode character
<code>ustrfrom($s, enc, mode$)</code>	converts the string s in encoding enc to a UTF-8 encoded Unicode string
<code>ustrinvalidcnt(s)</code>	the number of invalid UTF-8 sequences in s
<code>ustrleft(s, n)</code>	the first n Unicode characters of the Unicode string s
<code>ustrlen(s)</code>	the number of characters in the Unicode string s
<code>ustrlower($s[,loc]$)</code>	lowercase all characters of Unicode string s under the given locale loc
<code>ustrltrim(s)</code>	removes the leading Unicode whitespace characters and blanks from the Unicode string s
<code>ustrnormalize($s, norm$)</code>	normalizes Unicode string s to one of the five normalization forms specified by $norm$
<code>ustrpos($s_1, s_2[,n]$)</code>	the position in s_1 at which s_2 is first found; otherwise, 0
<code>ustrregexm($s, re[,noc]$)</code>	performs a match of a regular expression and evaluates to 1 if regular expression re is satisfied by the Unicode string s ; otherwise, 0
<code>ustrregexra($s_1, re, s_2[,noc]$)</code>	replaces all substrings within the Unicode string s_1 that match re with s_2 and returns the resulting string

<code>ustrregexrf(<i>s</i>₁,<i>re</i>,<i>s</i>₂[,<i>noc</i>])</code>	replaces the first substring within the Unicode string <i>s</i> ₁ that matches <i>re</i> with <i>s</i> ₂ and returns the resulting string
<code>ustrregexs(<i>n</i>)</code>	subexpression <i>n</i> from a previous <code>ustrregexm()</code> match
<code>ustrreverse(<i>s</i>)</code>	reverses the Unicode string <i>s</i>
<code>ustrright(<i>s</i>,<i>n</i>)</code>	the last <i>n</i> Unicode characters of the Unicode string <i>s</i>
<code>ustrrpos(<i>s</i>₁,<i>s</i>₂[,<i>n</i>])</code>	the position in <i>s</i> ₁ at which <i>s</i> ₂ is last found; otherwise, 0
<code>ustrrrtrim(<i>s</i>)</code>	remove trailing Unicode whitespace characters and blanks from the Unicode string <i>s</i>
<code>ustrsortkey(<i>s</i>[,<i>loc</i>])</code>	generates a null-terminated byte array that can be used by the <code>sort</code> command to produce the same order as <code>ustrcompare()</code>
<code>ustrsortkeyex(<i>s</i>,<i>loc</i>,<i>st</i>,<i>case</i>,<i>cslv</i>,<i>norm</i>,<i>num</i>,<i>alt</i>,<i>fr</i>)</code>	generates a null-terminated byte array that can be used by the <code>sort</code> command to produce the same order as <code>ustrcompare()</code>
<code>ustrtitle(<i>s</i>[,<i>loc</i>])</code>	a string with the first characters of Unicode words titlecased and other characters lowercased
<code>ustrto(<i>s</i>,<i>enc</i>,<i>mode</i>)</code>	converts the Unicode string <i>s</i> in UTF-8 encoding to a string in encoding <i>enc</i>
<code>ustrtohex(<i>s</i>[,<i>n</i>])</code>	escaped hex digit string of <i>s</i> up to 200 Unicode characters
<code>ustrtoname(<i>s</i>[,<i>p</i>])</code>	string <i>s</i> translated into a Stata name
<code>ustrtrim(<i>s</i>)</code>	removes leading and trailing Unicode whitespace characters and blanks from the Unicode string <i>s</i>
<code>ustrunescape(<i>s</i>)</code>	the Unicode string corresponding to the escaped sequences of <i>s</i>
<code>ustrupper(<i>s</i>[,<i>loc</i>])</code>	uppercase all characters in string <i>s</i> under the given locale <i>loc</i>
<code>ustrword(<i>s</i>,<i>n</i>[,<i>loc</i>])</code>	the <i>n</i> th Unicode word in the Unicode string <i>s</i>
<code>ustrwordcount(<i>s</i>[,<i>loc</i>])</code>	the number of nonempty Unicode words in the Unicode string <i>s</i>
<code>usubinstr(<i>s</i>₁,<i>s</i>₂,<i>s</i>₃,<i>n</i>)</code>	replaces the first <i>n</i> occurrences of the Unicode string <i>s</i> ₂ with the Unicode string <i>s</i> ₃ in <i>s</i> ₁
<code>usubstr(<i>s</i>,<i>n</i>₁,<i>n</i>₂)</code>	the Unicode substring of <i>s</i> , starting at <i>n</i> ₁ , for a length of <i>n</i> ₂
<code>vec(<i>M</i>)</code>	a column vector formed by listing the elements of <i>M</i> , starting with the first column and proceeding column by column
<code>vecdiag(<i>M</i>)</code>	the row vector containing the diagonal of matrix <i>M</i>
<code>week(<i>e</i>_d)</code>	the numeric week of the year corresponding to date <i>e</i> _d , the %td encoded date (days since 01jan1960)
<code>weekly(<i>s</i>₁,<i>s</i>₂[,<i>Y</i>])</code>	the <i>e</i> _w weekly date (weeks since 1960w1) corresponding to <i>s</i> ₁ based on <i>s</i> ₂ and <i>Y</i> ; <i>Y</i> specifies <i>topyear</i> ; see <code>date()</code>
<code>weibull(<i>a</i>,<i>b</i>,<i>x</i>)</code>	the cumulative Weibull distribution with shape <i>a</i> and scale <i>b</i>
<code>weibull(<i>a</i>,<i>b</i>,<i>g</i>,<i>x</i>)</code>	the cumulative Weibull distribution with shape <i>a</i> , scale <i>b</i> , and location <i>g</i>
<code>weibullden(<i>a</i>,<i>b</i>,<i>x</i>)</code>	the probability density function of the Weibull distribution with shape <i>a</i> and scale <i>b</i>
<code>weibullden(<i>a</i>,<i>b</i>,<i>g</i>,<i>x</i>)</code>	the probability density function of the Weibull distribution with shape <i>a</i> , scale <i>b</i> , and location <i>g</i>
<code>weibullph(<i>a</i>,<i>b</i>,<i>x</i>)</code>	the cumulative Weibull (proportional hazards) distribution with shape <i>a</i> and scale <i>b</i>

<code>weibullph(<i>a</i>,<i>b</i>,<i>g</i>,<i>x</i>)</code>	the cumulative Weibull (proportional hazards) distribution with shape <i>a</i> , scale <i>b</i> , and location <i>g</i>
<code>weibullphden(<i>a</i>,<i>b</i>,<i>x</i>)</code>	the probability density function of the Weibull (proportional hazards) distribution with shape <i>a</i> and scale <i>b</i>
<code>weibullphden(<i>a</i>,<i>b</i>,<i>g</i>,<i>x</i>)</code>	the probability density function of the Weibull (proportional hazards) distribution with shape <i>a</i> , scale <i>b</i> , and location <i>g</i>
<code>weibullphtail(<i>a</i>,<i>b</i>,<i>x</i>)</code>	the reverse cumulative Weibull (proportional hazards) distribution with shape <i>a</i> and scale <i>b</i>
<code>weibullphtail(<i>a</i>,<i>b</i>,<i>g</i>,<i>x</i>)</code>	the reverse cumulative Weibull (proportional hazards) distribution with shape <i>a</i> , scale <i>b</i> , and location <i>g</i>
<code>weibulltail(<i>a</i>,<i>b</i>,<i>x</i>)</code>	the reverse cumulative Weibull distribution with shape <i>a</i> and scale <i>b</i>
<code>weibulltail(<i>a</i>,<i>b</i>,<i>g</i>,<i>x</i>)</code>	the reverse cumulative Weibull distribution with shape <i>a</i> , scale <i>b</i> , and location <i>g</i>
<code>wofd(<i>e_d</i>)</code>	the <i>e_w</i> weekly date (weeks since 1960w1) containing date <i>e_d</i>
<code>word(<i>s</i>,<i>n</i>)</code>	the <i>n</i> th word in <i>s</i> ; missing ("") if <i>n</i> is missing
<code>wordbreaklocale(<i>loc</i>,<i>type</i>)</code>	the most closely related locale supported by ICU from <i>loc</i> if <i>type</i> is 1, the actual locale where the word-boundary analysis data come from if <i>type</i> is 2; or an empty string is returned for any other <i>type</i>
<code>wordcount(<i>s</i>)</code>	the number of words in <i>s</i>
<code>year(<i>e_d</i>)</code>	the numeric year corresponding to date <i>e_d</i>
<code>yearly(<i>s₁</i>,<i>s₂</i>[,<i>Y</i>])</code>	the <i>e_y</i> yearly date (year) corresponding to <i>s₁</i> based on <i>s₂</i> and <i>Y</i> ; <i>Y</i> specifies <i>topyear</i> ; see <code>date()</code>
<code>yh(<i>Y</i>,<i>H</i>)</code>	the <i>e_h</i> half-yearly date (half-years since 1960h1) corresponding to year <i>Y</i> , half-year <i>H</i>
<code>ym(<i>Y</i>,<i>M</i>)</code>	the <i>e_m</i> monthly date (months since 1960m1) corresponding to year <i>Y</i> , month <i>M</i>
<code>yofd(<i>e_d</i>)</code>	the <i>e_y</i> yearly date (year) containing date <i>e_d</i>
<code>yq(<i>Y</i>,<i>Q</i>)</code>	the <i>e_q</i> quarterly date (quarters since 1960q1) corresponding to year <i>Y</i> , quarter <i>Q</i>
<code>yw(<i>Y</i>,<i>W</i>)</code>	the <i>e_w</i> weekly date (weeks since 1960w1) corresponding to year <i>Y</i> , week <i>W</i>

Also see

[FN] **Functions by category**

[D] **egen** — Extensions to generate

[D] **generate** — Create or change contents of variable

[M-4] **Intro** — Categorical guide to Mata functions

[U] **13.3 Functions**

Contents

<code>bofd("cal",e_d)</code>	the e_b business date corresponding to e_d
<code>Cdhms(e_d, h, m, s)</code>	the e_{tC} datetime (ms. with leap seconds since 01jan1960 00:00:00.000) corresponding to e_d, h, m, s
<code>Chms(h, m, s)</code>	the e_{tC} datetime (ms. with leap seconds since 01jan1960 00:00:00.000) corresponding to h, m, s on 01jan1960
<code>Clock($s_1, s_2[, Y]$)</code>	the e_{tC} datetime (ms. with leap seconds since 01jan1960 00:00:00.000) corresponding to s_1 based on s_2 and Y
<code>clock($s_1, s_2[, Y]$)</code>	the e_{tc} datetime (ms. since 01jan1960 00:00:00.000) corresponding to s_1 based on s_2 and Y
<code>Cmdyhms(M, D, Y, h, m, s)</code>	the e_{tC} datetime (ms. with leap seconds since 01jan1960 00:00:00.000) corresponding to M, D, Y, h, m, s
<code>Cofc(e_{tc})</code>	the e_{tC} datetime (ms. with leap seconds since 01jan1960 00:00:00.000) of e_{tc} (ms. without leap seconds since 01jan1960 00:00:00.000)
<code>cofC(e_{tC})</code>	the e_{tc} datetime (ms. without leap seconds since 01jan1960 00:00:00.000) of e_{tC} (ms. with leap seconds since 01jan1960 00:00:00.000)
<code>Cofd(e_d)</code>	the e_{tC} datetime (ms. with leap seconds since 01jan1960 00:00:00.000) of date e_d at time 00:00:00.000
<code>cofd(e_d)</code>	the e_{tc} datetime (ms. since 01jan1960 00:00:00.000) of date e_d at time 00:00:00.000
<code>daily($s_1, s_2[, Y]$)</code>	a synonym for <code>date($s_1, s_2[, Y]$)</code>
<code>date($s_1, s_2[, Y]$)</code>	the e_d date (days since 01jan1960) corresponding to s_1 based on s_2 and Y
<code>day(e_d)</code>	the numeric day of the month corresponding to e_d
<code>dhms(e_d, h, m, s)</code>	the e_{tc} datetime (ms. since 01jan1960 00:00:00.000) corresponding to e_d, h, m , and s
<code>dofb($e_b, "cal"$)</code>	the e_d datetime corresponding to e_b
<code>dofC(e_{tC})</code>	the e_d date (days since 01jan1960) of datetime e_{tC} (ms. with leap seconds since 01jan1960 00:00:00.000)
<code>dofc(e_{tc})</code>	the e_d date (days since 01jan1960) of datetime e_{tc} (ms. since 01jan1960 00:00:00.000)
<code>dofh(e_h)</code>	the e_d date (days since 01jan1960) of the start of half-year e_h
<code>dofm(e_m)</code>	the e_d date (days since 01jan1960) of the start of month e_m
<code>dofq(e_q)</code>	the e_d date (days since 01jan1960) of the start of quarter e_q
<code>dofw(e_w)</code>	the e_d date (days since 01jan1960) of the start of week e_w
<code>dofy(e_y)</code>	the e_d date (days since 01jan1960) of 01jan in year e_y

<code>dow(e_d)</code>	the numeric day of the week corresponding to date e_d ; 0 = Sunday, 1 = Monday, ..., 6 = Saturday
<code>doy(e_d)</code>	the numeric day of the year corresponding to date e_d
<code>halfyear(e_d)</code>	the numeric half of the year corresponding to date e_d
<code>halfyearly($s_1, s_2[, Y]$)</code>	the e_h half-yearly date (half-years since 1960h1) corresponding to s_1 based on s_2 and Y ; Y specifies <i>topyear</i> ; see date()
<code>hh(e_{tc})</code>	the hour corresponding to datetime e_{tc} (ms. since 01jan1960 00:00:00.000)
<code>hhC(e_{tC})</code>	the hour corresponding to datetime e_{tC} (ms. with leap seconds since 01jan1960 00:00:00.000)
<code>hms(h, m, s)</code>	the e_{tc} datetime (ms. since 01jan1960 00:00:00.000) corresponding to h, m, s on 01jan1960
<code>hofd(e_d)</code>	the e_h half-yearly date (half years since 1960h1) containing date e_d
<code>hours(ms)</code>	$ms/3,600,000$
<code>mdy(M, D, Y)</code>	the e_d date (days since 01jan1960) corresponding to M, D, Y
<code>mdyhms(M, D, Y, h, m, s)</code>	the e_{tc} datetime (ms. since 01jan1960 00:00:00.000) corresponding to M, D, Y, h, m, s
<code>minutes(ms)</code>	$ms/60,000$
<code>mm(e_{tc})</code>	the minute corresponding to datetime e_{tc} (ms. since 01jan1960 00:00:00.000)
<code>mmC(e_{tC})</code>	the minute corresponding to datetime e_{tC} (ms. with leap seconds since 01jan1960 00:00:00.000)
<code>mofd(e_d)</code>	the e_m monthly date (months since 1960m1) containing date e_d
<code>month(e_d)</code>	the numeric month corresponding to date e_d
<code>monthly($s_1, s_2[, Y]$)</code>	the e_m monthly date (months since 1960m1) corresponding to s_1 based on s_2 and Y ; Y specifies <i>topyear</i> ; see date()
<code>msofhours(h)</code>	$h \times 3,600,000$
<code>msofminutes(m)</code>	$m \times 60,000$
<code>msofseconds(s)</code>	$s \times 1,000$
<code>qofd(e_d)</code>	the e_q quarterly date (quarters since 1960q1) containing date e_d
<code>quarter(e_d)</code>	the numeric quarter of the year corresponding to date e_d
<code>quarterly($s_1, s_2[, Y]$)</code>	the e_q quarterly date (quarters since 1960q1) corresponding to s_1 based on s_2 and Y ; Y specifies <i>topyear</i> ; see date()
<code>seconds(ms)</code>	$ms/1,000$
<code>ss(e_{tc})</code>	the second corresponding to datetime e_{tc} (ms. since 01jan1960 00:00:00.000)
<code>ssC(e_{tC})</code>	the second corresponding to datetime e_{tC} (ms. with leap seconds since 01jan1960 00:00:00.000)
<code>tC(l)</code>	convenience function to make typing dates and times in expressions easier
<code>tc(l)</code>	convenience function to make typing dates and times in expressions easier
<code>td(l)</code>	convenience function to make typing dates in expressions easier
<code>th(l)</code>	convenience function to make typing half-yearly dates in expressions easier

<code>tm(<i>l</i>)</code>	convenience function to make typing monthly dates in expressions easier
<code>tq(<i>l</i>)</code>	convenience function to make typing quarterly dates in expressions easier
<code>tw(<i>l</i>)</code>	convenience function to make typing weekly dates in expressions easier
<code>week(<i>e_d</i>)</code>	the numeric week of the year corresponding to date <i>e_d</i> , the %td encoded date (days since 01jan1960)
<code>weekly(<i>s₁</i>, <i>s₂</i> [, <i>Y</i>])</code>	the <i>e_w</i> weekly date (weeks since 1960w1) corresponding to <i>s₁</i> based on <i>s₂</i> and <i>Y</i> ; <i>Y</i> specifies <i>topyear</i> ; see date()
<code>wofd(<i>e_d</i>)</code>	the <i>e_w</i> weekly date (weeks since 1960w1) containing date <i>e_d</i>
<code>year(<i>e_d</i>)</code>	the numeric year corresponding to date <i>e_d</i>
<code>yearly(<i>s₁</i>, <i>s₂</i> [, <i>Y</i>])</code>	the <i>e_y</i> yearly date (year) corresponding to <i>s₁</i> based on <i>s₂</i> and <i>Y</i> ; <i>Y</i> specifies <i>topyear</i> ; see date()
<code>yh(<i>Y</i>, <i>H</i>)</code>	the <i>e_h</i> half-yearly date (half-years since 1960h1) corresponding to year <i>Y</i> , half-year <i>H</i>
<code>ym(<i>Y</i>, <i>M</i>)</code>	the <i>e_m</i> monthly date (months since 1960m1) corresponding to year <i>Y</i> , month <i>M</i>
<code>yofd(<i>e_d</i>)</code>	the <i>e_y</i> yearly date (year) containing date <i>e_d</i>
<code>yq(<i>Y</i>, <i>Q</i>)</code>	the <i>e_q</i> quarterly date (quarters since 1960q1) corresponding to year <i>Y</i> , quarter <i>Q</i>
<code>yw(<i>Y</i>, <i>W</i>)</code>	the <i>e_w</i> weekly date (weeks since 1960w1) corresponding to year <i>Y</i> , week <i>W</i>

Functions

Stata's date and time functions are described with examples in [\[U\] 25 Working with dates and times](#) and [\[D\] Datetime](#). What follows is a technical description. We use the following notation:

<i>e_b</i>	%tb business calendar date (days)
<i>e_{tc}</i>	%tc encoded datetime (ms. since 01jan1960 00:00:00.000)
<i>e_{tC}</i>	%tC encoded datetime (ms. with leap seconds since 01jan1960 00:00:00.000)
<i>e_d</i>	%td encoded date (days since 01jan1960)
<i>e_w</i>	%tw encoded weekly date (weeks since 1960w1)
<i>e_m</i>	%tm encoded monthly date (months since 1960m1)
<i>e_q</i>	%tq encoded quarterly date (quarters since 1960q1)
<i>e_h</i>	%th encoded half-yearly date (half-years since 1960h1)
<i>e_y</i>	%ty encoded yearly date (years)
<i>M</i>	month, 1–12
<i>D</i>	day of month, 1–31
<i>Y</i>	year, 0100–9999
<i>h</i>	hour, 0–23
<i>m</i>	minute, 0–59
<i>s</i>	second, 0–59 or 60 if leap seconds
<i>W</i>	week number, 1–52
<i>Q</i>	quarter number, 1–4
<i>H</i>	half-year number, 1 or 2

The date and time functions, where integer arguments are required, allow noninteger values and use the `floor()` of the value.

A Stata date-and-time (`%t`) variable is recorded as the milliseconds, days, weeks, etc., depending upon the units from 01jan1960; negative values indicate dates and times before 01jan1960. Allowable dates and times are those between 01jan0100 and 31dec9999, inclusive, but all functions are based on the Gregorian calendar, and values do not correspond to historical dates before Friday, 15oct1582.

`bofd("cal", e_d)`

Description: the e_b business date corresponding to e_d

Domain cal : business calendar names and formats

Domain e_d : `%td` as defined by business calendar named cal

Range: as defined by business calendar named cal

`Cdhms(e_d, h, m, s)`

Description: the e_{tC} datetime (ms. with leap seconds since 01jan1960 00:00:00.000) corresponding to e_d, h, m, s

Domain e_d : `%td` dates 01jan0100 to 31dec9999 (integers $-679,350$ to $2,936,549$)

Domain h : integers 0 to 23

Domain m : integers 0 to 59

Domain s : reals 0.000 to 60.999

Range: datetimes 01jan0100 00:00:00.000 to 31dec9999 23:59:59.999
(integers $-58,695,840,000,000$ to $> 253,717,919,999,999$) or *missing*

`Chms(h, m, s)`

Description: the e_{tC} datetime (ms. with leap seconds since 01jan1960 00:00:00.000) corresponding to h, m, s on 01jan1960

Domain h : integers 0 to 23

Domain m : integers 0 to 59

Domain s : reals 0.000 to 60.999

Range: datetimes 01jan0100 00:00:00.000 to 31dec9999 23:59:59.999
(integers $-58,695,840,000,000$ to $> 253,717,919,999,999$) or *missing*

`Clock(s_1, s_2 [, Y])`

Description: the e_{tC} datetime (ms. with leap seconds since 01jan1960 00:00:00.000) corresponding to s_1 based on s_2 and Y

Function `Clock()` works the same as function `clock()` except that `Clock()` returns a leap second-adjusted `%tC` value rather than an unadjusted `%tC` value. Use `Clock()` only if original time values have been adjusted for leap seconds.

Domain s_1 : strings

Domain s_2 : strings

Domain Y : integers 1000 to 9998 (but probably 2001 to 2099)

Range: datetimes 01jan0100 00:00:00.000 to 31dec9999 23:59:59.999
(integers $-58,695,840,000,000$ to $> 253,717,919,999,999$) or *missing*

`clock($s_1, s_2[, Y]$)`

Description: the e_{tc} datetime (ms. since 01jan1960 00:00:00.000) corresponding to s_1 based on s_2 and Y

s_1 contains the date, time, or both, recorded as a string, in virtually any format. Months can be spelled out, abbreviated (to three characters), or indicated as numbers; years can include or exclude the century; blanks and punctuation are allowed.

s_2 is any permutation of M, D, [##]Y, h, m, and s, with their order defining the order that month, day, year, hour, minute, and second occur (and whether they occur) in s_1 . ##, if specified, indicates the default century for two-digit years in s_1 . For instance, $s_2 = \text{"MD19Y hm"}$ would translate $s_1 = \text{"11/15/91 21:14"}$ as 15nov1991 21:14. The space in "MD19Y hm" was not significant and the string would have translated just as well with "MD19Yhm".

Y provides an alternate way of handling two-digit years. Y specifies the largest year that is to be returned when a two-digit year is encountered; see function `date()` below. If neither ## nor Y is specified, `clock()` returns *missing* when it encounters a two-digit year.

Domain s_1 : strings

Domain s_2 : strings

Domain Y : integers 1000 to 9998 (but probably 2001 to 2099)

Range: datetimes 01jan0100 00:00:00.000 to 31dec9999 23:59:59.999
(integers $-58,695,840,000,000$ to $253,717,919,999,999$) or *missing*

`Cmdyhms(M, D, Y, h, m, s)`

Description: the e_{tc} datetime (ms. with leap seconds since 01jan1960 00:00:00.000) corresponding to M, D, Y, h, m, s

Domain M : integers 1 to 12

Domain D : integers 1 to 31

Domain Y : integers 0100 to 9999 (but probably 1800 to 2100)

Domain h : integers 0 to 23

Domain m : integers 0 to 59

Domain s : reals 0.000 to 60.999

Range: datetimes 01jan0100 00:00:00.000 to 31dec9999 23:59:59.999
(integers $-58,695,840,000,000$ to $> 253,717,919,999,999$) or *missing*

`Cofc(e_{tc})`

Description: the e_{tc} datetime (ms. with leap seconds since 01jan1960 00:00:00.000) of e_{tc} (ms. without leap seconds since 01jan1960 00:00:00.000)

Domain e_{tc} : datetimes 01jan0100 00:00:00.000 to 31dec9999 23:59:59.999
(integers $-58,695,840,000,000$ to $253,717,919,999,999$)

Range: datetimes 01jan0100 00:00:00.000 to 31dec9999 23:59:59.999
(integers $-58,695,840,000,000$ to $> 253,717,919,999,999$)

`cofC(e_{tc})`

Description: the e_{tc} datetime (ms. with leap seconds since 01jan1960 00:00:00.000) of e_{tc} (ms. without leap seconds since 01jan1960 00:00:00.000)

Domain e_{tc} : datetimes 01jan0100 00:00:00.000 to 31dec9999 23:59:59.999
(integers $-58,695,840,000,000$ to $> 253,717,919,999,999$)

Range: datetimes 01jan0100 00:00:00.000 to 31dec9999 23:59:59.999
(integers $-58,695,840,000,000$ to $253,717,919,999,999$)

Cofd(e_d)

Description: the e_{tC} datetime (ms. with leap seconds since 01jan1960 00:00:00.000) of date e_d at time 00:00:00.000
Domain e_d : %td dates 01jan0100 to 31dec9999 (integers $-679,350$ to $2,936,549$)
Range: datetimes 01jan0100 00:00:00.000 to 31dec9999 23:59:59.999
(integers $-58,695,840,000,000$ to $>253,717,919,999,999$)

cofd(e_d)

Description: the e_{tc} datetime (ms. since 01jan1960 00:00:00.000) of date e_d at time 00:00:00.000
Domain e_d : %td dates 01jan0100 to 31dec9999 (integers $-679,350$ to $2,936,549$)
Range: datetimes 01jan0100 00:00:00.000 to 31dec9999 23:59:59.999
(integers $-58,695,840,000,000$ to $253,717,919,999,999$)

daily($s_1, s_2[, Y]$)

Description: a synonym for **date($s_1, s_2[, Y]$)**

date($s_1, s_2[, Y]$)

Description: the e_d date (days since 01jan1960) corresponding to s_1 based on s_2 and Y

s_1 contains the date, recorded as a string, in virtually any format. Months can be spelled out, abbreviated (to three characters), or indicated as numbers; years can include or exclude the century; blanks and punctuation are allowed.

s_2 is any permutation of M, D, and $[##]Y$, with their order defining the order that month, day, and year occur in s_1 . $##$, if specified, indicates the default century for two-digit years in s_1 . For instance, $s_2 = \text{"MD19Y"}$ would translate $s_1 = \text{"11/15/91"}$ as 15nov1991.

Y provides an alternate way of handling two-digit years. When a two-digit year is encountered, the largest year, *topyear*, that does not exceed Y is returned.

```
date("1/15/08", "MDY", 1999) = 15jan1908
date("1/15/08", "MDY", 2019) = 15jan2008

date("1/15/51", "MDY", 2000) = 15jan1951
date("1/15/50", "MDY", 2000) = 15jan1950
date("1/15/49", "MDY", 2000) = 15jan1949

date("1/15/01", "MDY", 2050) = 15jan2001
date("1/15/00", "MDY", 2050) = 15jan2000
```

If neither $##$ nor Y is specified, **date()** returns *missing* when it encounters a two-digit year. See *Working with two-digit years* in [D] **Datetime translation** for more information.

Domain s_1 : strings

Domain s_2 : strings

Domain Y : integers 1000 to 9998 (but probably 2001 to 2099)

Range: %td dates 01jan0100 to 31dec9999 (integers $-679,350$ to $2,936,549$) or *missing*

day(e_d)

Description: the numeric day of the month corresponding to e_d

Domain e_d : %td dates 01jan0100 to 31dec9999 (integers $-679,350$ to $2,936,549$)

Range: integers 1 to 31 or *missing*

dhms(e_d, h, m, s)

Description: the e_{tc} datetime (ms. since 01jan1960 00:00:00.000) corresponding to e_d , h , m , and

Domain e_d : %td dates 01jan0100 to 31dec9999 (integers $-679,350$ to $2,936,549$)

Domain h : integers 0 to 23

Domain m : integers 0 to 59

Domain s : reals 0.000 to 59.999

Range: datetimes 01jan0100 00:00:00.000 to 31dec9999 23:59:59.999
(integers $-58,695,840,000,000$ to $253,717,919,999,999$) or *missing*

dofb($e_b, "cal"$)

Description: the e_d datetime corresponding to e_b

Domain e_b : %tb as defined by business calendar named *cal*

Domain *cal*: business calendar names and formats

Range: as defined by business calendar named *cal*

dofC(e_{tC})

Description: the e_d date (days since 01jan1960) of datetime e_{tC} (ms. with leap seconds since 01jan1960 00:00:00.000)

Domain e_{tC} : datetimes 01jan0100 00:00:00.000 to 31dec9999 23:59:59.999
(integers $-58,695,840,000,000$ to $>253,717,919,999,999$)

Range: %td dates 01jan0100 to 31dec9999 (integers $-679,350$ to $2,936,549$)

dofc(e_{tc})

Description: the e_d date (days since 01jan1960) of datetime e_{tc} (ms. since 01jan1960 00:00:00.000)

Domain e_{tc} : datetimes 01jan0100 00:00:00.000 to 31dec9999 23:59:59.999
(integers $-58,695,840,000,000$ to $253,717,919,999,999$)

Range: %td dates 01jan0100 to 31dec9999 (integers $-679,350$ to $2,936,549$)

dofh(e_h)

Description: the e_d date (days since 01jan1960) of the start of half-year e_h

Domain e_h : %th dates 0100h1 to 9999h2 (integers $-3,720$ to $16,079$)

Range: %td dates 01jan0100 to 01jul9999 (integers $-679,350$ to $2,936,366$)

dofm(e_m)

Description: the e_d date (days since 01jan1960) of the start of month e_m

Domain e_m : %tm dates 0100m1 to 9999m12 (integers $-22,320$ to $96,479$)

Range: %td dates 01jan0100 to 01dec9999 (integers $-679,350$ to $2,936,519$)

dofq(e_q)

Description: the e_d date (days since 01jan1960) of the start of quarter e_q

Domain e_q : %tq dates 0100q1 to 9999q4 (integers $-7,440$ to $32,159$)

Range: %td dates 01jan0100 to 01oct9999 (integers $-679,350$ to $2,936,458$)

dofw(e_w)

Description: the e_d date (days since 01jan1960) of the start of week e_w

Domain e_w : %tw dates 0100w1 to 9999w52 (integers $-96,720$ to $418,079$)

Range: %td dates 01jan0100 to 24dec9999 (integers $-679,350$ to $2,936,542$)

dofy(e_y)

Description: the e_d date (days since 01jan1960) of 01jan in year e_y

Domain e_y : %ty dates 0100 to 9999 (integers 0100 to 9999)

Range: %td dates 01jan0100 to 01jan9999 (integers $-679,350$ to $2,936,185$)

dow(e_d)

Description: the numeric day of the week corresponding to date e_d ; 0 = Sunday, 1 = Monday, ..., 6 = Saturday

Domain e_d : %td dates 01jan0100 to 31dec9999 (integers $-679,350$ to $2,936,549$)

Range: integers 0 to 6 or *missing*

doy(e_d)

Description: the numeric day of the year corresponding to date e_d

Domain e_d : %td dates 01jan0100 to 31dec9999 (integers $-679,350$ to $2,936,549$)

Range: integers 1 to 366 or *missing*

halfyear(e_d)

Description: the numeric half of the year corresponding to date e_d

Domain e_d : %td dates 01jan0100 to 31dec9999 (integers $-679,350$ to $2,936,549$)

Range: integers 1, 2, or *missing*

halfyearly(s_1, s_2 [, Y])

Description: the e_h half-yearly date (half-years since 1960h1) corresponding to s_1 based on s_2 and Y ; Y specifies *topyear*; see [date\(\)](#)

Domain s_1 : strings

Domain s_2 : strings "HY" and "YH"; Y may be prefixed with ##

Domain Y : integers 1000 to 9998 (but probably 2001 to 2099)

Range: %th dates 0100h1 to 9999h2 (integers $-3,720$ to $16,079$) or *missing*

hh(e_{tc})

Description: the hour corresponding to datetime e_{tc} (ms. since 01jan1960 00:00:00.000)

Domain e_{tc} : datetimes 01jan0100 00:00:00.000 to 31dec9999 23:59:59.999
(integers $-58,695,840,000,000$ to $253,717,919,999,999$)

Range: integers 0 through 23, *missing*

hhC(e_{tC})

Description: the hour corresponding to datetime e_{tC} (ms. with leap seconds since 01jan1960 00:00:00.000)

Domain e_{tC} : datetimes 01jan0100 00:00:00.000 to 31dec9999 23:59:59.999
(integers $-58,695,840,000,000$ to $>253,717,919,999,999$)

Range: integers 0 through 23, *missing*

hms(h, m, s)

Description: the e_{tc} datetime (ms. since 01jan1960 00:00:00.000) corresponding to h, m, s on 01jan1960

Domain h : integers 0 to 23

Domain m : integers 0 to 59

Domain s : reals 0.000 to 59.999

Range: datetimes 01jan1960 00:00:00.000 to 01jan1960 23:59:59.999 (integers 0 to $86,399,999$ or *missing*)

hofd(e_d)

Description: the e_h half-yearly date (half years since 1960h1) containing date e_d
 Domain e_d : %td dates 01jan0100 to 31dec9999 (integers $-679,350$ to $2,936,549$)
 Range: %th dates 0100h1 to 9999h2 (integers $-3,720$ to $16,079$)

hours(ms)

Description: $ms/3,600,000$
 Domain ms : real; milliseconds
 Range: real or *missing*

mdy(M, D, Y)

Description: the e_d date (days since 01jan1960) corresponding to M, D, Y
 Domain M : integers 1 to 12
 Domain D : integers 1 to 31
 Domain Y : integers 0100 to 9999 (but probably 1800 to 2100)
 Range: %td dates 01jan0100 to 31dec9999 (integers $-679,350$ to $2,936,549$) or *missing*

mdyhms(M, D, Y, h, m, s)

Description: the e_{tc} datetime (ms. since 01jan1960 00:00:00.000) corresponding to M, D, Y, h, m, s
 Domain M : integers 1 to 12
 Domain D : integers 1 to 31
 Domain Y : integers 0100 to 9999 (but probably 1800 to 2100)
 Domain h : integers 0 to 23
 Domain m : integers 0 to 59
 Domain s : reals 0.000 to 59.999
 Range: datetimes 01jan0100 00:00:00.000 to 31dec9999 23:59:59.999
 (integers $-58,695,840,000,000$ to $253,717,919,999,999$) or *missing*

minutes(ms)

Description: $ms/60,000$
 Domain ms : real; milliseconds
 Range: real or *missing*

mm(e_{tc})

Description: the minute corresponding to datetime e_{tc} (ms. since 01jan1960 00:00:00.000)
 Domain e_{tc} : datetimes 01jan0100 00:00:00.000 to 31dec9999 23:59:59.999
 (integers $-58,695,840,000,000$ to $253,717,919,999,999$)
 Range: integers 0 through 59, *missing*

mmC(e_{tC})

Description: the minute corresponding to datetime e_{tC} (ms. with leap seconds since 01jan1960 00:00:00.000)
 Domain e_{tC} : datetimes 01jan0100 00:00:00.000 to 31dec9999 23:59:59.999
 (integers $-58,695,840,000,000$ to $> 253,717,919,999,999$)
 Range: integers 0 through 59, *missing*

moFd(e_d)

Description: the e_m monthly date (months since 1960m1) containing date e_d
 Domain e_d : %td dates 01jan0100 to 31dec9999 (integers $-679,350$ to $2,936,549$)
 Range: %tm dates 0100m1 to 9999m12 (integers $-22,320$ to $96,479$)

month(e_d)

Description: the numeric month corresponding to date e_d

Domain e_d : %td dates 01jan0100 to 31dec9999 (integers $-679,350$ to $2,936,549$)

Range: integers 1 to 12 or *missing*

monthly(s_1, s_2 [, Y])

Description: the e_m monthly date (months since 1960m1) corresponding to s_1 based on s_2 and Y ; Y specifies *topyear*; see [date\(\)](#)

Domain s_1 : strings

Domain s_2 : strings "MY" and "YM"; Y may be prefixed with ##

Domain Y : integers 1000 to 9998 (but probably 2001 to 2099)

Range: %tm dates 0100m1 to 9999m12 (integers $-22,320$ to $96,479$) or *missing*

msofhours(h)

Description: $h \times 3,600,000$

Domain h : real; hours

Range: real or *missing*; milliseconds

msofminutes(m)

Description: $m \times 60,000$

Domain m : real; minutes

Range: real or *missing*; milliseconds

msofseconds(s)

Description: $s \times 1,000$

Domain s : real; seconds

Range: real or *missing*; milliseconds

qofd(e_d)

Description: the e_q quarterly date (quarters since 1960q1) containing date e_d

Domain e_d : %td dates 01jan0100 to 31dec9999 (integers $-679,350$ to $2,936,549$)

Range: %tq dates 0100q1 to 9999q4 (integers $-7,440$ to $32,159$)

quarter(e_d)

Description: the numeric quarter of the year corresponding to date e_d

Domain e_d : %td dates 01jan0100 to 31dec9999 (integers $-679,350$ to $2,936,549$)

Range: integers 1 to 4 or *missing*

quarterly(s_1, s_2 [, Y])

Description: the e_q quarterly date (quarters since 1960q1) corresponding to s_1 based on s_2 and Y ; Y specifies *topyear*; see [date\(\)](#)

Domain s_1 : strings

Domain s_2 : strings "QY" and "YQ"; Y may be prefixed with ##

Domain Y : integers 1000 to 9998 (but probably 2001 to 2099)

Range: %tq dates 0100q1 to 9999q4 (integers $-7,440$ to $32,159$) or *missing*

seconds(ms)

Description: $ms/1,000$

Domain ms : real; milliseconds

Range: real or *missing*

ss(e_{tc})

Description: the second corresponding to datetime e_{tc} (ms. since 01jan1960 00:00:00.000)
 Domain e_{tc} : datetimes 01jan0100 00:00:00.000 to 31dec9999 23:59:59.999
 (integers $-58,695,840,000,000$ to $253,717,919,999,999$)
 Range: real 0.000 through 59.999, *missing*

ssC(e_{tC})

Description: the second corresponding to datetime e_{tC} (ms. with leap seconds since 01jan1960 00:00:00.000)
 Domain e_{tC} : datetimes 01jan0100 00:00:00.000 to 31dec9999 23:59:59.999
 (integers $-58,695,840,000,000$ to $> 253,717,919,999,999$)
 Range: real 0.000 through 60.999, *missing*

tc(l)

Description: convenience function to make typing dates and times in expressions easier
 Same as `tc()`, except returns leap second-adjusted values; for example, typing `tc(29nov2007 9:15)` is equivalent to typing `1511946900000`, whereas `tc(29nov2007 9:15)` is `1511946923000`.
 Domain l : datetime literal strings 01jan0100 00:00:00.000 to 31dec9999 23:59:59.999
 Range: datetimes 01jan0100 00:00:00.000 to 31dec9999 23:59:59.999
 (integers $-58,695,840,000,000$ to $> 253,717,919,999,999$)

tc(l)

Description: convenience function to make typing dates and times in expressions easier
 For example, typing `tc(2jan1960 13:42)` is equivalent to typing `135720000`; the date but not the time may be omitted, and then 01jan1960 is assumed; the seconds portion of the time may be omitted and is assumed to be 0.000; `tc(11:02)` is equivalent to typing `39720000`.
 Domain l : datetime literal strings 01jan0100 00:00:00.000 to 31dec9999 23:59:59.999
 Range: datetimes 01jan0100 00:00:00.000 to 31dec9999 23:59:59.999
 (integers $-58,695,840,000,000$ to $253,717,919,999,999$)

td(l)

Description: convenience function to make typing dates in expressions easier
 For example, typing `td(2jan1960)` is equivalent to typing `1`.
 Domain l : date literal strings 01jan0100 to 31dec9999
 Range: %td dates 01jan0100 to 31dec9999 (integers $-679,350$ to $2,936,549$)

th(l)

Description: convenience function to make typing half-yearly dates in expressions easier
 For example, typing `th(1960h2)` is equivalent to typing `1`.
 Domain l : half-year literal strings 0100h1 to 9999h2
 Range: %th dates 0100h1 to 9999h2 (integers $-3,720$ to $16,079$)

tm(l)

Description: convenience function to make typing monthly dates in expressions easier
 For example, typing `tm(1960m2)` is equivalent to typing `1`.
 Domain l : month literal strings 0100m1 to 9999m12
 Range: %tm dates 0100m1 to 9999m12 (integers $-22,320$ to $96,479$)

tq(*l*)

Description: convenience function to make typing quarterly dates in expressions easier

For example, typing `tq(1960q2)` is equivalent to typing `1.`

Domain *l*: quarter literal strings 0100q1 to 9999q4

Range: %tq dates 0100q1 to 9999q4 (integers −7,440 to 32,159)

tw(*l*)

Description: convenience function to make typing weekly dates in expressions easier

For example, typing `tw(1960w2)` is equivalent to typing `1.`

Domain *l*: week literal strings 0100w1 to 9999w52

Range: %tw dates 0100w1 to 9999w52 (integers −96,720 to 418,079)

week(*e_d*)

Description: the numeric week of the year corresponding to date *e_d*, the %td encoded date (days since 01jan1960)

Note: The first week of a year is the first 7-day period of the year.

Domain *e_d*: %td dates 01jan0100 to 31dec9999 (integers −679,350 to 2,936,549)

Range: integers 1 to 52 or *missing*

weekly(*s₁*, *s₂* [, *Y*])

Description: the *e_w* weekly date (weeks since 1960w1) corresponding to *s₁* based on *s₂* and *Y*; *Y* specifies *topyear*; see [date\(\)](#)

Domain *s₁*: strings

Domain *s₂*: strings "WY" and "YW"; *Y* may be prefixed with ##

Domain *Y*: integers 1000 to 9998 (but probably 2001 to 2099)

Range: %tw dates 0100w1 to 9999w52 (integers −96,720 to 418,079) or *missing*

wofd(*e_d*)

Description: the *e_w* weekly date (weeks since 1960w1) containing date *e_d*

Domain *e_d*: %td dates 01jan0100 to 31dec9999 (integers −679,350 to 2,936,549)

Range: %tw dates 0100w1 to 9999w52 (integers −96,720 to 418,079)

year(*e_d*)

Description: the numeric year corresponding to date *e_d*

Domain *e_d*: %td dates 01jan0100 to 31dec9999 (integers −679,350 to 2,936,549)

Range: integers 0100 to 9999 (but probably 1800 to 2100)

yearly(*s₁*, *s₂* [, *Y*])

Description: the *e_y* yearly date (year) corresponding to *s₁* based on *s₂* and *Y*; *Y* specifies *topyear*; see [date\(\)](#)

Domain *s₁*: strings

Domain *s₂*: string "Y"; *Y* may be prefixed with ##

Domain *Y*: integers 1000 to 9998 (but probably 2001 to 2099)

Range: %ty dates 0100 to 9999 (integers 0100 to 9999) or *missing*

yh(*Y*, *H*)

Description: the *e_h* half-yearly date (half-years since 1960h1) corresponding to year *Y*, half-year *H*

Domain *Y*: integers 1000 to 9999 (but probably 1800 to 2100)

Domain *H*: integers 1, 2

Range: %th dates 1000h1 to 9999h2 (integers −1,920 to 16,079)

ym(Y, M)

Description: the e_m monthly date (months since 1960m1) corresponding to year Y , month M
 Domain Y : integers 1000 to 9999 (but probably 1800 to 2100)
 Domain M : integers 1 to 12
 Range: %tm dates 1000m1 to 9999m12 (integers $-11,520$ to $96,479$)

yofd(e_d)

Description: the e_y yearly date (year) containing date e_d
 Domain e_d : %td dates 01jan0100 to 31dec9999 (integers $-679,350$ to $2,936,549$)
 Range: %ty dates 0100 to 9999 (integers 0100 to 9999)

yq(Y, Q)

Description: the e_q quarterly date (quarters since 1960q1) corresponding to year Y , quarter Q
 Domain Y : integers 1000 to 9999 (but probably 1800 to 2100)
 Domain Q : integers 1 to 4
 Range: %tq dates 1000q1 to 9999q4 (integers $-3,840$ to $32,159$)

yw(Y, W)

Description: the e_w weekly date (weeks since 1960w1) corresponding to year Y , week W
 Domain Y : integers 1000 to 9999 (but probably 1800 to 2100)
 Domain W : integers 1 to 52
 Range: %tw dates 1000w1 to 9999w52 (integers $-49,920$ to $418,079$)

Video example

[How to create a date variable from a date stored as a string](#)

References

- Cox, N. J. 2018. [Stata tip 130: 106610 and all that: Date variables that need to be fixed](#). *Stata Journal* 18: 755–757.
- Rajbhandari, A. 2015. A tour of datetime in Stata. *The Stata Blog: Not Elsewhere Classified*.
<http://blog.stata.com/2015/12/17/a-tour-of-datetime-in-stata-i/>.

Also see

- [FN] **Functions by category**
- [D] **Datetime** — Date and time values and variables
- [D] **egen** — Extensions to generate
- [D] **generate** — Create or change contents of variable
- [M-5] **date()** — Date and time manipulation
- [U] **13.3 Functions**

Contents

<code>abs(<i>x</i>)</code>	the absolute value of <i>x</i>
<code>ceil(<i>x</i>)</code>	the unique integer <i>n</i> such that $n - 1 < x \leq n$; <i>x</i> (not “.”) if <i>x</i> is missing, meaning that <code>ceil(.a) = .a</code>
<code>cloglog(<i>x</i>)</code>	the complementary log-log of <i>x</i>
<code>comb(<i>n</i>,<i>k</i>)</code>	the combinatorial function $n!/\{k!(n-k)!\}$
<code>digamma(<i>x</i>)</code>	the <code>digamma()</code> function, $d \ln \Gamma(x)/dx$
<code>exp(<i>x</i>)</code>	the exponential function e^x
<code>expm1(<i>x</i>)</code>	$e^x - 1$ with higher precision than <code>exp(<i>x</i>) - 1</code> for small values of $ x $
<code>floor(<i>x</i>)</code>	the unique integer <i>n</i> such that $n \leq x < n + 1$; <i>x</i> (not “.”) if <i>x</i> is missing, meaning that <code>floor(.a) = .a</code>
<code>int(<i>x</i>)</code>	the integer obtained by truncating <i>x</i> toward 0 (thus, <code>int(5.2) = 5</code> and <code>int(-5.8) = -5</code>); <i>x</i> (not “.”) if <i>x</i> is missing, meaning that <code>int(.a) = .a</code>
<code>invcloglog(<i>x</i>)</code>	the inverse of the complementary log-log function of <i>x</i>
<code>invlogit(<i>x</i>)</code>	the inverse of the logit function of <i>x</i>
<code>ln(<i>x</i>)</code>	the natural logarithm, $\ln(x)$
<code>ln1m(<i>x</i>)</code>	the natural logarithm of $1 - x$ with higher precision than <code>ln(1 - <i>x</i>)</code> for small values of $ x $
<code>ln1p(<i>x</i>)</code>	the natural logarithm of $1 + x$ with higher precision than <code>ln(1 + <i>x</i>)</code> for small values of $ x $
<code>lnfactorial(<i>n</i>)</code>	the natural log of <i>n</i> factorial = $\ln(n!)$
<code>lngamma(<i>x</i>)</code>	$\ln\{\Gamma(x)\}$
<code>log(<i>x</i>)</code>	a synonym for <code>ln(<i>x</i>)</code>
<code>log10(<i>x</i>)</code>	the base-10 logarithm of <i>x</i>
<code>log1m(<i>x</i>)</code>	a synonym for <code>ln1m(<i>x</i>)</code>
<code>log1p(<i>x</i>)</code>	a synonym for <code>ln1p(<i>x</i>)</code>
<code>logit(<i>x</i>)</code>	the log of the odds ratio of <i>x</i> , $\text{logit}(x) = \ln\{x/(1-x)\}$
<code>max(<i>x</i>₁,<i>x</i>₂,...,<i>x</i>_{<i>n</i>})</code>	the maximum value of <i>x</i> ₁ , <i>x</i> ₂ ,..., <i>x</i> _{<i>n</i>}
<code>min(<i>x</i>₁,<i>x</i>₂,...,<i>x</i>_{<i>n</i>})</code>	the minimum value of <i>x</i> ₁ , <i>x</i> ₂ ,..., <i>x</i> _{<i>n</i>}
<code>mod(<i>x</i>,<i>y</i>)</code>	the modulus of <i>x</i> with respect to <i>y</i>
<code>reldif(<i>x</i>,<i>y</i>)</code>	the “relative” difference $ x - y /(y + 1)$; 0 if both arguments are the same type of extended missing value; <i>missing</i> if only one argument is missing or if the two arguments are two different types of <i>missing</i>

<code>round(x,y)</code> or <code>round(x)</code>	x rounded in units of y or x rounded to the nearest integer if the argument y is omitted; x (not “.”) if x is missing (meaning that <code>round(.a) = .a</code> and that <code>round(.a,y) = .a</code> if y is not missing) and if y is missing, then “.” is returned
<code>sign(x)</code>	the sign of x : -1 if $x < 0$, 0 if $x = 0$, 1 if $x > 0$, or <i>missing</i> if x is missing
<code>sqrt(x)</code>	the square root of x
<code>sum(x)</code>	the running sum of x , treating missing values as zero
<code>trigamma(x)</code>	the second derivative of <code>lngamma(x)</code> $= d^2 \ln \Gamma(x) / dx^2$
<code>trunc(x)</code>	a synonym for <code>int(x)</code>

Functions

`abs(x)`

Description: the absolute value of x
Domain: $-8\text{e}+307$ to $8\text{e}+307$
Range: 0 to $8\text{e}+307$

`ceil(x)`

Description: the unique integer n such that $n - 1 < x \leq n$; x (not “.”) if x is missing, meaning that `ceil(.a) = .a`
Also see `floor(x)`, `int(x)`, and `round(x)`.
Domain: $-8\text{e}+307$ to $8\text{e}+307$
Range: integers in $-8\text{e}+307$ to $8\text{e}+307$

`cloglog(x)`

Description: the complementary log-log of x
 $\text{cloglog}(x) = \ln\{-\ln(1-x)\}$
Domain: 0 to 1
Range: $-8\text{e}+307$ to $8\text{e}+307$

`comb(n , k)`

Description: the combinatorial function $n! / \{k!(n-k)!\}$
Domain n : integers 1 to $1\text{e}+305$
Domain k : integers 0 to n
Range: 0 to $8\text{e}+307$ or *missing*

`digamma(x)`

Description: the `digamma()` function, $d \ln \Gamma(x) / dx$
This is the derivative of `lngamma(x)`. The `digamma(x)` function is sometimes called the psi function, $\psi(x)$.
Domain: $-1\text{e}+15$ to $8\text{e}+307$
Range: $-8\text{e}+307$ to $8\text{e}+307$ or *missing*

`exp(x)`

Description: the exponential function e^x
This function is the inverse of `ln(x)`. To compute $e^x - 1$ with high precision for small values of $|x|$, use `expm1(x)`.
Domain: $-8\text{e}+307$ to 709
Range: 0 to $8\text{e}+307$

expm1(x)

Description: $e^x - 1$ with higher precision than `exp(x) - 1` for small values of $|x|$

Domain: $-8\text{e}+307$ to 709

Range: -1 to $8\text{e}+307$

floor(x)

Description: the unique integer n such that $n \leq x < n + 1$; x (not “.”) if x is missing, meaning that `floor(.a) = .a`

Also see `ceil(x)`, `int(x)`, and `round(x)`.

Domain: $-8\text{e}+307$ to $8\text{e}+307$

Range: integers in $-8\text{e}+307$ to $8\text{e}+307$

int(x)

Description: the integer obtained by truncating x toward 0 (thus, `int(5.2) = 5` and `int(-5.8) = -5`); x (not “.”) if x is missing, meaning that `int(.a) = .a`

One way to obtain the closest integer to x is `int(x +sign(x)/2)`, which simplifies to `int(x +0.5)` for $x \geq 0$. However, use of the `round()` function is preferred. Also see `round(x)`, `ceil(x)`, and `floor(x)`.

Domain: $-8\text{e}+307$ to $8\text{e}+307$

Range: integers in $-8\text{e}+307$ to $8\text{e}+307$

invcloglog(x)

Description: the inverse of the complementary log-log function of x

$$\text{invcloglog}(x) = 1 - \exp\{-\exp(x)\}$$

Domain: $-8\text{e}+307$ to $8\text{e}+307$

Range: 0 to 1 or *missing*

invlogit(x)

Description: the inverse of the logit function of x

$$\text{invlogit}(x) = \exp(x) / \{1 + \exp(x)\}$$

Domain: $-8\text{e}+307$ to $8\text{e}+307$

Range: 0 to 1 or *missing*

ln(x)

Description: the natural logarithm, $\ln(x)$

This function is the inverse of `exp(x)`. The logarithm of x in base b can be calculated via $\log_b(x) = \log_a(x) / \log_a(b)$. Hence,

$$\log_5(x) = \ln(x) / \ln(5) = \log(x) / \log(5) = \log_{10}(x) / \log_{10}(5)$$

$$\log_2(x) = \ln(x) / \ln(2) = \log(x) / \log(2) = \log_{10}(x) / \log_{10}(2)$$

You can calculate $\log_b(x)$ by using the formula that best suits your needs. To compute $\ln(1 - x)$ and $\ln(1 + x)$ with high precision for small values of $|x|$, use `ln1m(x)` and `ln1p(x)`, respectively.

Domain: $1\text{e}-323$ to $8\text{e}+307$

Range: -744 to 709

ln1m(*x*)

Description: the natural logarithm of $1 - x$ with higher precision than `ln(1 - x)` for small values of $|x|$

Domain: $-8\text{e}+307$ to $1 - \text{c}(\text{epsdoubler})$

Range: -37 to 709

ln1p(*x*)

Description: the natural logarithm of $1 + x$ with higher precision than `ln(1 + x)` for small values of $|x|$

Domain: $-1 + \text{c}(\text{epsdoubler})$ to $8\text{e}+307$

Range: -37 to 709

lnfactorial(*n*)

Description: the natural log of n factorial = $\ln(n!)$

To calculate $n!$, use `round(exp(lnfactorial(n)), 1)` to ensure that the result is an integer. Logs of factorials are generally more useful than the factorials themselves because of overflow problems.

Domain: integers 0 to $1\text{e}+305$

Range: 0 to $8\text{e}+307$

lngamma(*x*)

Description: $\ln\{\Gamma(x)\}$

Here the gamma function, $\Gamma(x)$, is defined by $\Gamma(x) = \int_0^\infty t^{x-1}e^{-t}dt$. For integer values of $x > 0$, this is $\ln((x - 1)!)$.

`lngamma(x)` for $x < 0$ returns a number such that `exp(lngamma(x))` is equal to the absolute value of the gamma function, $\Gamma(x)$. That is, `lngamma(x)` always returns a real (not complex) result.

Domain: $-2,147,483,648$ to $1\text{e}+305$ (excluding negative integers)

Range: $-8\text{e}+307$ to $8\text{e}+307$

log(*x*)

Description: a synonym for `ln(x)`

log10(*x*)

Description: the base-10 logarithm of x

Domain: $1\text{e}-323$ to $8\text{e}+307$

Range: -323 to 308

log1m(*x*)

Description: a synonym for `ln1m(x)`

log1p(*x*)

Description: a synonym for `ln1p(x)`

logit(*x*)

Description: the log of the odds ratio of x , $\text{logit}(x) = \ln\{x/(1 - x)\}$

Domain: 0 to 1 (exclusive)

Range: $-8\text{e}+307$ to $8\text{e}+307$ or *missing*

max(x_1, x_2, \dots, x_n)Description: the maximum value of x_1, x_2, \dots, x_n Unless all arguments are *missing*, missing values are ignored. $\text{max}(2, 10, ., 7) = 10$ $\text{max}(., ., .) = .$ Domain x_1 : $-8\text{e}+307$ to $8\text{e}+307$ or *missing*Domain x_2 : $-8\text{e}+307$ to $8\text{e}+307$ or *missing*

...

Domain x_n : $-8\text{e}+307$ to $8\text{e}+307$ or *missing*Range: $-8\text{e}+307$ to $8\text{e}+307$ or *missing***min**(x_1, x_2, \dots, x_n)Description: the minimum value of x_1, x_2, \dots, x_n Unless all arguments are *missing*, missing values are ignored. $\text{min}(2, 10, ., 7) = 2$ $\text{min}(., ., .) = .$ Domain x_1 : $-8\text{e}+307$ to $8\text{e}+307$ or *missing*Domain x_2 : $-8\text{e}+307$ to $8\text{e}+307$ or *missing*

...

Domain x_n : $-8\text{e}+307$ to $8\text{e}+307$ or *missing*Range: $-8\text{e}+307$ to $8\text{e}+307$ or *missing***mod**(x, y)Description: the modulus of x with respect to y $\text{mod}(x, y) = x - y \text{ floor}(x/y)$ $\text{mod}(x, 0) = .$ Domain x : $-8\text{e}+307$ to $8\text{e}+307$ Domain y : 0 to $8\text{e}+307$ Range: 0 to $8\text{e}+307$ **reldif**(x, y)Description: the “relative” difference $|x - y|/(|y| + 1)$; 0 if both arguments are the same type of extended missing value; *missing* if only one argument is missing or if the two arguments are two different types of *missing*Domain x : $-8\text{e}+307$ to $8\text{e}+307$ or *missing*Domain y : $-8\text{e}+307$ to $8\text{e}+307$ or *missing*Range: $-8\text{e}+307$ to $8\text{e}+307$ or *missing*

round(*x*,*y*) or **round(*x*)**

Description: *x* rounded in units of *y* or *x* rounded to the nearest integer if the argument *y* is omitted; *x* (not “.”) if *x* is missing (meaning that `round(.a) = .a` and that `round(.a,y) = .a` if *y* is not missing) and if *y* is missing, then “.” is returned

For *y* = 1, or with *y* omitted, this amounts to the closest integer to *x*; `round(5.2,1)` is 5, as is `round(4.8,1)`; `round(-5.2,1)` is -5, as is `round(-4.8,1)`. The rounding definition is generalized for *y* ≠ 1. With *y* = 0.01, for instance, *x* is rounded to two decimal places; `round(sqrt(2),.01)` is 1.41. *y* may also be larger than 1; `round(28,5)` is 30, which is 28 rounded to the closest multiple of 5. For *y* = 0, the function is defined as returning *x* unmodified. Also see `int(x)`, `ceil(x)`, and `floor(x)`.

Domain *x*: -8e+307 to 8e+307

Domain *y*: -8e+307 to 8e+307

Range: -8e+307 to 8e+307

sign(*x*)

Description: the sign of *x*: -1 if *x* < 0, 0 if *x* = 0, 1 if *x* > 0, or *missing* if *x* is missing

Domain: -8e+307 to 8e+307 or *missing*

Range: -1, 0, 1 or *missing*

sqrt(*x*)

Description: the square root of *x*

Domain: 0 to 8e+307

Range: 0 to 1e+154

sum(*x*)

Description: the running sum of *x*, treating missing values as zero

For example, following the command `generate y=sum(x)`, the *j*th observation on *y* contains the sum of the first through *j*th observations on *x*. See [D] [egen](#) for an alternative sum function, `total()`, that produces a constant equal to the overall sum.

Domain: all real numbers or *missing*

Range: -8e+307 to 8e+307 (excluding *missing*)

trigamma(*x*)

Description: the second derivative of $\text{lngamma}(x) = d^2 \ln \Gamma(x) / dx^2$

The `trigamma()` function is the derivative of `digamma(x)`.

Domain: -1e+15 to 8e+307

Range: 0 to 8e+307 or *missing*

trunc(*x*)

Description: a synonym for `int(x)`

Video example

[How to round a continuous variable](#)

References

- Abramowitz, M., and I. A. Stegun, ed. 1964. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Washington, DC: National Bureau of Standards.
- Cox, N. J. 2003. [Stata tip 2: Building with floors and ceilings](#). *Stata Journal* 3: 446–447.
- . 2007. [Stata tip 43: Remainders, selections, sequences, extractions: Uses of the modulus](#). *Stata Journal* 7: 143–145.
- . 2018. [Speaking Stata: From rounding to binning](#). *Stata Journal* 18: 741–754.
- Oldham, K. B., J. C. Myland, and J. Spanier. 2009. *An Atlas of Functions*. 2nd ed. New York: Springer.

Also see

- [FN] [Functions by category](#)
- [D] [egen](#) — Extensions to generate
- [D] [generate](#) — Create or change contents of variable
- [M-4] [Intro](#) — Categorical guide to Mata functions
- [U] [13.3 Functions](#)

Contents

<code>cholesky(<i>M</i>)</code>	the Cholesky decomposition of the matrix: if $R = \text{cholesky}(S)$, then $RR^T = S$
<code>coleqnumb(<i>M</i>,<i>s</i>)</code>	the equation number of M associated with column equation s ; <i>missing</i> if the column equation cannot be found
<code>colnfreeparms(<i>M</i>)</code>	the number of free parameters in columns of M
<code>colnumb(<i>M</i>,<i>s</i>)</code>	the column number of M associated with column name s ; <i>missing</i> if the column cannot be found
<code>colsof(<i>M</i>)</code>	the number of columns of M
<code>corr(<i>M</i>)</code>	the correlation matrix of the variance matrix
<code>det(<i>M</i>)</code>	the determinant of matrix M
<code>diag(<i>M</i>)</code>	the square, diagonal matrix created from the row or column vector
<code>diag0cnt(<i>M</i>)</code>	the number of zeros on the diagonal of M
<code>el(<i>s</i>,<i>i</i>,<i>j</i>)</code>	$s[\text{floor}(i), \text{floor}(j)]$, the i, j element of the matrix named s ; <i>missing</i> if i or j are out of range or if matrix s does not exist
<code>get(<i>systemname</i>)</code>	a copy of Stata internal system matrix <i>systemname</i>
<code>hadamard(<i>M</i>,<i>N</i>)</code>	a matrix whose i, j element is $M[i, j] \cdot N[i, j]$ (if M and N are not the same size, this function reports a conformability error)
<code>I(<i>n</i>)</code>	an $n \times n$ identity matrix if n is an integer; otherwise, a $\text{round}(n) \times \text{round}(n)$ identity matrix
<code>inv(<i>M</i>)</code>	the inverse of the matrix M
<code>invsym(<i>M</i>)</code>	the inverse of M if M is positive definite
<code>issymmetric(<i>M</i>)</code>	1 if the matrix is symmetric; otherwise, 0
<code>J(<i>r</i>,<i>c</i>,<i>z</i>)</code>	the $r \times c$ matrix containing elements z
<code>matmissing(<i>M</i>)</code>	1 if any elements of the matrix are missing; otherwise, 0
<code>matuniform(<i>r</i>,<i>c</i>)</code>	the $r \times c$ matrices containing uniformly distributed pseudorandom numbers on the interval (0, 1)
<code>mreldif(<i>X</i>,<i>Y</i>)</code>	the relative difference of X and Y , where the relative difference is defined as $\max_{i,j} \{ x_{ij} - y_{ij} / (y_{ij} + 1)\}$
<code>nullmat(<i>matname</i>)</code>	use with the row-join (,) and column-join (\) operators
<code>roweqnumb(<i>M</i>,<i>s</i>)</code>	the equation number of M associated with row equation s ; <i>missing</i> if the row equation cannot be found
<code>rownfreeparms(<i>M</i>)</code>	the number of free parameters in rows of M
<code>rownumb(<i>M</i>,<i>s</i>)</code>	the row number of M associated with row name s ; <i>missing</i> if the row cannot be found
<code>rowsof(<i>M</i>)</code>	the number of rows of M
<code>sweep(<i>M</i>,<i>i</i>)</code>	matrix M with i th row/column swept
<code>trace(<i>M</i>)</code>	the trace of matrix M

<code>vec(M)</code>	a column vector formed by listing the elements of M , starting with the first column and proceeding column by column
<code>vecdiag(M)</code>	the row vector containing the diagonal of matrix M

Functions

We divide the basic matrix functions into two groups, according to whether they return a matrix or a scalar:

Matrix functions returning a matrix
Matrix functions returning a scalar

Matrix functions returning a matrix

In addition to the functions listed below, see [P] **matrix svd** for singular value decomposition, [P] **matrix symeigen** for eigenvalues and eigenvectors of symmetric matrices, and [P] **matrix eigenvalues** for eigenvalues of nonsymmetric matrices.

`cholesky(M)`

Description: the Cholesky decomposition of the matrix: if $R = \text{cholesky}(S)$, then $RR^T = S$. R^T indicates the transpose of R . Row and column names are obtained from M .

Domain: $n \times n$, positive-definite, symmetric matrices

Range: $n \times n$ lower-triangular matrices

`corr(M)`

Description: the correlation matrix of the variance matrix

Row and column names are obtained from M .

Domain: $n \times n$ symmetric variance matrices

Range: $n \times n$ symmetric correlation matrices

`diag(M)`

Description: the square, diagonal matrix created from the row or column vector

Row and column names are obtained from the column names of M if M is a row vector or from the row names of M if M is a column vector.

Domain: $1 \times n$ and $n \times 1$ vectors

Range: $n \times n$ diagonal matrices

`get(systemname)`

Description: a copy of Stata internal system matrix *systemname*

This function is included for backward compatibility with previous versions of Stata.

Domain: existing names of system matrices

Range: matrices

hadamard(M, N)

Description: a matrix whose i, j element is $M[i, j] \cdot N[i, j]$ (if M and N are not the same size, this function reports a conformability error)

Domain M : $m \times n$ matrices

Domain N : $m \times n$ matrices

Range: $m \times n$ matrices

I(n)

Description: an $n \times n$ identity matrix if n is an integer; otherwise, a `round(n)` \times `round(n)` identity matrix

Domain: real scalars 1 to `c(max_matdim)`

Range: identity matrices

inv(M)

Description: the inverse of the matrix M

If M is singular, this will result in an error.

The function `invsym()` should be used in preference to `inv()` because `invsym()` is more accurate. The row names of the result are obtained from the column names of M , and the column names of the result are obtained from the row names of M .

Domain: $n \times n$ nonsingular matrices

Range: $n \times n$ matrices

invsym(M)

Description: the inverse of M if M is positive definite

If M is not positive definite, rows will be inverted until the diagonal terms are zero or negative; the rows and columns corresponding to these terms will be set to 0, producing a g2 inverse. The row names of the result are obtained from the column names of M , and the column names of the result are obtained from the row names of M .

Domain: $n \times n$ symmetric matrices

Range: $n \times n$ symmetric matrices

J(r, c, z)

Description: the $r \times c$ matrix containing elements z

Domain r : integer scalars 1 to `c(max_matdim)`

Domain c : integer scalars 1 to `c(max_matdim)`

Domain z : scalars $-8\text{e}+307$ to $8\text{e}+307$

Range: $r \times c$ matrices

matuniform(r, c)

Description: the $r \times c$ matrices containing uniformly distributed pseudorandom numbers on the interval (0, 1)

Domain r : integer scalars 1 to `c(max_matdim)`

Domain c : integer scalars 1 to `c(max_matdim)`

Range: $r \times c$ matrices

nullmat(matname)

Description: use with the row-join (,) and column-join (\) operators

Consider the following code fragment, which is an attempt to create the vector (1, 2, 3, 4):

```
forvalues i = 1/4 {
    mat v = (v, 'i')
}
```

The above program will not work because, the first time through the loop, *v* will not yet exist, and thus forming (*v*, 'i') makes no sense. **nullmat()** relaxes that restriction:

```
forvalues i = 1/4 {
    mat v = (nullmat(v), 'i')
}
```

The **nullmat()** function informs Stata that if *v* does not exist, the function row-join is to be generalized. Joining nothing with 'i' results in ('i'). Thus the first time through the loop, *v* = (1) is formed. The second time through, *v* does exist, so *v* = (1, 2) is formed, and so on.

nullmat() can be used only with the , and \ operators.

Domain: matrix names, existing and nonexisting

Range: matrices including null if *matname* does not exist

sweep(*M*, *i*)

Description: matrix *M* with *i*th row/column swept

The row and column names of the resultant matrix are obtained from *M*, except that the *n*th row and column names are interchanged. If $B = \text{sweep}(A, k)$, then

$$\begin{aligned} B_{kk} &= \frac{1}{A_{kk}} \\ B_{ik} &= -\frac{A_{ik}}{A_{kk}}, & i \neq k \\ B_{kj} &= \frac{A_{kj}}{A_{kk}}, & j \neq k \\ B_{ij} &= A_{ij} - \frac{A_{ik}A_{kj}}{A_{kk}}, & i \neq k, j \neq k \end{aligned}$$

Domain *M*: $n \times n$ matrices

Domain *i*: integer scalars 1 to *n*

Range: $n \times n$ matrices

vec(*M*)

Description: a column vector formed by listing the elements of *M*, starting with the first column and proceeding column by column

Domain: matrices

Range: column vectors ($n \times 1$ matrices)

vecdiag(M)

Description: the row vector containing the diagonal of matrix M

`vecdiag()` is the opposite of `diag()`. The row name is set to `r1`; the column names are obtained from the column names of M .

Domain: $n \times n$ matrices

Range: $1 \times n$ vectors

Matrix functions returning a scalar**coleqnumb(M, s)**

Description: the equation number of M associated with column equation s ; *missing* if the column equation cannot be found

Domain M : matrices

Domain s : strings

Range: integer scalars 1 to `c(max_matdim)` or *missing*

colnfreeparms(M)

Description: the number of free parameters in columns of M

Domain: matrices

Range: integer scalars 0 to `c(max_matdim)`

colnumb(M, s)

Description: the column number of M associated with column name s ; *missing* if the column cannot be found

Domain M : matrices

Domain s : strings

Range: integer scalars 1 to `c(max_matdim)` or *missing*

colsof(M)

Description: the number of columns of M

Domain: matrices

Range: integer scalars 1 to `c(max_matdim)`

det(M)

Description: the determinant of matrix M

Domain: $n \times n$ (square) matrices

Range: scalars $-8\text{e}+307$ to $8\text{e}+307$

diag0cnt(M)

Description: the number of zeros on the diagonal of M

Domain: $n \times n$ (square) matrices

Range: integer scalars 0 to n

el(s, i, j)

Description: $s[\text{floor}(i), \text{floor}(j)]$, the i, j element of the matrix named s ; *missing* if i or j are out of range or if matrix s does not exist

Domain s : strings containing matrix name

Domain i : scalars 1 to `c(max_matdim)`

Domain j : scalars 1 to `c(max_matdim)`

Range: scalars $-8\text{e}+307$ to $8\text{e}+307$ or *missing*

issymmetric(M)

Description: 1 if the matrix is symmetric; otherwise, 0

Domain M : matrices

Range: integers 0 and 1

matmissing(M)

Description: 1 if any elements of the matrix are missing; otherwise, 0

Domain M : matrices

Range: integers 0 and 1

mreldif(X, Y)

Description: the relative difference of X and Y , where the relative difference is defined as $\max_{i,j} \{|x_{ij} - y_{ij}| / (|y_{ij}| + 1)\}$

Domain X : matrices

Domain Y : matrices with same number of rows and columns as X

Range: scalars $-8\text{e}+307$ to $8\text{e}+307$

roweqnumb(M, s)

Description: the equation number of M associated with row equation s ; *missing* if the row equation cannot be found

Domain M : matrices

Domain s : strings

Range: integer scalars 1 to $c(\text{max_matdim})$ or *missing*

rownfreeparms(M)

Description: the number of free parameters in rows of M

Domain: matrices

Range: integer scalars 0 to $c(\text{max_matdim})$

rownumb(M, s)

Description: the row number of M associated with row name s ; *missing* if the row cannot be found

Domain M : matrices

Domain s : strings

Range: integer scalars 1 to $c(\text{max_matdim})$ or *missing*

rowsof(M)

Description: the number of rows of M

Domain: matrices

Range: integer scalars 1 to $c(\text{max_matdim})$

trace(M)

Description: the trace of matrix M

Domain: $n \times n$ (square) matrices

Range: scalars $-8\text{e}+307$ to $8\text{e}+307$

Jacques Salomon Hadamard (1865–1963) was born in Versailles, France. He had a tumultuous childhood, eating elephant meat to survive and enduring the premature deaths of two younger sisters. Hadamard taught while working on his doctorate, which he obtained in 1892 from École Normale Supérieure. His dissertation is recognized as the first examination of singularities. Hadamard published a paper on the Riemann zeta function, for which he was awarded the Grand Prix des Sciences Mathématiques in 1892. Shortly after, he became a professor at the University of Bordeaux and made many significant contributions over the course of four years. For example, in 1893 he published a paper on determinant inequalities, giving rise to Hadamard matrices. Then in 1896, he used complex analysis to prove the prime number theorem, and he was awarded the Bordin Prize by the Academy of Sciences for his work on dynamic trajectories. In the following years, he published books on two-dimensional and three-dimensional geometry, as well as an influential paper on functional analysis. He was elected to presidency of the French Mathematical Society in 1906 and as chair of mechanics at the Collège de France in 1909. Faced with the tragic deaths of two of his sons during World War I, Hadamard buried himself in his work. He continued to publish outstanding work in new areas, including probability theory, education, and psychology. In 1956, he was awarded the CNRS Gold Medal for his many contributions.

Reference

Maz'ya, V. G., and T. O. Shaposhnikova. 1998. *Jacques Hadamard, A Universal mathematician*. Providence, RI: American Mathematical Society.

Also see

- [FN] [Functions by category](#)
- [D] [egen](#) — Extensions to generate
- [D] [generate](#) — Create or change contents of variable
- [M-4] [Intro](#) — Categorical guide to Mata functions
- [U] [13.3 Functions](#)
- [U] [14.8 Matrix functions](#)

Contents

<code>autocode(x, n, x_0, x_1)</code>	partitions the interval from x_0 to x_1 into n equal-length intervals and returns the upper bound of the interval that contains x
<code>byteorder()</code>	1 if your computer stores numbers by using a hilo byte order and evaluates to 2 if your computer stores numbers by using a lohi byte order
<code>c(name)</code>	the value of the system or constant result <code>c(name)</code> (see [P] creturn)
<code>_caller()</code>	version of the program or session that invoked the currently running program; see [P] version
<code>chop(x, ϵ)</code>	<code>round(x)</code> if <code>abs($x - \text{round}(x)$) < ϵ</code> ; otherwise, x ; or x if x is missing
<code>clip(x, a, b)</code>	x if $a < x < b$, b if $x \geq b$, a if $x \leq a$, or <i>missing</i> if x is missing or if $a > b$; x if x is missing
<code>cond($x, a, b[, c]$)</code>	a if x is <i>true</i> and nonmissing, b if x is <i>false</i> , and c if x is <i>missing</i> ; a if c is not specified and x evaluates to <i>missing</i>
<code>e(name)</code>	the value of stored result <code>e(name)</code> ; see [U] 18.8 Accessing results calculated by other programs
<code>e(sample)</code>	1 if the observation is in the estimation sample and 0 otherwise
<code>epsdouble()</code>	the machine precision of a double-precision number
<code>epsfloat()</code>	the machine precision of a floating-point number
<code>fileexists(f)</code>	1 if the file specified by f exists; otherwise, 0
<code>fileread(f)</code>	the contents of the file specified by f
<code>filereaderror(s)</code>	0 or positive integer, said value having the interpretation of a return code
<code>filewrite($f, s[, r]$)</code>	writes the string specified by s to the file specified by f and returns the number of bytes in the resulting file
<code>float(x)</code>	the value of x rounded to <code>float</code> precision
<code>fmtwidth($fntstr$)</code>	the output length of the <code>%fnt</code> contained in $fntstr$; <i>missing</i> if $fntstr$ does not contain a valid <code>%fnt</code>
<code>frval()</code>	returns values of variables stored in other frames
<code>_frval()</code>	programmer's version of <code>frval()</code>
<code>has_eprop(name)</code>	1 if $name$ appears as a word in <code>e(properties)</code> ; otherwise, 0
<code>inlist(z, a, b, \dots)</code>	1 if z is a member of the remaining arguments; otherwise, 0
<code>inrange(z, a, b)</code>	1 if it is known that $a \leq z \leq b$; otherwise, 0
<code>irecode(x, x_1, \dots, x_n)</code>	<i>missing</i> if x is missing or x_1, \dots, x_n is not weakly increasing; 0 if $x \leq x_1$; 1 if $x_1 < x \leq x_2$; 2 if $x_2 < x \leq x_3$; ...; n if $x > x_n$
<code>matrix(exp)</code>	restricts name interpretation to scalars and matrices; see scalar()

<code>maxbyte()</code>	the largest value that can be stored in storage type <code>byte</code>
<code>maxdouble()</code>	the largest value that can be stored in storage type <code>double</code>
<code>maxfloat()</code>	the largest value that can be stored in storage type <code>float</code>
<code>maxint()</code>	the largest value that can be stored in storage type <code>int</code>
<code>maxlong()</code>	the largest value that can be stored in storage type <code>long</code>
<code>mi(x_1, x_2, \dots, x_n)</code>	a synonym for <code>missing(x_1, x_2, \dots, x_n)</code>
<code>minbyte()</code>	the smallest value that can be stored in storage type <code>byte</code>
<code>mindouble()</code>	the smallest value that can be stored in storage type <code>double</code>
<code>minfloat()</code>	the smallest value that can be stored in storage type <code>float</code>
<code>minint()</code>	the smallest value that can be stored in storage type <code>int</code>
<code>minlong()</code>	the smallest value that can be stored in storage type <code>long</code>
<code>missing(x_1, x_2, \dots, x_n)</code>	1 if any x_i evaluates to <i>missing</i> ; otherwise, 0
<code>r(name)</code>	the value of the stored result <code>r(name)</code> ; see [U] 18.8 Accessing results calculated by other programs
<code>recode(x, x_1, \dots, x_n)</code>	<i>missing</i> if x_1, x_2, \dots, x_n is not weakly increasing; x if x is missing; x_1 if $x \leq x_1$; x_2 if $x \leq x_2, \dots$; otherwise, x_n if $x > x_1, x_2, \dots, x_{n-1}$. $x_i \geq .$ is interpreted as $x_i = +\infty$
<code>replay()</code>	1 if the first nonblank character of local macro '0' is a comma, or if '0' is empty
<code>return(name)</code>	the value of the to-be-stored result <code>r(name)</code> ; see [P] return
<code>s(name)</code>	the value of stored result <code>s(name)</code> ; see [U] 18.8 Accessing results calculated by other programs
<code>scalar(exp)</code>	restricts name interpretation to scalars and matrices
<code>smallestdouble()</code>	the smallest double-precision number greater than zero

Functions

`autocode(x, n, x_0, x_1)`

Description: partitions the interval from x_0 to x_1 into n equal-length intervals and returns the upper bound of the interval that contains x

This function is an automated version of `recode()`. See [U] 26 Working with categorical data and factor variables for an example.

The algorithm for `autocode()` is

```

if ( $n \geq . \mid x_0 \geq . \mid x_1 \geq . \mid n \leq 0 \mid x_0 \geq x_1$ )
  then return missing
if  $x \geq .$ , then return  $x$ 
otherwise
  for  $i = 1$  to  $n - 1$ 
     $xmap = x_0 + i * (x_1 - x_0) / n$ 
    if  $x \leq xmap$  then return  $xmap$ 
  end
otherwise
  return  $x_1$ 

```

Domain x : $-8e+307$ to $8e+307$

Domain n : integers 1 to $8e+307$

Domain x_0 : $-8e+307$ to $8e+307$

Domain x_1 : x_0 to $8e+307$

Range: x_0 to x_1

`byteorder()`

Description: 1 if your computer stores numbers by using a hilo byte order and evaluates to 2 if your computer stores numbers by using a lohi byte order

Consider the number 1 written as a 2-byte integer. On some computers (called hilo), it is written as “00 01”, and on other computers (called lohi), it is written as “01 00” (with the least significant byte written first). There are similar issues for 4-byte integers, 4-byte floats, and 8-byte floats. Stata automatically handles byte-order differences for Stata-created files. Users need not be concerned about this issue. Programmers producing custom binary files can use `byteorder()` to determine the native byte ordering; see [P] file.

Range: 1 and 2

`c(name)`

Description: the value of the system or constant result `c(name)` (see [P] [creturn](#))

Referencing `c(name)` will return an error if the result does not exist.

Domain: names

Range: real values, strings, or *missing*

`_caller()`

Description: version of the program or session that invoked the currently running program; see [P] [version](#)

The current version at the time of this writing is 16, so 16 is the upper end of this range. If Stata 16.1 were the current version, 16.1 would be the upper end of this range, and likewise, if Stata 17 were the current version, 17 would be the upper end of this range. This is a function for use by programmers.

Range: 1 to 16.0

`chop(x, ϵ)`

Description: $\text{round}(x)$ if $\text{abs}(x - \text{round}(x)) < \epsilon$; otherwise, x ; or x if x is missing

Domain x : $-8\text{e}+307$ to $8\text{e}+307$

Domain ϵ : $-8\text{e}+307$ to $8\text{e}+307$

Range: $-8\text{e}+307$ to $8\text{e}+307$

`clip(x, a, b)`

Description: x if $a < x < b$, b if $x \geq b$, a if $x \leq a$, or *missing* if x is missing or if $a > b$; x if x is missing

If a or b is missing, this is interpreted as $a = -\infty$ or $b = +\infty$, respectively.

Domain x : $-8\text{e}+307$ to $8\text{e}+307$

Domain a : $-8\text{e}+307$ to $8\text{e}+307$

Domain b : $-8\text{e}+307$ to $8\text{e}+307$

Range: $-8\text{e}+307$ to $8\text{e}+307$

`cond(x, a, b[, c])`

Description: a if x is *true* and nonmissing, b if x is *false*, and c if x is *missing*; a if c is not specified and x evaluates to *missing*

Note that expressions such as $x > 2$ will never evaluate to *missing*.

`cond(x>2,50,70)` returns 50 if $x > 2$ (includes $x \geq .$)

`cond(x>2,50,70)` returns 70 if $x \leq 2$

If you need a case for missing values in the above examples, try

`cond(missing(x), ., cond(x>2,50,70))` returns $.$ if x is *missing*,
returns 50 if $x > 2$, and returns 70 if $x \leq 2$

If the first argument is a scalar that may contain a missing value or a variable containing missing values, the fourth argument has an effect.

`cond(wage,1,0,.)` returns 1 if *wage* is not zero and not missing

`cond(wage,1,0,.)` returns 0 if *wage* is zero

`cond(wage,1,0,.)` returns $.$ if *wage* is *missing*

Caution: If the first argument to `cond()` is a logical expression, that is, `cond(x>2,50,70,.)`, the fourth argument is never reached.

Domain x : $-8\text{e}+307$ to $8\text{e}+307$ or *missing*; 0 \Rightarrow *false*, otherwise interpreted as *true*

Domain a : numbers and strings

Domain b : numbers if a is a number; strings if a is a string

Domain c : numbers if a is a number; strings if a is a string

Range: a , b , and c

e(name)

Description: the value of stored result `e(name)`; see [U] 18.8 Accessing results calculated by other programs

`e(name)` = scalar missing if the stored result does not exist

`e(name)` = specified matrix if the stored result is a matrix

`e(name)` = scalar numeric value if the stored result is a scalar

Domain: names

Range: strings, scalars, matrices, or *missing*

e(sample)

Description: 1 if the observation is in the estimation sample and 0 otherwise

Range: 0 and 1

epsdouble()

Description: the machine precision of a double-precision number

If $d < \text{epsdouble}()$ and (double) $x = 1$, then $x + d =$ (double) 1. This function takes no arguments, but the parentheses must be included.

Range: a double-precision number close to 0

epsfloat()

Description: the machine precision of a floating-point number

If $d < \text{epsfloat}()$ and (float) $x = 1$, then $x + d =$ (float) 1. This function takes no arguments, but the parentheses must be included.

Range: a floating-point number close to 0

fileexists(f)

Description: 1 if the file specified by f exists; otherwise, 0

If the file exists but is not readable, `fileexists()` will still return 1, because it does exist. If the “file” is a directory, `fileexists()` will return 0.

Domain: filenames

Range: 0 and 1

fileread(f)

Description: the contents of the file specified by f

If the file does not exist or an I/O error occurs while reading the file, then “`fileread() error #`” is returned, where $\#$ is a standard Stata error return code.

Domain: filenames

Range: strings

filereaderror(*s*)

Description: 0 or positive integer, said value having the interpretation of a return code

It is used like this

```
. generate strL s = fileread(filename) if fileexists(filename)
. assert filereaderror(s)==0
```

or this

```
. generate strL s = fileread(filename) if fileexists(filename)
. generate rc = filereaderror(s)
```

That is, `filereaderror(s)` is used on the result returned by `fileread(filename)` to determine whether an I/O error occurred.

In the example, we only `fileread()` files that `fileexists()`. That is not required. If the file does not exist, that will be detected by `filereaderror()` as an error. The way we showed the example, we did not want to read missing files as errors. If we wanted to treat missing files as errors, we would have coded

```
. generate strL s = fileread(filename)
. assert filereaderror(s)==0
```

or

```
. generate strL s = fileread(filename)
. generate rc = filereaderror(s)
```

Domain: strings

Range: integers

filewrite(*f*,*s*[,*r*])

Description: writes the string specified by *s* to the file specified by *f* and returns the number of bytes in the resulting file

If the optional argument *r* is specified as 1, the file specified by *f* will be replaced if it exists. If *r* is specified as 2, the file specified by *f* will be appended to if it exists. Any other values of *r* are treated as if *r* were not specified; that is, *f* will only be written to if it does not already exist.

When the file *f* is freshly created or is replaced, the value returned by `filewrite()` is the number of bytes written to the file, `strlen(s)`. If *r* is specified as 2, and thus `filewrite()` is appending to an existing file, the value returned is the total number of bytes in the resulting file; that is, the value is the sum of the number of the bytes in the file as it existed before `filewrite()` was called and the number of bytes newly written to it, `strlen(s)`.

If the file exists and *r* is not specified as 1 or 2, or an error occurs while writing to the file, then a negative number (#) is returned, where `abs(#)` is a standard Stata error return code.

Domain *f*: filenames

Domain *s*: strings

Domain *r*: integers 1 or 2

Range: integers

float(*x*)

Description: the value of *x* rounded to float precision

Although you may store your numeric variables as `byte`, `int`, `long`, `float`, or `double`, Stata converts all numbers to `double` before performing any calculations. Consequently, difficulties can arise in comparing numbers that have no finite binary representation.

For example, if the variable `x` is stored as a `float` and contains the value 1.1 (a repeating “decimal” in binary), the expression `x==1.1` will evaluate to `false` because the literal 1.1 is the `double` representation of 1.1, which is different from the `float` representation stored in `x`. (They differ by 2.384×10^{-8} .) The expression `x==float(1.1)` will evaluate to `true` because the `float()` function converts the literal 1.1 to its `float` representation before it is compared with `x`. (See [U] 13.12 Precision and problems therein for more information.)

Domain: $-1\text{e}+38$ to $1\text{e}+38$

Range: $-1\text{e}+38$ to $1\text{e}+38$

fmtwidth(*fmtstr*)

Description: the output length of the `%fmt` contained in *fmtstr*; *missing* if *fmtstr* does not contain a valid `%fmt`

For example, `fmtwidth("%9.2f")` returns 9 and `fmtwidth("%tc")` returns 18.

Range: strings

frval(*lvar*,*var*)

Description: returns values of variables stored in other frames

The frame functions `frval()` and `_frval()` access values of variables in frames outside the current frame. If you do not know what a frame is, see [D] frames intro.

The two functions do the same thing, but `frval()` is easier to use, and it is safer. `_frval()` is a programmer’s function.

lvar is the name of a variable created by `frlink` that links the current frame to another frame.

var is the name of a variable in the other frame.

Returned is the value of *var* from the observation in the other frame that matches the observation in the current frame.

Example 1: The current frame contains data on persons. Among the variables in the current frame is `countyid` containing the county in which each person lives.

Frame `frcounty` contains data on counties. In these data, variable `countyid` also records the county’s ID, and the other variables record county characteristics.

In the current frame, you have previously created variable `linkcnty` that links the current frame to `frcounty`. You did this by typing

```
. frlink m:1 countyid, frame(frcounty) generate(linkcnty)
```

Thus, you can now type

```
. generate rel_income = income / frval(linkcnty, median_income)
```


`income` is an existing variable in the current frame. `median_income` is an existing variable in `frcounty`. `rel_income` will be a new variable in the current frame, containing the income of each person divided by the median income of the county in which they live.

Example 2: It is usual to name frames after dataset names and to name link variables after frame names. Here is an example of this, following the names used above:

```
. use persons, clear
. frame create county
. frame county: use county
. frlink m:1 countyid, frame(county)
. generate rel_income = income / frval(county, median_income)
```

Domain *lvar*: the name of a variable created by `frlink` that links the current frame to another frame

Domain *var*: any variable (string or numeric) in the frame to which *lvar* links; varname abbreviation is allowed

Range: range of *var*, plus missing value (missing value is defined as `.` when *var* contains numeric data and `""` when *var* contains string data; missing value is returned for observations in the current frame that are unmatched in the other frame)

`frval(lvar, var, unnm)`

Description: the `frval()` function described above but with a third argument *unnm*

`frval()` returns the value of *var* from the observation in the frame linked using *lvar* that matches the observation in the current frame and the value in *unnm* if there is no matching observation.

For example, type

```
. generate median_inc = frval(county, median_income, .a)
```

to create new variable `median_inc` in the current frame, containing `median_income` from the other frame, or `.a` when there is no matched observation in the other frame.

Domain *lvar*: the name of a variable created by `frlink` that links the current frame to another frame

Domain *var*: any variable (string or numeric) in the frame to which *lvar* links; varname abbreviation is allowed

Domain *unnm*: any numeric value if *var* is numeric; any string value when *var* is string

Range: range of *var*, plus *unnm*

`_frval(frm, var, i)`

Description: programmer's version of `frval()`

It is useful for those wishing to write their own `frlink` and create special (or at least different) effects.

`_frval()` returns values of variables stored in other frames. It returns *var*'s *i*th observation (`var[i]`) from the frame *frm*; see [D] [frames intro](#).

If *i* is outside the valid range of observations for the frame, `_frval()` returns missing.

For example, you have two datasets in memory. The current frame is named `default` and contains 57 observations. The other dataset, we will assume, is stored in frame `xdata`. It contains different variables but on the same 57 observations. The two datasets are in the same order so that observation 1 in `default` corresponds to observation 1 in `xdata`, observation 2 to observation 2, and so on. You can type

```
. generate hrlywage = income / _frval(xdata, hrswrked, _n)
```

This will divide values of `income` stored in `default` by values of `hrswrked` stored in `xdata`.

The first thing to notice is that `_frval()`'s first two arguments are not expressions. You just type the name of the frame and the name of the variable without embedding them in quotes. We specified `xdata` for the frame name and `hrswrked` for the variable name.

The second thing to notice is that the third argument is an expression. To emphasize that, let's change the example. Assume that `xdata` contains 58 instead of 57 observations. Assume that observation 1 in `default` corresponds to observation 2 in `xdata`, observation 2 corresponds to observation 3, and so on. There is no observation in `default` that corresponds to observation 1 in `xdata`. In this case, you type

```
. generate hrlywage = income / _frval(xdata, hrswrked, _n+1)
```

These examples are artificial. You will normally use `_frval()` by creating a variable in `default` that contains the corresponding observation numbers in `xdata`. If the variable were called `xobsno`, then in the first example, `xobsno` would contain 1, 2, ..., 57.

In the second example, `xobsno` would contain 2, 3, ..., 58.

In another example, `xobsno` might contain 9, 6, ..., 32, which is to say, the numbers 2, 3, ..., 58, but permuted to reflect the datasets' jumbled order.

In yet another example, `xobsno` might contain 9, 6, 9, ..., 32, which is to say, observation 1 and 3 in `default` both correspond to observation 9 in `xdata`. `xdata` in this example might record geographic location and in `default`, persons in observations 1 and 3 live in the same locale.

And in a final example, `xobsno` might contain all the above and missing values (.). The missing values would indicate observations in `default` that have no corresponding observation in `xdata`. If observations 7 and 11 contained missing, that means there would be no observations in `xdata` corresponding to observations 7 and 11 in `default`. (`_frval()` has a second syntax that allows you to specify the value returned when there are no corresponding observations; see below.)

Regardless of the complexity of the example, the value of `xobsno` in observation j is the corresponding observation number i in `xdata`. Regardless of complexity, to create new variable `hrlywage` in `default`, you would type

```
. generate hrlywage = income / _frval(xdata, hrswrked, xobsno)
```

That leaves only the question of how to generate `xobsno` in all the above situations, and it is easy to do. See [\[D\] `frlink`](#).

There are two more things to know.

First, variables across frames are distinct. If the variable we have been calling `income` in `default` were named `x`, and the variable `hrswrked` in `xdata` were also named `x`, you would type

```
. generate hrlywage = x / _frval(xdata, x, xobsno)
```

Second, although we have demonstrated the use of `_frval()` with numeric variables, it works with string variables too. If `var` is a string variable name, `_frval()` returns a string result.

Domain <i>frm</i> :	any existing frame name
Domain <i>var</i> :	any existing variable name in <i>frm</i> ; varname abbreviation is allowed
Domain <i>i</i> :	any numeric values including missing values even though the nonmissing values should be integers in the range 1 to <i>frm</i> 's <code>_N</code> ; nonintegers will be interpreted as the corresponding integer obtained by truncation, and values outside the range will be treated as if they were missing value
Range:	range of <i>var</i> in <i>frm</i> plus missing value; numeric missing value (.) when <i>var</i> is numeric, and string missing value ("") when <i>var</i> is string

`_frval(frm, var, i, v)`

Description: the `_frval()` function described above but with a fourth argument *v*

`_frval()` returns values of variables stored in other frames. It returns *var*'s *i*th observation (`var[i]`) from the frame *frm*.

When *v* is specified, `_frval()` returns *v* if `var[i]` is missing or if *i* is outside the valid range of observations.

```
. generate hwage = income / _frval(xdata, hrswrked, xobsno, .z)
. generate hwage = income / _frval(xdata, hrswrked, xobsno, avg)
```

In the first case, `.z` is returned for observations in which `xobsno` contains values that are out of range. In the second case, the value recorded in variable `avg` is returned.

Domain <i>frm</i> :	any existing frame name
Domain <i>var</i> :	any existing variable name in <i>frm</i> ; varname abbreviation is allowed
Domain <i>i</i> :	any numeric values including missing values even though the nonmissing values should be integers in the range 1 to <i>frm</i> 's <code>_N</code> ; nonintegers will be interpreted as the corresponding integer obtained by truncation, and values outside the range will be treated as if they were missing value
Domain <i>v</i> :	any numeric value when <i>var</i> is numeric; any string value when <i>var</i> is string (can be a constant or vary observation by observation)
Range:	range of <i>var</i> in <i>frm</i> plus <i>v</i>

`has_eprop(name)`

Description: 1 if *name* appears as a word in `e(properties)`; otherwise, 0

Domain: names

Range: 0 or 1

inlist(*z,a,b,...*)

Description: 1 if *z* is a member of the remaining arguments; otherwise, 0

All arguments must be reals or all must be strings. The number of arguments is between 2 and 250 for reals and between 2 and 10 for strings.

Domain: all reals or all strings

Range: 0 or 1

inrange(*z,a,b*)

Description: 1 if it is known that $a \leq z \leq b$; otherwise, 0

The following ordered rules apply:

- $z \geq .$ returns 0.
- $a \geq .$ and $b = .$ returns 1.
- $a \geq .$ returns 1 if $z \leq b$; otherwise, it returns 0.
- $b \geq .$ returns 1 if $a \leq z$; otherwise, it returns 0.

Otherwise, 1 is returned if $a \leq z \leq b$.

If the arguments are strings, “.” is interpreted as “”.

Domain: all reals or all strings

Range: 0 or 1

irecode(*x,x₁,x₂,x₃,...,x_n*)

Description: *missing* if *x* is missing or x_1, \dots, x_n is not weakly increasing; 0 if $x \leq x_1$; 1 if $x_1 < x \leq x_2$; 2 if $x_2 < x \leq x_3$; ...; *n* if $x > x_n$

Also see [autocode\(\)](#) and [recode\(\)](#) for other styles of recode functions.

`irecode(3, -10, -5, -3, -3, 0, 15, .) = 5`

Domain *x*: $-8\text{e}+307$ to $8\text{e}+307$

Domain *x_i*: $-8\text{e}+307$ to $8\text{e}+307$

Range: nonnegative integers

matrix(*exp*)

Description: restricts name interpretation to scalars and matrices; see [scalar\(\)](#)

Domain: any valid expression

Range: evaluation of *exp*

maxbyte()

Description: the largest value that can be stored in storage type byte

This function takes no arguments, but the parentheses must be included.

Range: one integer number

maxdouble()

Description: the largest value that can be stored in storage type double

This function takes no arguments, but the parentheses must be included.

Range: one double-precision number

maxfloat()

Description: the largest value that can be stored in storage type float

This function takes no arguments, but the parentheses must be included.

Range: one floating-point number

maxint()

Description: the largest value that can be stored in storage type `int`

Range: This function takes no arguments, but the parentheses must be included.
one integer number

maxlong()

Description: the largest value that can be stored in storage type `long`

Range: This function takes no arguments, but the parentheses must be included.
one integer number

mi(x_1, x_2, \dots, x_n)

Description: a synonym for `missing(x_1, x_2, \dots, x_n)`

minbyte()

Description: the smallest value that can be stored in storage type `byte`

Range: This function takes no arguments, but the parentheses must be included.
one integer number

mindouble()

Description: the smallest value that can be stored in storage type `double`

Range: This function takes no arguments, but the parentheses must be included.
one double-precision number

minfloat()

Description: the smallest value that can be stored in storage type `float`

Range: This function takes no arguments, but the parentheses must be included.
one floating-point number

minint()

Description: the smallest value that can be stored in storage type `int`

Range: This function takes no arguments, but the parentheses must be included.
one integer number

minlong()

Description: the smallest value that can be stored in storage type `long`

Range: This function takes no arguments, but the parentheses must be included.
one integer number

missing(x_1, x_2, \dots, x_n)Description: 1 if any x_i evaluates to *missing*; otherwise, 0

Stata has two concepts of missing values: a numeric missing value (`.`, `.a`, `.b`, `...`, `.z`) and a string missing value (`""`). `missing()` returns 1 (meaning *true*) if any expression x_i evaluates to *missing*. If x is numeric, `missing(x)` is equivalent to $x \geq .$. If x is string, `missing(x)` is equivalent to $x == ""$.

Domain x_i : any string or numeric expression

Range: 0 and 1

r(*name*)Description: the value of the stored result `r(name)`; see [\[U\] 18.8 Accessing results calculated by other programs](#)`r(name)` = scalar missing if the stored result does not exist`r(name)` = specified matrix if the stored result is a matrix`r(name)` = scalar numeric value if the stored result is a scalar that can be interpreted as a number

Domain: names

Range: strings, scalars, matrices, or *missing***recode**(x, x_1, x_2, \dots, x_n)Description: *missing* if x_1, x_2, \dots, x_n is not weakly increasing; x if x is missing; x_1 if $x \leq x_1$; x_2 if $x \leq x_2, \dots$; otherwise, x_n if $x > x_1, x_2, \dots, x_{n-1}$. $x_i \geq .$ is interpreted as $x_i = +\infty$ Also see [autocode\(\)](#) and [irecode\(\)](#) for other styles of recode functions.Domain x : $-8e+307$ to $8e+307$ or *missing*Domain x_1 : $-8e+307$ to $8e+307$ Domain x_2 : x_1 to $8e+307$

...

Domain x_n : x_{n-1} to $8e+307$ Range: x_1, x_2, \dots, x_n or *missing***replay**()Description: 1 if the first nonblank character of local macro `'0'` is a comma, or if `'0'` is empty

This is a function for use by programmers writing estimation commands; see [\[P\] ereturn](#).

Range: integers 0 and 1, meaning *false* and *true*, respectively**return**(*name*)Description: the value of the to-be-stored result `r(name)`; see [\[P\] return](#)`return(name)` = scalar missing if the stored result does not exist`return(name)` = specified matrix if the stored result is a matrix`return(name)` = scalar numeric value if the stored result is a scalar

Domain: names

Range: strings, scalars, matrices, or *missing*

s(name)

Description: the value of stored result `s(name)`; see [U] 18.8 Accessing results calculated by other programs

`s(name) = .` if the stored result does not exist

Domain: names

Range: strings or *missing*

scalar(exp)

Description: restricts name interpretation to scalars and matrices

Names in expressions can refer to names of variables in the dataset, names of matrices, or names of scalars. Matrices and scalars can have the same names as variables in the dataset. If names conflict, Stata assumes that you are referring to the name of the variable in the dataset.

`matrix()` and `scalar()` explicitly state that you are referring to matrices and scalars. `matrix()` and `scalar()` are the same function; scalars and matrices may not have the same names and so cannot be confused. Typing `scalar(x)` makes it clear that you are referring to the scalar or matrix named `x` and not the variable named `x`, should there happen to be a variable of that name.

Domain: any valid expression

Range: evaluation of *exp*

smallestdouble()

Description: the smallest double-precision number greater than zero

If $0 < d < \text{smallestdouble}()$, then *d* does not have full double precision; these are called the denormalized numbers. This function takes no arguments, but the parentheses must be included.

Range: a double-precision number close to 0

References

Kantor, D., and N. J. Cox. 2005. Depending on conditions: A tutorial on the `cond()` function. *Stata Journal* 5: 413–420.

Rising, W. R. 2010. Stata tip 86: The `missing()` function. *Stata Journal* 10: 303–304.

Also see

[FN] [Functions by category](#)

[D] [egen](#) — Extensions to generate

[D] [generate](#) — Create or change contents of variable

[M-4] [Programming](#) — Programming functions

[U] [13.3 Functions](#)

Contents

<code>rbeta(a,b)</code>	beta(<i>a,b</i>) random variates, where <i>a</i> and <i>b</i> are the beta distribution shape parameters
<code>rbinomial(n,p)</code>	binomial(<i>n,p</i>) random variates, where <i>n</i> is the number of trials and <i>p</i> is the success probability
<code>rcauchy(a,b)</code>	Cauchy(<i>a,b</i>) random variates, where <i>a</i> is the location parameter and <i>b</i> is the scale parameter
<code>rchi2(df)</code>	chi-squared, with <i>df</i> degrees of freedom, random variates
<code>rexponential(b)</code>	exponential random variates with scale <i>b</i>
<code>rgamma(a,b)</code>	gamma(<i>a,b</i>) random variates, where <i>a</i> is the gamma shape parameter and <i>b</i> is the scale parameter
<code>rhypergeometric(N,K,n)</code>	hypergeometric random variates
<code>rigaussian(m,a)</code>	inverse Gaussian random variates with mean <i>m</i> and shape parameter <i>a</i>
<code>rlaplace(m,b)</code>	Laplace(<i>m,b</i>) random variates with mean <i>m</i> and scale parameter <i>b</i>
<code>rlogistic()</code>	logistic variates with mean 0 and standard deviation $\pi/\sqrt{3}$
<code>rlogistic(s)</code>	logistic variates with mean 0, scale <i>s</i> , and standard deviation $s\pi/\sqrt{3}$
<code>rlogistic(m,s)</code>	logistic variates with mean <i>m</i> , scale <i>s</i> , and standard deviation $s\pi/\sqrt{3}$
<code>rnbinomial(n,p)</code>	negative binomial random variates
<code>rnormal()</code>	standard normal (Gaussian) random variates, that is, variates from a normal distribution with a mean of 0 and a standard deviation of 1
<code>rnormal(m)</code>	normal(<i>m</i> ,1) (Gaussian) random variates, where <i>m</i> is the mean and the standard deviation is 1
<code>rnormal(m,s)</code>	normal(<i>m,s</i>) (Gaussian) random variates, where <i>m</i> is the mean and <i>s</i> is the standard deviation
<code>rpoisson(m)</code>	Poisson(<i>m</i>) random variates, where <i>m</i> is the distribution mean
<code>rt(df)</code>	Student's <i>t</i> random variates, where <i>df</i> is the degrees of freedom
<code>runiform()</code>	uniformly distributed random variates over the interval (0, 1)
<code>runiform(a,b)</code>	uniformly distributed random variates over the interval (<i>a</i> , <i>b</i>)
<code>runiformint(a,b)</code>	uniformly distributed random integer variates on the interval [<i>a</i> , <i>b</i>]

<code>rweibull(a,b)</code>	Weibull variates with shape a and scale b
<code>rweibull(a,b,g)</code>	Weibull variates with shape a , scale b , and location g
<code>rweibullph(a,b)</code>	Weibull (proportional hazards) variates with shape a and scale b
<code>rweibullph(a,b,g)</code>	Weibull (proportional hazards) variates with shape a , scale b , and location g

Functions

The term “pseudorandom number” is used to emphasize that the numbers are generated by formulas and are thus not truly random. From now on, we will drop the “pseudo” and just say random numbers.

For information on setting the random-number seed, see [\[R\] set seed](#).

`runiform()`

Description: uniformly distributed random variates over the interval $(0, 1)$

Range: `runiform()` can be seeded with the `set seed` command; see [\[R\] set seed](#).
`c(epsdouble)` to $1 - c(epsdouble)$

`runiform(a,b)`

Description: uniformly distributed random variates over the interval (a, b)

Domain a : `c(mindouble)` to `c(maxdouble)`

Domain b : `c(mindouble)` to `c(maxdouble)`

Range: $a + c(epsdouble)$ to $b - c(epsdouble)$

`runiformint(a,b)`

Description: uniformly distributed random integer variates on the interval $[a, b]$

If a or b is nonintegral, `runiformint(a,b)` returns `runiformint(floor(a), floor(b))`.
Domain a : -2^{53} to 2^{53} (may be nonintegral)
Domain b : -2^{53} to 2^{53} (may be nonintegral)
Range: -2^{53} to 2^{53}

`rbeta(a,b)`

Description: `beta(a,b)` random variates, where a and b are the beta distribution shape parameters

Besides using the standard methodology for generating random variates from a given distribution, `rbeta()` uses the specialized algorithms of Johnk ([Gentle 2003](#)), [Atkinson and Whittaker \(1970, 1976\)](#), [Devroye \(1986\)](#), and [Schmeiser and Babu \(1980\)](#).

Domain a : 0.05 to $1e+5$

Domain b : 0.15 to $1e+5$

Range: 0 to 1 (exclusive)

rbinomial(*n*,*p*)

Description: binomial(*n*,*p*) random variates, where *n* is the number of trials and *p* is the success probability

Besides using the standard methodology for generating random variates from a given distribution, `rbinomial()` uses the specialized algorithms of [Kachitvichyanukul \(1982\)](#), [Kachitvichyanukul and Schmeiser \(1988\)](#), and [Kemp \(1986\)](#).

Domain *n*: 1 to 1e+11

Domain *p*: 1e-8 to 1-1e-8

Range: 0 to *n*

rcauchy(*a*,*b*)

Description: Cauchy(*a*,*b*) random variates, where *a* is the location parameter and *b* is the scale parameter

Domain *a*: -1e+300 to 1e+300

Domain *b*: 1e-100 to 1e+300

Range: `c(mindouble)` to `c(maxdouble)`

rchi2(*df*)

Description: chi-squared, with *df* degrees of freedom, random variates

Domain *df*: 2e-4 to 2e+8

Range: 0 to `c(maxdouble)`

rexponential(*b*)

Description: exponential random variates with scale *b*

Domain *b*: 1e-323 to 8e+307

Range: 1e-323 to 8e+307

rgamma(*a*,*b*)

Description: gamma(*a*,*b*) random variates, where *a* is the gamma shape parameter and *b* is the scale parameter

Methods for generating gamma variates are taken from [Ahrens and Dieter \(1974\)](#), [Best \(1983\)](#), and [Schmeiser and Lal \(1980\)](#).

Domain *a*: 1e-4 to 1e+8

Domain *b*: `c(smallestdouble)` to `c(maxdouble)`

Range: 0 to `c(maxdouble)`

rhypergeometric(*N*,*K*,*n*)

Description: hypergeometric random variates

The distribution parameters are integer valued, where *N* is the population size, *K* is the number of elements in the population that have the attribute of interest, and *n* is the sample size.

Besides using the standard methodology for generating random variates from a given distribution, `rhypergeometric()` uses the specialized algorithms of [Kachitvichyanukul \(1982\)](#) and [Kachitvichyanukul and Schmeiser \(1985\)](#).

Domain *N*: 2 to 1e+6

Domain *K*: 1 to *N*-1

Domain *n*: 1 to *N*-1

Range: $\max(0, n - N + K)$ to $\min(K, n)$

rigaussian(*m,a*)

Description: inverse Gaussian random variates with mean m and shape parameter a

`rigaussian()` is based on a method proposed by Michael, Schucany, and Haas (1976).

Domain m : $1e-10$ to 1000

Domain a : 0.001 to $1e+10$

Range: 0 to $c(\text{maxdouble})$

rlaplace(*m,b*)

Description: Laplace(m,b) random variates with mean m and scale parameter b

Domain m : $-1e+300$ to $1e+300$

Domain b : $1e-300$ to $1e+300$

Range: $c(\text{mindouble})$ to $c(\text{maxdouble})$

rlogistic()

Description: logistic variates with mean 0 and standard deviation $\pi/\sqrt{3}$

The variates x are generated by $x = \text{invlogistic}(0,1,u)$, where u is a random `uniform(0,1)` variate.

Range: $c(\text{mindouble})$ to $c(\text{maxdouble})$

rlogistic(*s*)

Description: logistic variates with mean 0 , scale s , and standard deviation $s\pi/\sqrt{3}$

The variates x are generated by $x = \text{invlogistic}(0,s,u)$, where u is a random `uniform(0,1)` variate.

Domain s : 0 to $c(\text{maxdouble})$

Range: $c(\text{mindouble})$ to $c(\text{maxdouble})$

rlogistic(*m,s*)

Description: logistic variates with mean m , scale s , and standard deviation $s\pi/\sqrt{3}$

The variates x are generated by $x = \text{invlogistic}(m,s,u)$, where u is a random `uniform(0,1)` variate.

Domain m : $c(\text{mindouble})$ to $c(\text{maxdouble})$

Domain s : 0 to $c(\text{maxdouble})$

Range: $c(\text{mindouble})$ to $c(\text{maxdouble})$

rnbinomial(*n,p*)

Description: negative binomial random variates

If n is integer valued, `rnbinomial()` returns the number of failures before the n th success, where the probability of success on a single trial is p . n can also be nonintegral.

Domain n : $1e-4$ to $1e+5$

Domain p : $1e-4$ to $1-1e-4$

Range: 0 to $2^{53} - 1$

rnorm()

Description: standard normal (Gaussian) random variates, that is, variates from a normal distribution with a mean of 0 and a standard deviation of 1
Range: `c(mindouble)` to `c(maxdouble)`

rnorm(*m*)

Description: `normal(m,1)` (Gaussian) random variates, where *m* is the mean and the standard deviation is 1
Domain *m*: `c(mindouble)` to `c(maxdouble)`
Range: `c(mindouble)` to `c(maxdouble)`

rnorm(*m*,*s*)

Description: `normal(m,s)` (Gaussian) random variates, where *m* is the mean and *s* is the standard deviation
The methods for generating normal (Gaussian) random variates are taken from [Knuth \(1998, 122–128\)](#); [Marsaglia, MacLaren, and Bray \(1964\)](#); and [Walker \(1977\)](#).
Domain *m*: `c(mindouble)` to `c(maxdouble)`
Domain *s*: 0 to `c(maxdouble)`
Range: `c(mindouble)` to `c(maxdouble)`

rpoisson(*m*)

Description: `Poisson(m)` random variates, where *m* is the distribution mean
Poisson variates are generated using the probability integral transform methods of [Kemp and Kemp \(1990, 1991\)](#) and the method of [Kachitvichyanukul \(1982\)](#).
Domain *m*: `1e-6` to `1e+11`
Range: 0 to $2^{53} - 1$

rt(*df*)

Description: Student's *t* random variates, where *df* is the degrees of freedom
Student's *t* variates are generated using the method of [Kinderman and Monahan \(1977, 1980\)](#).
Domain *df*: 1 to $2^{53} - 1$
Range: `c(mindouble)` to `c(maxdouble)`

rweibull(*a*,*b*)

Description: Weibull variates with shape *a* and scale *b*
The variates *x* are generated by $x = \text{invweibulltail}(a,b,0,u)$, where *u* is a random `uniform(0,1)` variate.
Domain *a*: `0.01` to `1e+6`
Domain *b*: `1e-323` to `8e+307`
Range: `1e-323` to `8e+307`

rweibull(*a*,*b*,*g*)Description: Weibull variates with shape *a*, scale *b*, and location *g*The variates *x* are generated by $x = \text{invweibulltail}(a, b, g, u)$, where *u* is a random uniform(0,1) variate.Domain *a*: 0.01 to 1e+6Domain *b*: 1e-323 to 8e+307Domain *g*: -8e+307 to 8e+307Range: $g + c(\text{epsdouble})$ to 8e+307**rweibullph(*a*,*b*)**Description: Weibull (proportional hazards) variates with shape *a* and scale *b*The variates *x* are generated by $x = \text{invweibullphtail}(a, b, 0, u)$, where *u* is a random uniform(0,1) variate.Domain *a*: 0.01 to 1e+6Domain *b*: 1e-323 to 8e+307

Range: 1e-323 to 8e+307

rweibullph(*a*,*b*,*g*)Description: Weibull (proportional hazards) variates with shape *a*, scale *b*, and location *g*The variates *x* are generated by $x = \text{invweibullphtail}(a, b, g, u)$, where *u* is a random uniform(0,1) variate.Domain *a*: 0.01 to 1e+6Domain *b*: 1e-323 to 8e+307Domain *g*: -8e+307 to 8e+307Range: $g + c(\text{epsdouble})$ to 8e+307

Remarks and examples

It is ironic that the first thing to note about random numbers is how to make them reproducible. Before using a random-number function, type

```
set seed #
```

where *#* is any integer between 0 and $2^{31} - 1$, inclusive, to draw the same sequence of random numbers. It does not matter which integer you choose as your seed; they are all equally good. See [R] [set seed](#).

`runiform()` is the basis for all the other random-number functions because all the other random-number functions transform uniform (0,1) random numbers to the specified distribution.

`runiform()` implements the 64-bit Mersenne Twister (`mt64`), the stream 64-bit Mersenne Twister (`mt64s`), and the 32-bit “keep it simple stupid” (`kiss32`) random-number generators (RNGs) for generating uniform (0,1) random numbers. `runiform()` uses the `mt64` RNG by default.

`runiform()` uses the `kiss32` RNG only when the user version is less than 14 or when the RNG has been set to `kiss32`; see [P] [version](#) for details about setting the user version. We recommend that you do not change the default RNG, but see [R] [set rng](#) for details.

❑ Technical note

Although we recommend that you use `runiform()`, we made generator-specific versions of `runiform()` available for advanced users who want to hardcode their generator choice. The function `runiform_mt64()` always uses the `mt64` RNG to generate uniform (0, 1) random numbers, the function `runiform_mt64s()` always uses the `mt64s` RNG to generate uniform (0, 1) random numbers, the function `runiform_kiss32()` always uses the `kiss32` RNG to generate uniform (0, 1) random numbers. In fact, generator-specific versions are available for all the implemented distributions. For example, `rnormal_mt64()`, `rnormal_mt64s`, and `rnormal_kiss32()` use transforms of `mt64`, `mt64s`, and `kiss32` uniform variates, respectively, to generate standard normal variates.



❑ Technical note

Both the `mt64` and the `kiss32` RNGs produce uniform variates that pass many tests for randomness. Many researchers prefer the `mt64` to the `kiss32` RNG because the `mt64` generator has a longer period and a finer resolution and requires a higher dimension before patterns appear; see [Matsumoto and Nishimura \(1998\)](#).

The `mt64` RNG has a period of $2^{19937} - 1$ and a resolution of 2^{-53} ; see [Matsumoto and Nishimura \(1998\)](#). Each stream of the `mt64s` RNG contains 2^{128} random numbers, and `mt64s` has a resolution of 2^{-53} ; see [Haramoto et al. \(2008\)](#). The `kiss32` RNG has a period of about 2^{126} and a resolution of 2^{-32} ; see [Methods and formulas](#) below.



❑ Technical note

This technical note explains how to restart a RNG from its current spot.

The current spot in the sequence of a RNG is part of the state of a RNG. If you tell me the state of a RNG, I know where it is in its sequence, and I can compute the next random number. The state of a RNG is a complicated object that requires more space than the integers used to seed a generator. For instance, an `mt64` state is a 5011-digit, base-16 number preceded by three letters.

If you want to restart a RNG from where it left off, you should store the current state in a macro and then set the state of the RNG when you want to restart it. For example, suppose we set a seed and draw some random numbers.

```
. set obs 3
number of observations (_N) was 0, now 3
. set seed 12345
. generate x = runiform()
. list x
```

	x
1.	.3576297
2.	.4004426
3.	.6893833

We store the state of the RNG so that we can pick up right here in the sequence.

```
. local rngstate "c(rngstate)'"
```

We draw some more random numbers.

```
. replace x = runiform()  
(3 real changes made)  
. list x
```

	x
1.	.5597356
2.	.5744513
3.	.2076905

Now, we set the state of the RNG to where it was and draw those same random numbers again.

```
. set rngstate 'rngstate'  
. replace x = runiform()  
(0 real changes made)  
. list x
```

	x
1.	.5597356
2.	.5744513
3.	.2076905

□

Methods and formulas

All the nonuniform generators are based on the uniform `mt64`, `mt64s`, and `kiss32` RNGs.

The `mt64` RNG is well documented in [Matsumoto and Nishimura \(1998\)](http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html) and on their website <http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html>. The `mt64` RNG implements the 64-bit version discussed at <http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt64.html>. The `mt64s` RNG is based on a method proposed by [Haramoto et al. \(2008\)](http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt64.html). The default seed of all three RNGs is 123456789.

kiss32 generator

The `kiss32` uniform RNG implemented in `runiform()` is based on George Marsaglia's (G. Marsaglia, 1994, pers. comm.) 32-bit pseudorandom-integer generator `kiss32`. The integer `kiss32` RNG is composed of two 32-bit pseudorandom-integer generators and two 16-bit integer generators (combined to make one 32-bit integer generator). The four generators are defined by the recursions

$$x_n = 69069 x_{n-1} + 1234567 \pmod{2^{32}} \quad (1)$$

$$y_n = y_{n-1}(I + L^{13})(I + R^{17})(I + L^5) \quad (2)$$

$$z_n = 65184(z_{n-1} \bmod 2^{16}) + \text{int}(z_{n-1}/2^{16}) \quad (3)$$

$$w_n = 63663(w_{n-1} \bmod 2^{16}) + \text{int}(w_{n-1}/2^{16}) \quad (4)$$

In (2), the 32-bit word y_n is viewed as a 1×32 binary vector; L is the 32×32 matrix that produces a left shift of one (L has 1s on the first left subdiagonal, 0s elsewhere); and R is L transpose, affecting a right shift by one. In (3) and (4), $\text{int}(x)$ is the integer part of x .

The integer `kiss32` RNG produces the 32-bit random integer

$$R_n = x_n + y_n + z_n + 2^{16}w_n \mod 2^{32}$$

The `kiss32` uniform RNG implemented in `runiform()` takes the output from the integer `kiss32` RNG and divides it by 2^{32} to produce a real number on the interval $(0, 1)$. (Zeros are discarded, and the first nonzero result is returned.)

The recursion (5)–(8) have, respectively, the periods

$$2^{32} \tag{5}$$

$$2^{32} - 1 \tag{6}$$

$$(65184 \cdot 2^{16} - 2)/2 \approx 2^{31} \tag{7}$$

$$(63663 \cdot 2^{16} - 2)/2 \approx 2^{31} \tag{8}$$

Thus the overall period for the integer `kiss32` RNG is

$$2^{32} \cdot (2^{32} - 1) \cdot (65184 \cdot 2^{15} - 1) \cdot (63663 \cdot 2^{15} - 1) \approx 2^{126}$$

When Stata first comes up, it initializes the four recursions in `kiss32` by using the seeds

$$x_0 = 123456789$$

$$y_0 = 521288629$$

$$z_0 = 362436069$$

$$w_0 = 2262615$$

Successive calls to the `kiss32` uniform RNG implemented in `runiform()` then produce the sequence

$$\frac{R_1}{2^{32}}, \frac{R_2}{2^{32}}, \frac{R_3}{2^{32}}, \dots$$

Hence, the `kiss32` uniform RNG implemented in `runiform()` gives the same sequence of random numbers in every Stata session (measured from the start of the session) unless you reinitialize the seed. The full seed is the set of four numbers (x, y, z, w) , but you can reinitialize the seed by simply issuing the command

```
. set seed #
```

where $\#$ is any integer between 0 and $2^{31} - 1$, inclusive. When this command is issued, the initial value x_0 is set equal to $\#$, and the other three recursions are restarted at the seeds y_0 , z_0 , and w_0 given above. The first 100 random numbers are discarded, and successive calls to the `kiss32` uniform RNG implemented in `runiform()` give the sequence

$$\frac{R'_{101}}{2^{32}}, \frac{R'_{102}}{2^{32}}, \frac{R'_{103}}{2^{32}}, \dots$$

However, if the command

```
. set seed 123456789
```

is given, the first 100 random numbers are not discarded, and you get the same sequence of random numbers that the `kiss32` RNG produces when Stata restarts; also see [R] [set seed](#).

Acknowledgments

We thank the late George Marsaglia, formerly of Florida State University, for providing his `kiss32` RNG.

We thank John R. Gleason (retired) of Syracuse University for directing our attention to [Wichura \(1988\)](#) for calculating the cumulative normal density accurately, for sharing his experiences about techniques with us, and for providing C code to make the calculations.

We thank Makoto Matsumoto and Takuji Nishimura for deriving the Mersenne Twister and distributing their code for their generator so that it could be rapidly and effectively tested.

References

- Ahrens, J. H., and U. Dieter. 1974. Computer methods for sampling from gamma, beta, Poisson, and binomial distributions. *Computing* 12: 223–246.
- Atkinson, A. C., and J. C. Whittaker. 1970. Algorithm AS 134: The generation of beta random variables with one parameter greater than and one parameter less than 1. *Applied Statistics* 28: 90–93.
- . 1976. A switching algorithm for the generation of beta random variables with at least one parameter less than 1. *Journal of the Royal Statistical Society, Series A* 139: 462–467.
- Best, D. J. 1983. A note on gamma variate generators with shape parameters less than unity. *Computing* 30: 185–188.
- Buis, M. L. 2007. [Stata tip 48: Discrete uses for `uniform\(\)`](#). *Stata Journal* 7: 434–435.
- Devroye, L. 1986. *Non-uniform Random Variate Generation*. New York: Springer.
- Gentle, J. E. 2003. *Random Number Generation and Monte Carlo Methods*. 2nd ed. New York: Springer.
- Gopal, K. 2016. How to generate random numbers in Stata. *The Stata Blog: Not Elsewhere Classified*. <http://blog.stata.com/2016/03/10/how-to-generate-random-numbers-in-stata/>.
- Gould, W. W. 2012a. Using Stata's random-number generators, part 1. *The Stata Blog: Not Elsewhere Classified*. <http://blog.stata.com/2012/07/18/using-statas-random-number-generators-part-1/>.
- . 2012b. Using Stata's random-number generators, part 2: Drawing without replacement. *The Stata Blog: Not Elsewhere Classified*. <http://blog.stata.com/2012/08/03/using-statas-random-number-generators-part-2-drawing-without-replacement/>.
- . 2012c. Using Stata's random-number generators, part 3: Drawing with replacement. *The Stata Blog: Not Elsewhere Classified*. <http://blog.stata.com/2012/08/29/using-statas-random-number-generators-part-3-drawing-with-replacement/>.
- . 2012d. Using Stata's random-number generators, part 4: Details. *The Stata Blog: Not Elsewhere Classified*. <http://blog.stata.com/2012/10/24/using-statas-random-number-generators-part-4-details/>.
- Grayling, M. J., and A. P. Mander. 2018. Calculations involving the multivariate normal and multivariate t distributions with and without truncation. *Stata Journal* 18: 826–843.
- Haramoto, H., M. Matsumoto, T. Nishimura, F. Panneton, and P. L'Ecuyer. 2008. Efficient jump ahead for F2-linear random number generators. *INFORMS Journal on Computing* 20: 385–390.
- Hilbe, J. M. 2010. [Creating synthetic discrete-response regression models](#). *Stata Journal* 10: 104–124.
- Huber, C. 2014. How to simulate multilevel/longitudinal data. *The Stata Blog: Not Elsewhere Classified*. <http://blog.stata.com/2014/07/18/how-to-simulate-multilevellongitudinal-data/>.

- Kachitvichyanukul, V. 1982. Computer Generation of Poisson, Binomial, and Hypergeometric Random Variables. PhD thesis, Purdue University.
- Kachitvichyanukul, V., and B. W. Schmeiser. 1985. Computer generation of hypergeometric random variates. *Journal of Statistical Computation and Simulation* 22: 127–145.
- . 1988. Binomial random variate generation. *Communications of the Association for Computing Machinery* 31: 216–222.
- Kemp, A. W., and C. D. Kemp. 1990. A composition-search algorithm for low-parameter Poisson generation. *Journal of Statistical Computation and Simulation* 35: 239–244.
- Kemp, C. D. 1986. A modal method for generating binomial variates. *Communications in Statistics—Theory and Methods* 15: 805–813.
- Kemp, C. D., and A. W. Kemp. 1991. Poisson random variate generation. *Applied Statistics* 40: 143–158.
- Kinderman, A. J., and J. F. Monahan. 1977. Computer generation of random variables using the ratio of uniform deviates. *ACM Transactions on Mathematical Software* 3: 257–260.
- . 1980. New methods for generating Student's t and gamma variables. *Computing* 25: 369–377.
- Knuth, D. E. 1998. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*. 3rd ed. Reading, MA: Addison-Wesley.
- Lee, S. 2015. [Generating univariate and multivariate nonnormal data](#). *Stata Journal* 15: 95–109.
- Lukácsy, K. 2011. [Generating random samples from user-defined distributions](#). *Stata Journal* 11: 299–304.
- Marsaglia, G., M. D. MacLaren, and T. A. Bray. 1964. A fast procedure for generating normal random variables. *Communications of the Association for Computing Machinery* 7: 4–10.
- Matsumoto, M., and T. Nishimura. 1998. Mersenne Twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation* 8: 3–30.
- Michael, J. R., W. R. Schucany, and R. W. Haas. 1976. Generating random variates using transformations with multiple roots. *American Statistician* 30: 88–90.
- Schmeiser, B. W., and A. J. G. Babu. 1980. Beta variate generation via exponential majorizing functions. *Operations Research* 28: 917–926.
- Schmeiser, B. W., and R. Lal. 1980. Squeeze methods for generating gamma variates. *Journal of the American Statistical Association* 75: 679–682.
- Walker, A. J. 1977. An efficient method for generating discrete random variables with general distributions. *ACM Transactions on Mathematical Software* 3: 253–256.
- Wichura, M. J. 1988. Algorithm AS241: The percentage points of the normal distribution. *Applied Statistics* 37: 477–484.

Also see

[FN] [Functions by category](#)

[D] [egen](#) — Extensions to generate

[D] [generate](#) — Create or change contents of variable

[R] [set rng](#) — Set which random-number generator (RNG) to use

[R] [set rngstream](#) — Specify the stream for the stream random-number generator

[R] [set seed](#) — Specify random-number seed and state

[M-5] [runiform\(\)](#) — Uniform and nonuniform pseudorandom variates

[U] [13.3 Functions](#)

Contents

<code>tin(<i>d</i>₁,<i>d</i>₂)</code>	<i>true</i> if $d_1 \leq t \leq d_2$, where t is the time variable previously <code>tsset</code>
<code>twithin(<i>d</i>₁,<i>d</i>₂)</code>	<i>true</i> if $d_1 < t < d_2$, where t is the time variable previously <code>tsset</code>

Functions

`tin(d1,d2)`

Description: *true* if $d_1 \leq t \leq d_2$, where t is the time variable previously `tsset`

You must have previously `tsset` the data to use `tin()`; see [TS] `tsset`. When you `tsset` the data, you specify a time variable, t , and the format on t states how it is recorded. You type d_1 and d_2 according to that format.

If t has a `%tc` format, you could type `tin(5jan1992 11:15, 14apr2002 12:25)`.

If t has a `%td` format, you could type `tin(5jan1992, 14apr2002)`.

If t has a `%tw` format, you could type `tin(1985w1, 2002w15)`.

If t has a `%tm` format, you could type `tin(1985m1, 2002m4)`.

If t has a `%tq` format, you could type `tin(1985q1, 2002q2)`.

If t has a `%th` format, you could type `tin(1985h1, 2002h1)`.

If t has a `%ty` format, you could type `tin(1985, 2002)`.

If t has a `%tb` format, you could type `tin(5jan1992, 14apr2002)`. This will work as expected even if the arguments of `tin()` are not business days.

Otherwise, t is just a set of integers, and you could type `tin(12, 38)`.

The details of the `%t` format do not matter. If your t is formatted `%tdmm/dd/yy` so that 5jan1992 displays as 1/5/92, you would still type the date in day–month–year order: `tin(5jan1992, 14apr2002)`.

Domain d_1 : date or time literals or strings recorded in units of t previously `tsset` or blank to indicate no minimum date

Domain d_2 : date or time literals or strings recorded in units of t previously `tsset` or blank to indicate no maximum date

Range: 0 and 1, 1 \Rightarrow *true*

`twwithin(d_1, d_2)`

Description: *true* if $d_1 < t < d_2$, where t is the time variable previously `tsset`

See `tin()` above; `twwithin()` is similar, except the range is exclusive.

Domain d_1 : date or time literals or strings recorded in units of t previously `tsset` or blank to indicate no minimum date

Domain d_2 : date or time literals or strings recorded in units of t previously `tsset` or blank to indicate no maximum date

Range: 0 and 1, $1 \Rightarrow \textit{true}$

Also see

[FN] **Functions by category**

[D] **egen** — Extensions to generate

[D] **generate** — Create or change contents of variable

[U] **13.3 Functions**

Contents

<code>betaden(<i>a</i>,<i>b</i>,<i>x</i>)</code>	the probability density of the beta distribution, where <i>a</i> and <i>b</i> are the shape parameters; 0 if $x < 0$ or $x > 1$
<code>binomial(<i>n</i>,<i>k</i>,θ)</code>	the probability of observing <code>floor(<i>k</i>)</code> or fewer successes in <code>floor(<i>n</i>)</code> trials when the probability of a success on one trial is θ ; 0 if $k < 0$; or 1 if $k > n$
<code>binomialp(<i>n</i>,<i>k</i>,<i>p</i>)</code>	the probability of observing <code>floor(<i>k</i>)</code> successes in <code>floor(<i>n</i>)</code> trials when the probability of a success on one trial is <i>p</i>
<code>binomialtail(<i>n</i>,<i>k</i>,θ)</code>	the probability of observing <code>floor(<i>k</i>)</code> or more successes in <code>floor(<i>n</i>)</code> trials when the probability of a success on one trial is θ ; 1 if $k < 0$; or 0 if $k > n$
<code>binormal(<i>h</i>,<i>k</i>,ρ)</code>	the joint cumulative distribution $\Phi(h, k, \rho)$ of bivariate normal with correlation ρ
<code>cauchy(<i>a</i>,<i>b</i>,<i>x</i>)</code>	the cumulative Cauchy distribution with location parameter <i>a</i> and scale parameter <i>b</i>
<code>cauchyden(<i>a</i>,<i>b</i>,<i>x</i>)</code>	the probability density of the Cauchy distribution with location parameter <i>a</i> and scale parameter <i>b</i>
<code>cauchytail(<i>a</i>,<i>b</i>,<i>x</i>)</code>	the reverse cumulative (upper tail or survivor) Cauchy distribution with location parameter <i>a</i> and scale parameter <i>b</i>
<code>chi2(<i>df</i>,<i>x</i>)</code>	the cumulative χ^2 distribution with <i>df</i> degrees of freedom; 0 if $x < 0$
<code>chi2den(<i>df</i>,<i>x</i>)</code>	the probability density of the chi-squared distribution with <i>df</i> degrees of freedom; 0 if $x < 0$
<code>chi2tail(<i>df</i>,<i>x</i>)</code>	the reverse cumulative (upper tail or survivor) χ^2 distribution with <i>df</i> degrees of freedom; 1 if $x < 0$
<code>dgammapda(<i>a</i>,<i>x</i>)</code>	$\frac{\partial P(a, x)}{\partial a}$, where $P(a, x) = \text{gammap}(a, x)$; 0 if $x < 0$
<code>dgammapdada(<i>a</i>,<i>x</i>)</code>	$\frac{\partial^2 P(a, x)}{\partial a^2}$, where $P(a, x) = \text{gammap}(a, x)$; 0 if $x < 0$
<code>dgammapdadx(<i>a</i>,<i>x</i>)</code>	$\frac{\partial^2 P(a, x)}{\partial a \partial x}$, where $P(a, x) = \text{gammap}(a, x)$; 0 if $x < 0$
<code>dgammapdx(<i>a</i>,<i>x</i>)</code>	$\frac{\partial P(a, x)}{\partial x}$, where $P(a, x) = \text{gammap}(a, x)$; 0 if $x < 0$
<code>dgammapdxdx(<i>a</i>,<i>x</i>)</code>	$\frac{\partial^2 P(a, x)}{\partial x^2}$, where $P(a, x) = \text{gammap}(a, x)$; 0 if $x < 0$
<code>dunnettprob(<i>k</i>,<i>df</i>,<i>x</i>)</code>	the cumulative multiple range distribution that is used in Dunnett's multiple-comparison method with <i>k</i> ranges and <i>df</i> degrees of freedom; 0 if $x < 0$
<code>exponential(<i>b</i>,<i>x</i>)</code>	the cumulative exponential distribution with scale <i>b</i>
<code>exponentialden(<i>b</i>,<i>x</i>)</code>	the probability density function of the exponential distribution with scale <i>b</i>
<code>exponentialtail(<i>b</i>,<i>x</i>)</code>	the reverse cumulative exponential distribution with scale <i>b</i>

<code>F(df₁, df₂, f)</code>	the cumulative F distribution with df_1 numerator and df_2 denominator degrees of freedom: $F(df_1, df_2, f) = \int_0^f \text{Fden}(df_1, df_2, t) dt$; 0 if $f < 0$
<code>Fden(df₁, df₂, f)</code>	the probability density function of the F distribution with df_1 numerator and df_2 denominator degrees of freedom; 0 if $f < 0$
<code>Ftail(df₁, df₂, f)</code>	the reverse cumulative (upper tail or survivor) F distribution with df_1 numerator and df_2 denominator degrees of freedom; 1 if $f < 0$
<code>gammaden(a, b, g, x)</code>	the probability density function of the gamma distribution; 0 if $x < g$
<code>gammap(a, x)</code>	the cumulative gamma distribution with shape parameter a ; 0 if $x < 0$
<code>gammaptail(a, x)</code>	the reverse cumulative (upper tail or survivor) gamma distribution with shape parameter a ; 1 if $x < 0$
<code>hypergeometric(N, K, n, k)</code>	the cumulative probability of the hypergeometric distribution
<code>hypergeometricp(N, K, n, k)</code>	the hypergeometric probability of k successes out of a sample of size n , from a population of size N containing K elements that have the attribute of interest
<code>ibeta(a, b, x)</code>	the cumulative beta distribution with shape parameters a and b ; 0 if $x < 0$; or 1 if $x > 1$
<code>ibetatail(a, b, x)</code>	the reverse cumulative (upper tail or survivor) beta distribution with shape parameters a and b ; 1 if $x < 0$; or 0 if $x > 1$
<code>igaussian(m, a, x)</code>	the cumulative inverse Gaussian distribution with mean m and shape parameter a ; 0 if $x \leq 0$
<code>igaussianden(m, a, x)</code>	the probability density of the inverse Gaussian distribution with mean m and shape parameter a ; 0 if $x \leq 0$
<code>igaussiantail(m, a, x)</code>	the reverse cumulative (upper tail or survivor) inverse Gaussian distribution with mean m and shape parameter a ; 1 if $x \leq 0$
<code>invbinomial(n, k, p)</code>	the inverse of the cumulative binomial; that is, θ (θ = probability of success on one trial) such that the probability of observing <code>floor(k)</code> or fewer successes in <code>floor(n)</code> trials is p
<code>invbinomialtail(n, k, p)</code>	the inverse of the right cumulative binomial; that is, θ (θ = probability of success on one trial) such that the probability of observing <code>floor(k)</code> or more successes in <code>floor(n)</code> trials is p
<code>invcauchy(a, b, p)</code>	the inverse of <code>cauchy()</code> : if <code>cauchy(a, b, x) = p</code> , then <code>invcauchy(a, b, p) = x</code>
<code>invcauchytail(a, b, p)</code>	the inverse of <code>cauchytail()</code> : if <code>cauchytail(a, b, x) = p</code> , then <code>invcauchytail(a, b, p) = x</code>
<code>invchi2(df, p)</code>	the inverse of <code>chi2()</code> : if <code>chi2(df, x) = p</code> , then <code>invchi2(df, p) = x</code>
<code>invchi2tail(df, p)</code>	the inverse of <code>chi2tail()</code> : if <code>chi2tail(df, x) = p</code> , then <code>invchi2tail(df, p) = x</code>
<code>invdunnettprob(k, df, p)</code>	the inverse cumulative multiple range distribution that is used in Dunnett's multiple-comparison method with k ranges and df degrees of freedom
<code>invexponential(b, p)</code>	the inverse cumulative exponential distribution with scale b : if <code>exponential(b, x) = p</code> , then <code>invexponential(b, p) = x</code>

<code>invexponentialtail(b,p)</code>	the inverse reverse cumulative exponential distribution with scale b : if <code>exponentialtail(b,x) = p</code> , then <code>invexponentialtail(b,p) = x</code>
<code>invF(df₁,df₂,p)</code>	the inverse cumulative F distribution: if <code>F(df₁,df₂,f) = p</code> , then <code>invF(df₁,df₂,p) = f</code>
<code>invFtail(df₁,df₂,p)</code>	the inverse reverse cumulative (upper tail or survivor) F distribution: if <code>Ftail(df₁,df₂,f) = p</code> , then <code>invFtail(df₁,df₂,p) = f</code>
<code>invgammap(a,p)</code>	the inverse cumulative gamma distribution: if <code>gammap(a,x) = p</code> , then <code>invgammap(a,p) = x</code>
<code>invgammaptail(a,p)</code>	the inverse reverse cumulative (upper tail or survivor) gamma distribution: if <code>gammaptail(a,x) = p</code> , then <code>invgammaptail(a,p) = x</code>
<code>invibeta(a,b,p)</code>	the inverse cumulative beta distribution: if <code>ibeta(a,b,x) = p</code> , then <code>invibeta(a,b,p) = x</code>
<code>invibetatail(a,b,p)</code>	the inverse reverse cumulative (upper tail or survivor) beta distribution: if <code>ibetatail(a,b,x) = p</code> , then <code>invibetatail(a,b,p) = x</code>
<code>invgaussian(m,a,p)</code>	the inverse of <code>igaussian()</code> : if <code>igaussian(m,a,x) = p</code> , then <code>invgaussian(m,a,p) = x</code>
<code>invgaussiantail(m,a,p)</code>	the inverse of <code>igaussiantail()</code> : if <code>igaussiantail(m,a,x) = p</code> , then <code>invgaussiantail(m,a,p) = x</code>
<code>invlaplace(m,b,p)</code>	the inverse of <code>laplace()</code> : if <code>laplace(m,b,x) = p</code> , then <code>invlaplace(m,b,p) = x</code>
<code>invlaplacetail(m,b,p)</code>	the inverse of <code>laplacetail()</code> : if <code>laplacetail(m,b,x) = p</code> , then <code>invlaplacetail(m,b,p) = x</code>
<code>invlogistic(p)</code>	the inverse cumulative logistic distribution: if <code>logistic(x) = p</code> , then <code>invlogistic(p) = x</code>
<code>invlogistic(s,p)</code>	the inverse cumulative logistic distribution: if <code>logistic(s,x) = p</code> , then <code>invlogistic(s,p) = x</code>
<code>invlogistic(m,s,p)</code>	the inverse cumulative logistic distribution: if <code>logistic(m,s,x) = p</code> , then <code>invlogistic(m,s,p) = x</code>
<code>invlogistictail(p)</code>	the inverse reverse cumulative logistic distribution: if <code>logistictail(x) = p</code> , then <code>invlogistictail(p) = x</code>
<code>invlogistictail(s,p)</code>	the inverse reverse cumulative logistic distribution: if <code>logistictail(s,x) = p</code> , then <code>invlogistictail(s,p) = x</code>
<code>invlogistictail(m,s,p)</code>	the inverse reverse cumulative logistic distribution: if <code>logistictail(m,s,x) = p</code> , then <code>invlogistictail(m,s,p) = x</code>
<code>invnbinomial(n,k,q)</code>	the value of the negative binomial parameter, p , such that $q =$ <code>nbinomial(n,k,p)</code>
<code>invnbinomialtail(n,k,q)</code>	the value of the negative binomial parameter, p , such that $q =$ <code>nbinomialtail(n,k,p)</code>
<code>invnchi2(df,np,p)</code>	the inverse cumulative noncentral χ^2 distribution: if <code>nchi2(df,np,x) = p</code> , then <code>invnchi2(df,np,p) = x</code>
<code>invnchi2tail(df,np,p)</code>	the inverse reverse cumulative (upper tail or survivor) non-central χ^2 distribution: if <code>nchi2tail(df,np,x) = p</code> , then <code>invnchi2tail(df,np,p) = x</code>

<code>invnF(df₁,df₂,np,p)</code>	the inverse cumulative noncentral F distribution: if $\text{nF}(df_1,df_2,np,f) = p$, then $\text{invnF}(df_1,df_2,np,p) = f$
<code>invnFtail(df₁,df₂,np,p)</code>	the inverse reverse cumulative (upper tail or survivor) noncentral F distribution: if $\text{nFtail}(df_1,df_2,np,f) = p$, then $\text{invnFtail}(df_1,df_2,np,p) = f$
<code>invnibeta(a,b,np,p)</code>	the inverse cumulative noncentral beta distribution: if $\text{nibeta}(a,b,np,x) = p$, then $\text{invnibeta}(a,b,np,p) = x$
<code>invnormal(p)</code>	the inverse cumulative standard normal distribution: if $\text{normal}(z) = p$, then $\text{invnormal}(p) = z$
<code>invnt(df,np,p)</code>	the inverse cumulative noncentral Student's t distribution: if $\text{nt}(df,np,t) = p$, then $\text{invnt}(df,np,p) = t$
<code>invnttail(df,np,p)</code>	the inverse reverse cumulative (upper tail or survivor) noncentral Student's t distribution: if $\text{nttail}(df,np,t) = p$, then $\text{invnttail}(df,np,p) = t$
<code>invpoisson(k,p)</code>	the Poisson mean such that the cumulative Poisson distribution evaluated at k is p : if $\text{poisson}(m,k) = p$, then $\text{invpoisson}(k,p) = m$
<code>invpoisontail(k,q)</code>	the Poisson mean such that the reverse cumulative Poisson distribution evaluated at k is q : if $\text{poisontail}(m,k) = q$, then $\text{invpoisontail}(k,q) = m$
<code>invt(df,p)</code>	the inverse cumulative Student's t distribution: if $\text{t}(df,t) = p$, then $\text{invt}(df,p) = t$
<code>invttail(df,p)</code>	the inverse reverse cumulative (upper tail or survivor) Student's t distribution: if $\text{ttail}(df,t) = p$, then $\text{invttail}(df,p) = t$
<code>invtukeyprob(k,df,p)</code>	the inverse cumulative Tukey's Studentized range distribution with k ranges and df degrees of freedom
<code>invweibull(a,b,p)</code>	the inverse cumulative Weibull distribution with shape a and scale b : if $\text{weibull}(a,b,x) = p$, then $\text{invweibull}(a,b,p) = x$
<code>invweibull(a,b,g,p)</code>	the inverse cumulative Weibull distribution with shape a , scale b , and location g : if $\text{weibull}(a,b,g,x) = p$, then $\text{invweibull}(a,b,g,p) = x$
<code>invweibullph(a,b,p)</code>	the inverse cumulative Weibull (proportional hazards) distribution with shape a and scale b : if $\text{weibullph}(a,b,x) = p$, then $\text{invweibullph}(a,b,p) = x$
<code>invweibullph(a,b,g,p)</code>	the inverse cumulative Weibull (proportional hazards) distribution with shape a , scale b , and location g : if $\text{weibullph}(a,b,g,x) = p$, then $\text{invweibullph}(a,b,g,p) = x$
<code>invweibullphtail(a,b,p)</code>	the inverse reverse cumulative Weibull (proportional hazards) distribution with shape a and scale b : if $\text{weibullphtail}(a,b,x) = p$, then $\text{invweibullphtail}(a,b,p) = x$
<code>invweibullphtail(a,b,g,p)</code>	the inverse reverse cumulative Weibull (proportional hazards) distribution with shape a , scale b , and location g : if $\text{weibullphtail}(a,b,g,x) = p$, then $\text{invweibullphtail}(a,b,g,p) = x$
<code>invweibulltail(a,b,p)</code>	the inverse reverse cumulative Weibull distribution with shape a and scale b : if $\text{weibulltail}(a,b,x) = p$, then $\text{invweibulltail}(a,b,p) = x$
<code>invweibulltail(a,b,g,p)</code>	the inverse reverse cumulative Weibull distribution with shape a , scale b , and location g : if $\text{weibulltail}(a,b,g,x) = p$, then $\text{invweibulltail}(a,b,g,p) = x$

<code>laplace(m, b, x)</code>	the cumulative Laplace distribution with mean m and scale parameter b
<code>laplaceden(m, b, x)</code>	the probability density of the Laplace distribution with mean m and scale parameter b
<code>laplacetail(m, b, x)</code>	the reverse cumulative (upper tail or survivor) Laplace distribution with mean m and scale parameter b
<code>lncauchyden(a, b, x)</code>	the natural logarithm of the density of the Cauchy distribution with location parameter a and scale parameter b
<code>lnigammaden(a, b, x)</code>	the natural logarithm of the inverse gamma density, where a is the shape parameter and b is the scale parameter
<code>lnigaussianden(m, a, x)</code>	the natural logarithm of the inverse Gaussian density with mean m and shape parameter a
<code>lniwishartden(df, V, X)</code>	the natural logarithm of the density of the inverse Wishart distribution; missing if $df \leq n - 1$
<code>lnlaplaceden(m, b, x)</code>	the natural logarithm of the density of the Laplace distribution with mean m and scale parameter b
<code>lnmvnormalden(M, V, X)</code>	the natural logarithm of the multivariate normal density
<code>lnnormal(z)</code>	the natural logarithm of the cumulative standard normal distribution
<code>lnnormalden(z)</code>	the natural logarithm of the standard normal density, $N(0, 1)$
<code>lnnormalden(x, σ)</code>	the natural logarithm of the normal density with mean 0 and standard deviation σ
<code>lnnormalden(x, μ, σ)</code>	the natural logarithm of the normal density with mean μ and standard deviation σ , $N(\mu, \sigma^2)$
<code>lnwishartden(df, V, X)</code>	the natural logarithm of the density of the Wishart distribution; missing if $df \leq n - 1$
<code>logistic(x)</code>	the cumulative logistic distribution with mean 0 and standard deviation $\pi/\sqrt{3}$
<code>logistic(s, x)</code>	the cumulative logistic distribution with mean 0, scale s , and standard deviation $s\pi/\sqrt{3}$
<code>logistic(m, s, x)</code>	the cumulative logistic distribution with mean m , scale s , and standard deviation $s\pi/\sqrt{3}$
<code>logisticden(x)</code>	the density of the logistic distribution with mean 0 and standard deviation $\pi/\sqrt{3}$
<code>logisticden(s, x)</code>	the density of the logistic distribution with mean 0, scale s , and standard deviation $s\pi/\sqrt{3}$
<code>logisticden(m, s, x)</code>	the density of the logistic distribution with mean m , scale s , and standard deviation $s\pi/\sqrt{3}$
<code>logistictail(x)</code>	the reverse cumulative logistic distribution with mean 0 and standard deviation $\pi/\sqrt{3}$
<code>logistictail(s, x)</code>	the reverse cumulative logistic distribution with mean 0, scale s , and standard deviation $s\pi/\sqrt{3}$
<code>logistictail(m, s, x)</code>	the reverse cumulative logistic distribution with mean m , scale s , and standard deviation $s\pi/\sqrt{3}$
<code>nbetaden(a, b, np, x)</code>	the probability density function of the noncentral beta distribution; 0 if $x < 0$ or $x > 1$
<code>nbinomial(n, k, p)</code>	the cumulative probability of the negative binomial distribution

<code>nbinomialp(<i>n</i>,<i>k</i>,<i>p</i>)</code>	the negative binomial probability
<code>nbinomialtail(<i>n</i>,<i>k</i>,<i>p</i>)</code>	the reverse cumulative probability of the negative binomial distribution
<code>nchi2(<i>df</i>,<i>np</i>,<i>x</i>)</code>	the cumulative noncentral χ^2 distribution; 0 if $x < 0$
<code>nchi2den(<i>df</i>,<i>np</i>,<i>x</i>)</code>	the probability density of the noncentral χ^2 distribution; 0 if $x < 0$
<code>nchi2tail(<i>df</i>,<i>np</i>,<i>x</i>)</code>	the reverse cumulative (upper tail or survivor) noncentral χ^2 distribution; 1 if $x < 0$
<code>nF(<i>df</i>₁,<i>df</i>₂,<i>np</i>,<i>f</i>)</code>	the cumulative noncentral F distribution with df_1 numerator and df_2 denominator degrees of freedom and noncentrality parameter np ; 0 if $f < 0$
<code>nFden(<i>df</i>₁,<i>df</i>₂,<i>np</i>,<i>f</i>)</code>	the probability density function of the noncentral F distribution with df_1 numerator and df_2 denominator degrees of freedom and noncentrality parameter np ; 0 if $f < 0$
<code>nFtail(<i>df</i>₁,<i>df</i>₂,<i>np</i>,<i>f</i>)</code>	the reverse cumulative (upper tail or survivor) noncentral F distribution with df_1 numerator and df_2 denominator degrees of freedom and noncentrality parameter np ; 1 if $f < 0$
<code>nibeta(<i>a</i>,<i>b</i>,<i>np</i>,<i>x</i>)</code>	the cumulative noncentral beta distribution; 0 if $x < 0$; or 1 if $x > 1$
<code>normal(<i>z</i>)</code>	the cumulative standard normal distribution
<code>normalden(<i>z</i>)</code>	the standard normal density, $N(0, 1)$
<code>normalden(<i>x</i>,σ)</code>	the normal density with mean 0 and standard deviation σ
<code>normalden(<i>x</i>,μ,σ)</code>	the normal density with mean μ and standard deviation σ , $N(\mu, \sigma^2)$
<code>npnchi2(<i>df</i>,<i>x</i>,<i>p</i>)</code>	the noncentrality parameter, np , for noncentral χ^2 : if $nchi2(df, np, x) = p$, then <code>npnchi2(<i>df</i>,<i>x</i>,<i>p</i>)</code> = np
<code>npnF(<i>df</i>₁,<i>df</i>₂,<i>f</i>,<i>p</i>)</code>	the noncentrality parameter, np , for the noncentral F : if $nF(df_1, df_2, np, f) = p$, then <code>npnF(<i>df</i>₁,<i>df</i>₂,<i>f</i>,<i>p</i>)</code> = np
<code>npnt(<i>df</i>,<i>t</i>,<i>p</i>)</code>	the noncentrality parameter, np , for the noncentral Student's t distribution: if $nt(df, np, t) = p$, then <code>npnt(<i>df</i>,<i>t</i>,<i>p</i>)</code> = np
<code>nt(<i>df</i>,<i>np</i>,<i>t</i>)</code>	the cumulative noncentral Student's t distribution with df degrees of freedom and noncentrality parameter np
<code>ntden(<i>df</i>,<i>np</i>,<i>t</i>)</code>	the probability density function of the noncentral Student's t distribution with df degrees of freedom and noncentrality parameter np
<code>nttail(<i>df</i>,<i>np</i>,<i>t</i>)</code>	the reverse cumulative (upper tail or survivor) noncentral Student's t distribution with df degrees of freedom and noncentrality parameter np
<code>poisson(<i>m</i>,<i>k</i>)</code>	the probability of observing <code>floor(<i>k</i>)</code> or fewer outcomes that are distributed as Poisson with mean m
<code>poissonp(<i>m</i>,<i>k</i>)</code>	the probability of observing <code>floor(<i>k</i>)</code> outcomes that are distributed as Poisson with mean m
<code>poisontail(<i>m</i>,<i>k</i>)</code>	the probability of observing <code>floor(<i>k</i>)</code> or more outcomes that are distributed as Poisson with mean m
<code>t(<i>df</i>,<i>t</i>)</code>	the cumulative Student's t distribution with df degrees of freedom
<code>tden(<i>df</i>,<i>t</i>)</code>	the probability density function of Student's t distribution
<code>ttail(<i>df</i>,<i>t</i>)</code>	the reverse cumulative (upper tail or survivor) Student's t distribution; the probability $T > t$
<code>tukeyprob(<i>k</i>,<i>df</i>,<i>x</i>)</code>	the cumulative Tukey's Studentized range distribution with k ranges and df degrees of freedom; 0 if $x < 0$

<code>weibull(a,b,x)</code>	the cumulative Weibull distribution with shape a and scale b
<code>weibull(a,b,g,x)</code>	the cumulative Weibull distribution with shape a , scale b , and location g
<code>weibullden(a,b,x)</code>	the probability density function of the Weibull distribution with shape a and scale b
<code>weibullden(a,b,g,x)</code>	the probability density function of the Weibull distribution with shape a , scale b , and location g
<code>weibullph(a,b,x)</code>	the cumulative Weibull (proportional hazards) distribution with shape a and scale b
<code>weibullph(a,b,g,x)</code>	the cumulative Weibull (proportional hazards) distribution with shape a , scale b , and location g
<code>weibullphden(a,b,x)</code>	the probability density function of the Weibull (proportional hazards) distribution with shape a and scale b
<code>weibullphden(a,b,g,x)</code>	the probability density function of the Weibull (proportional hazards) distribution with shape a , scale b , and location g
<code>weibullphtail(a,b,x)</code>	the reverse cumulative Weibull (proportional hazards) distribution with shape a and scale b
<code>weibullphtail(a,b,g,x)</code>	the reverse cumulative Weibull (proportional hazards) distribution with shape a , scale b , and location g
<code>weibulltail(a,b,x)</code>	the reverse cumulative Weibull distribution with shape a and scale b
<code>weibulltail(a,b,g,x)</code>	the reverse cumulative Weibull distribution with shape a , scale b , and location g

Functions

Statistical functions are listed alphabetically under the following headings:

- Beta and noncentral beta distributions*
- Binomial distribution*
- Cauchy distribution*
- Chi-squared and noncentral chi-squared distributions*
- Dunnett's multiple range distribution*
- Exponential distribution*
- F and noncentral F distributions*
- Gamma distribution*
- Hypergeometric distribution*
- Inverse Gaussian distribution*
- Laplace distribution*
- Logistic distribution*
- Negative binomial distribution*
- Normal (Gaussian), binormal, and multivariate normal distributions*
- Poisson distribution*
- Student's t and noncentral Student's t distributions*
- Tukey's Studentized range distribution*
- Weibull distribution*
- Weibull (proportional hazards) distribution*
- Wishart distribution*

Beta and noncentral beta distributions

`betaden(a,b,x)`

Description: the probability density of the beta distribution, where *a* and *b* are the shape parameters; 0 if *x* < 0 or *x* > 1

The probability density of the beta distribution is

$$\text{betaden}(a,b,x) = \frac{x^{a-1}(1-x)^{b-1}}{\int_0^1 t^{a-1}(1-t)^{b-1}dt} = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}x^{a-1}(1-x)^{b-1}$$

Domain *a*: 1e-323 to 8e+307

Domain *b*: 1e-323 to 8e+307

Domain *x*: -8e+307 to 8e+307; interesting domain is 0 ≤ *x* ≤ 1

Range: 0 to 8e+307

ibeta(a,b,x)

Description: the cumulative beta distribution with shape parameters a and b ; 0 if $x < 0$; or 1 if $x > 1$

The cumulative beta distribution with shape parameters a and b is defined by

$$I_x(a,b) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} \int_0^x t^{a-1}(1-t)^{b-1} dt$$

ibeta() returns the regularized incomplete beta function, also known as the incomplete beta function ratio. The incomplete beta function without regularization is given by $(\text{gamma}(a)*\text{gamma}(b)/\text{gamma}(a+b))*\text{ibeta}(a,b,x)$ or, better when a or b might be large, $\exp(\text{lngamma}(a)+\text{lngamma}(b)-\text{lngamma}(a+b))*\text{ibeta}(a,b,x)$.

Here is an example of the use of the regularized incomplete beta function. Although Stata has a cumulative binomial function (see **binomial()**), the probability that an event occurs k or fewer times in n trials, when the probability of one event is p , can be evaluated as `cond(k==n,1,1-ibeta(k+1,n-k,p))`. The reverse cumulative binomial (the probability that an event occurs k or more times) can be evaluated as `cond(k==0,1,ibeta(k,n-k+1,p))`. See Press et al. (2007, 270–273) for a more complete description and for suggested uses for this function.

Domain a : 1e–10 to 1e+17

Domain b : 1e–10 to 1e+17

Domain x : –8e+307 to 8e+307; interesting domain is $0 \leq x \leq 1$

Range: 0 to 1

ibetatail(a,b,x)

Description: the reverse cumulative (upper tail or survivor) beta distribution with shape parameters a and b ; 1 if $x < 0$; or 0 if $x > 1$

The reverse cumulative (upper tail or survivor) beta distribution with shape parameters a and b is defined by

$$\text{ibetatail}(a,b,x) = 1 - \text{ibeta}(a,b,x) = \int_x^1 \text{betaden}(a,b,t) dt$$

ibetatail() is also known as the complement to the incomplete beta function (ratio).

Domain a : 1e–10 to 1e+17

Domain b : 1e–10 to 1e+17

Domain x : –8e+307 to 8e+307; interesting domain is $0 \leq x \leq 1$

Range: 0 to 1

invibeta(a,b,p)

Description: the inverse cumulative beta distribution: if $\text{ibeta}(a,b,x) = p$, then $\text{invibeta}(a,b,p) = x$

Domain a : 1e–10 to 1e+17

Domain b : 1e–10 to 1e+17

Domain p : 0 to 1

Range: 0 to 1

invibetatail(*a,b,p*)

Description: the inverse reverse cumulative (upper tail or survivor) beta distribution: if $\text{ibetatail}(a,b,x) = p$, then $\text{invibetatail}(a,b,p) = x$

Domain *a*: 1e-10 to 1e+17

Domain *b*: 1e-10 to 1e+17

Domain *p*: 0 to 1

Range: 0 to 1

nbetaden(*a,b,np,x*)

Description: the probability density function of the noncentral beta distribution; 0 if $x < 0$ or $x > 1$

The probability density function of the noncentral beta distribution is defined as

$$\sum_{j=0}^{\infty} \frac{e^{-np/2} (np/2)^j}{\Gamma(j+1)} \left\{ \frac{\Gamma(a+b+j)}{\Gamma(a+j)\Gamma(b)} x^{a+j-1} (1-x)^{b-1} \right\}$$

where *a* and *b* are shape parameters, *np* is the noncentrality parameter, and *x* is the value of a beta random variable.

$\text{nbetaden}(a,b,0,x) = \text{betaden}(a,b,x)$, but **betaden()** is the preferred function to use for the central beta distribution. **nbetaden()** is computed using an algorithm described in [Johnson, Kotz, and Balakrishnan \(1995\)](#).

Domain *a*: 1e-323 to 8e+307

Domain *b*: 1e-323 to 8e+307

Domain *np*: 0 to 1,000

Domain *x*: -8e+307 to 8e+307; interesting domain is $0 \leq x \leq 1$

Range: 0 to 8e+307

nibeta(*a,b,np,x*)

Description: the cumulative noncentral beta distribution; 0 if $x < 0$; or 1 if $x > 1$

The cumulative noncentral beta distribution is defined as

$$I_x(a,b,np) = \sum_{j=0}^{\infty} \frac{e^{-np/2} (np/2)^j}{\Gamma(j+1)} I_x(a+j,b)$$

where *a* and *b* are shape parameters, *np* is the noncentrality parameter, *x* is the value of a beta random variable, and $I_x(a,b)$ is the cumulative beta distribution, **ibeta()**.

$\text{nibeta}(a,b,0,x) = \text{ibeta}(a,b,x)$, but **ibeta()** is the preferred function to use for the central beta distribution. **nibeta()** is computed using an algorithm described in [Johnson, Kotz, and Balakrishnan \(1995\)](#).

Domain *a*: 1e-323 to 8e+307

Domain *b*: 1e-323 to 8e+307

Domain *np*: 0 to 10,000

Domain *x*: -8e+307 to 8e+307; interesting domain is $0 \leq x \leq 1$

Range: 0 to 1

`invnibeta(a,b,np,p)`

Description: the inverse cumulative noncentral beta distribution: if
 $\text{nibeta}(a,b,np,x) = p$, then $\text{invnibeta}(a,b,np,p) = x$
 Domain *a*: 1e-323 to 8e+307
 Domain *b*: 1e-323 to 8e+307
 Domain *np*: 0 to 1,000
 Domain *p*: 0 to 1
 Range: 0 to 1

Binomial distribution

`binomialp(n,k,p)`

Description: the probability of observing `floor(k)` successes in `floor(n)` trials when the probability of a success on one trial is *p*
 Domain *n*: 1 to 1e+6
 Domain *k*: 0 to *n*
 Domain *p*: 0 to 1
 Range: 0 to 1

`binomial(n,k,θ)`

Description: the probability of observing `floor(k)` or fewer successes in `floor(n)` trials when the probability of a success on one trial is *θ*; 0 if *k* < 0; or 1 if *k* > *n*
 Domain *n*: 0 to 1e+17
 Domain *k*: -8e+307 to 8e+307; interesting domain is $0 \leq k < n$
 Domain *θ*: 0 to 1
 Range: 0 to 1

`binomialtail(n,k,θ)`

Description: the probability of observing `floor(k)` or more successes in `floor(n)` trials when the probability of a success on one trial is *θ*; 1 if *k* < 0; or 0 if *k* > *n*
 Domain *n*: 0 to 1e+17
 Domain *k*: -8e+307 to 8e+307; interesting domain is $0 \leq k < n$
 Domain *θ*: 0 to 1
 Range: 0 to 1

`invbinomial(n,k,p)`

Description: the inverse of the cumulative binomial; that is, *θ* (*θ* = probability of success on one trial) such that the probability of observing `floor(k)` or fewer successes in `floor(n)` trials is *p*
 Domain *n*: 1 to 1e+17
 Domain *k*: 0 to *n*-1
 Domain *p*: 0 to 1 (exclusive)
 Range: 0 to 1

`invbinomialtail(n, k, p)`

Description: the inverse of the right cumulative binomial; that is, θ (θ = probability of success on one trial) such that the probability of observing `floor(k)` or more successes in `floor(n)` trials is p

Domain n : 1 to $1\text{e}+17$

Domain k : 1 to n

Domain p : 0 to 1 (exclusive)

Range: 0 to 1

Cauchy distribution

`cauchyden(a, b, x)`

Description: the probability density of the Cauchy distribution with location parameter a and scale parameter b

Domain a : $-1\text{e}+300$ to $1\text{e}+300$

Domain b : $1\text{e}-100$ to $1\text{e}+300$

Domain x : $-8\text{e}+307$ to $8\text{e}+307$

Range: 0 to $8\text{e}+307$

`cauchy(a, b, x)`

Description: the cumulative Cauchy distribution with location parameter a and scale parameter b

Domain a : $-1\text{e}+300$ to $1\text{e}+300$

Domain b : $1\text{e}-100$ to $1\text{e}+300$

Domain x : $-8\text{e}+307$ to $8\text{e}+307$

Range: 0 to 1

`cauchytail(a, b, x)`

Description: the reverse cumulative (upper tail or survivor) Cauchy distribution with location parameter a and scale parameter b

`cauchytail(a, b, x) = 1 - cauchy(a, b, x)`

Domain a : $-1\text{e}+300$ to $1\text{e}+300$

Domain b : $1\text{e}-100$ to $1\text{e}+300$

Domain x : $-8\text{e}+307$ to $8\text{e}+307$

Range: 0 to 1

`invcauchy(a, b, p)`

Description: the inverse of `cauchy()`: if `cauchy(a, b, x) = p` , then

`invcauchy(a, b, p) = x`

Domain a : $-1\text{e}+300$ to $1\text{e}+300$

Domain b : $1\text{e}-100$ to $1\text{e}+300$

Domain p : 0 to 1 (exclusive)

Range: $-8\text{e}+307$ to $8\text{e}+307$

`invcauchytail(a,b,p)`

Description: the inverse of `cauchytail()`: if `cauchytail(a,b,x) = p`, then
`invcauchytail(a,b,p) = x`

Domain *a*: $-1\text{e}+300$ to $1\text{e}+300$

Domain *b*: $1\text{e}-100$ to $1\text{e}+300$

Domain *p*: 0 to 1 (exclusive)

Range: $-8\text{e}+307$ to $8\text{e}+307$

`lncauchyden(a,b,x)`

Description: the natural logarithm of the density of the Cauchy distribution with location parameter *a* and scale parameter *b*

Domain *a*: $-1\text{e}+300$ to $1\text{e}+300$

Domain *b*: $1\text{e}-100$ to $1\text{e}+300$

Domain *x*: $-8\text{e}+307$ to $8\text{e}+307$

Range: -1650 to 230

Augustin-Louis Cauchy (1789–1857) was born in Paris, France. He obtained a degree in engineering with honors from École Polytechnique, where he would later teach mathematics. While working as a military engineer, he published two papers on polyhedra, one of which was a solution to a problem presented to him by Joseph-Louis Lagrange. In 1816, he won the Grand Prix for his work on wave propagation.

Cauchy's contributions were numerous and far reaching, as evident by the many concepts and theorems named after him. Some examples include the Cauchy criterion for convergence, Cauchy's theorem for finite groups, the Cauchy distribution, and the Cauchy stress tensor. His contributions were so vast that once all of his work was collected, it comprised 27 volumes. His name is engraved on the Eiffel Tower, along with 71 other scientists and mathematicians.

Chi-squared and noncentral chi-squared distributions

`chi2den(df,x)`

Description: the probability density of the chi-squared distribution with *df* degrees of freedom; 0 if $x < 0$

`chi2den(df,x) = gammaden(df/2,2,0,x)`

Domain *df*: $2\text{e}-10$ to $2\text{e}+17$ (may be nonintegral)

Domain *x*: $-8\text{e}+307$ to $8\text{e}+307$

Range: 0 to $8\text{e}+307$

`chi2(df,x)`

Description: the cumulative χ^2 distribution with *df* degrees of freedom; 0 if $x < 0$

`chi2(df,x) = gammap(df/2,x/2)`

Domain *df*: $2\text{e}-10$ to $2\text{e}+17$ (may be nonintegral)

Domain *x*: $-8\text{e}+307$ to $8\text{e}+307$; interesting domain is $x \geq 0$

Range: 0 to 1

chi2tail(df,x)

Description: the reverse cumulative (upper tail or survivor) χ^2 distribution with df degrees of freedom; 1 if $x < 0$

$$\text{chi2tail}(df, x) = 1 - \text{chi2}(df, x)$$

Domain df : $2e-10$ to $2e+17$ (may be nonintegral)

Domain x : $-8e+307$ to $8e+307$; interesting domain is $x \geq 0$

Range: 0 to 1

invchi2(df,p)

Description: the inverse of **chi2()**: if $\text{chi2}(df, x) = p$, then $\text{invchi2}(df, p) = x$

Domain df : $2e-10$ to $2e+17$ (may be nonintegral)

Domain p : 0 to 1

Range: 0 to $8e+307$

invchi2tail(df,p)

Description: the inverse of **chi2tail()**: if $\text{chi2tail}(df, x) = p$, then $\text{invchi2tail}(df, p) = x$

Domain df : $2e-10$ to $2e+17$ (may be nonintegral)

Domain p : 0 to 1

Range: 0 to $8e+307$

nchi2den(df,np,x)

Description: the probability density of the noncentral χ^2 distribution; 0 if $x < 0$

df denotes the degrees of freedom, np is the noncentrality parameter, and x is the value of χ^2 .

$\text{nchi2den}(df, 0, x) = \text{chi2den}(df, x)$, but **chi2den()** is the preferred function to use for the central χ^2 distribution.

Domain df : $2e-10$ to $1e+6$ (may be nonintegral)

Domain np : 0 to 10,000

Domain x : $-8e+307$ to $8e+307$

Range: 0 to $8e+307$

nchi2(df,np,x)

Description: the cumulative noncentral χ^2 distribution; 0 if $x < 0$

The cumulative noncentral χ^2 distribution is defined as

$$\int_0^x \frac{e^{-t/2} e^{-np/2}}{2^{df/2}} \sum_{j=0}^{\infty} \frac{t^{df/2+j-1} np^j}{\Gamma(df/2+j) 2^{2j} j!} dt$$

where df denotes the degrees of freedom, np is the noncentrality parameter, and x is the value of χ^2 .

$\text{nchi2}(df, 0, x) = \text{chi2}(df, x)$, but **chi2()** is the preferred function to use for the central χ^2 distribution.

Domain df : $2e-10$ to $1e+6$ (may be nonintegral)

Domain np : 0 to 10,000

Domain x : $-8e+307$ to $8e+307$; interesting domain is $x \geq 0$

Range: 0 to 1

`nchi2tail(df, np, x)`

Description: the reverse cumulative (upper tail or survivor) noncentral χ^2 distribution; 1 if $x < 0$
 df denotes the degrees of freedom, np is the noncentrality parameter, and x is the value of χ^2 .

Domain df : $2e-10$ to $1e+6$ (may be nonintegral)

Domain np : 0 to 10,000

Domain x : $-8e+307$ to $8e+307$

Range: 0 to 1

`invnchi2(df, np, p)`

Description: the inverse cumulative noncentral χ^2 distribution: if
 $nchi2(df, np, x) = p$, then $invnchi2(df, np, p) = x$

Domain df : $2e-10$ to $1e+6$ (may be nonintegral)

Domain np : 0 to 10,000

Domain p : 0 to 1

Range: 0 to $8e+307$

`invnchi2tail(df, np, p)`

Description: the inverse reverse cumulative (upper tail or survivor) noncentral χ^2 distribution: if
 $nchi2tail(df, np, x) = p$, then $invnchi2tail(df, np, p) = x$

Domain df : $2e-10$ to $1e+6$ (may be nonintegral)

Domain np : 0 to 10,000

Domain p : 0 to 1

Range: 0 to $8e+307$

`npnchi2(df, x, p)`

Description: the noncentrality parameter, np , for noncentral χ^2 : if
 $nchi2(df, np, x) = p$, then $npnchi2(df, x, p) = np$

Domain df : $2e-10$ to $1e+6$ (may be nonintegral)

Domain x : 0 to $8e+307$

Domain p : 0 to 1

Range: 0 to 10,000

Dunnett's multiple range distribution

`dunnettprob(k, df, x)`

Description: the cumulative multiple range distribution that is used in Dunnett's multiple-comparison method with k ranges and df degrees of freedom; 0 if $x < 0$

`dunnettprob()` is computed using an algorithm described in [Miller \(1981\)](#).

Domain k : 2 to $1e+6$

Domain df : 2 to $1e+6$

Domain x : $-8e+307$ to $8e+307$; interesting domain is $x \geq 0$

Range: 0 to 1

`invdunnettprob(k, df, p)`

Description: the inverse cumulative multiple range distribution that is used in Dunnett's multiple-comparison method with k ranges and df degrees of freedom

If `dunnettprob(k, df, x) = p`, then `invdunnettprob(k, df, p) = x`.

`invdunnettprob()` is computed using an algorithm described in [Miller \(1981\)](#).

Domain k : 2 to 1e+6

Domain df : 2 to 1e+6

Domain p : 0 to 1 (right exclusive)

Range: 0 to 8e+307

Charles William Dunnett (1921–2007) was a Canadian statistician best known for his work on multiple-comparison procedures. He was born in Windsor, Ontario, and graduated in mathematics and physics from McMaster University. After naval service in World War II, Dunnett's career included further graduate work, teaching, and research at Toronto, Columbia, the New York State Maritime College, the Department of National Health and Welfare in Ottawa, Cornell, Lederle Laboratories, and Aberdeen before he became Professor of Clinical Epidemiology and Biostatistics at McMaster University in 1974. He was President and Gold Medalist of the Statistical Society of Canada. Throughout his career, Dunnett took a keen interest in computing. According to Google Scholar, his 1955 paper on comparing treatments with a control has been cited over 4,000 times.

Exponential distribution

`exponentialden(b, x)`

Description: the probability density function of the exponential distribution with scale b

The probability density function of the exponential distribution is

$$\frac{1}{b} \exp(-x/b)$$

where b is the scale and x is the value of an exponential variate.

Domain b : 1e-323 to 8e+307

Domain x : -8e+307 to 8e+307; interesting domain is $x \geq 0$

Range: 1e-323 to 8e+307

`exponential(b, x)`

Description: the cumulative exponential distribution with scale b

The cumulative distribution function of the exponential distribution is

$$1 - \exp(-x/b)$$

for $x \geq 0$ and 0 for $x < 0$, where b is the scale and x is the value of an exponential variate.

The mean of the exponential distribution is b and its variance is b^2 .

Domain b : 1e-323 to 8e+307

Domain x : -8e+307 to 8e+307; interesting domain is $x \geq 0$

Range: 0 to 1

exponentialtail(b, x)

Description: the reverse cumulative exponential distribution with scale b

The reverse cumulative distribution function of the exponential distribution is

$$\exp(-x/b)$$

where b is the scale and x is the value of an exponential variate.

Domain b : 1e-323 to 8e+307

Domain x : -8e+307 to 8e+307; interesting domain is $x \geq 0$

Range: 0 to 1

invexponential(b, p)

Description: the inverse cumulative exponential distribution with scale b : if

$\text{exponential}(b, x) = p$, then $\text{invexponential}(b, p) = x$

Domain b : 1e-323 to 8e+307

Domain p : 0 to 1

Range: 1e-323 to 8e+307

invexponentialtail(b, p)

Description: the inverse reverse cumulative exponential distribution with scale b :

if $\text{exponentialtail}(b, x) = p$, then

$\text{invexponentialtail}(b, p) = x$

Domain b : 1e-323 to 8e+307

Domain p : 0 to 1

Range: 1e-323 to 8e+307

F and noncentral F distributions**Fden(df_1, df_2, f)**

Description: the probability density function of the F distribution with df_1 numerator and df_2 denominator degrees of freedom; 0 if $f < 0$

The probability density function of the F distribution with df_1 numerator and df_2 denominator degrees of freedom is defined as

$$\text{Fden}(df_1, df_2, f) = \frac{\Gamma(\frac{df_1 + df_2}{2})}{\Gamma(\frac{df_1}{2})\Gamma(\frac{df_2}{2})} \left(\frac{df_1}{df_2}\right)^{\frac{df_1}{2}} \cdot f^{\frac{df_1}{2} - 1} \left(1 + \frac{df_1}{df_2} f\right)^{-\frac{1}{2}(df_1 + df_2)}$$

Domain df_1 : 1e-323 to 8e+307 (may be nonintegral)

Domain df_2 : 1e-323 to 8e+307 (may be nonintegral)

Domain f : -8e+307 to 8e+307; interesting domain is $f \geq 0$

Range: 0 to 8e+307

F(df₁, df₂, f)

Description: the cumulative F distribution with df_1 numerator and df_2 denominator degrees of freedom: $F(df_1, df_2, f) = \int_0^f Fden(df_1, df_2, t) dt$; 0 if $f < 0$

Domain df_1 : 2e-10 to 2e+17 (may be nonintegral)

Domain df_2 : 2e-10 to 2e+17 (may be nonintegral)

Domain f : -8e+307 to 8e+307; interesting domain is $f \geq 0$

Range: 0 to 1

Ftail(df₁, df₂, f)

Description: the reverse cumulative (upper tail or survivor) F distribution with df_1 numerator and df_2 denominator degrees of freedom; 1 if $f < 0$

$$Ftail(df_1, df_2, f) = 1 - F(df_1, df_2, f).$$

Domain df_1 : 2e-10 to 2e+17 (may be nonintegral)

Domain df_2 : 2e-10 to 2e+17 (may be nonintegral)

Domain f : -8e+307 to 8e+307; interesting domain is $f \geq 0$

Range: 0 to 1

invF(df₁, df₂, p)

Description: the inverse cumulative F distribution: if $F(df_1, df_2, f) = p$, then

$$invF(df_1, df_2, p) = f$$

Domain df_1 : 2e-10 to 2e+17 (may be nonintegral)

Domain df_2 : 2e-10 to 2e+17 (may be nonintegral)

Domain p : 0 to 1

Range: 0 to 8e+307

invFtail(df₁, df₂, p)

Description: the inverse reverse cumulative (upper tail or survivor) F distribution: if $Ftail(df_1, df_2, f) = p$, then $invFtail(df_1, df_2, p) = f$

Domain df_1 : 2e-10 to 2e+17 (may be nonintegral)

Domain df_2 : 2e-10 to 2e+17 (may be nonintegral)

Domain p : 0 to 1

Range: 0 to 8e+307

nFden(df_1, df_2, np, f)

Description: the probability density function of the noncentral F distribution with df_1 numerator and df_2 denominator degrees of freedom and noncentrality parameter np ; 0 if $f < 0$

nFden($df_1, df_2, 0, f$) = **Fden**(df_1, df_2, f), but **Fden**() is the preferred function to use for the central F distribution.

Also, if F follows the noncentral F distribution with df_1 and df_2 degrees of freedom and noncentrality parameter np , then

$$\frac{df_1 F}{df_2 + df_1 F}$$

follows a noncentral beta distribution with shape parameters $a = df_1/2$, $b = df_2/2$, and noncentrality parameter np , as given in **nbetaden**(). **nFden**() is computed based on this relationship.

Domain df_1 : 1e-323 to 8e+307 (may be nonintegral)

Domain df_2 : 1e-323 to 8e+307 (may be nonintegral)

Domain np : 0 to 1,000

Domain f : -8e+307 to 8e+307; interesting domain is $f \geq 0$

Range: 0 to 8e+307

nF(df_1, df_2, np, f)

Description: the cumulative noncentral F distribution with df_1 numerator and df_2 denominator degrees of freedom and noncentrality parameter np ; 0 if $f < 0$

nF($df_1, df_2, 0, f$) = **F**(df_1, df_2, f)

nF() is computed using **nibeta**() based on the relationship between the noncentral beta and noncentral F distributions: **nF**(df_1, df_2, np, f) = **nibeta**($df_1/2, df_2/2, np, df_1 \times f / \{(df_1 \times f) + df_2\}$).

Domain df_1 : 2e-10 to 1e+8

Domain df_2 : 2e-10 to 1e+8

Domain np : 0 to 10,000

Domain f : -8e+307 to 8e+307

Range: 0 to 1

nFtail(df_1, df_2, np, f)

Description: the reverse cumulative (upper tail or survivor) noncentral F distribution with df_1 numerator and df_2 denominator degrees of freedom and noncentrality parameter np ; 1 if $f < 0$

nFtail() is computed using **nibeta**() based on the relationship between the noncentral beta and F distributions. See [Johnson, Kotz, and Balakrishnan \(1995\)](#) for more details.

Domain df_1 : 1e-323 to 8e+307 (may be nonintegral)

Domain df_2 : 1e-323 to 8e+307 (may be nonintegral)

Domain np : 0 to 1,000

Domain f : -8e+307 to 8e+307; interesting domain is $f \geq 0$

Range: 0 to 1

invnF(df_1, df_2, np, p)Description: the inverse cumulative noncentral F distribution: if $nF(df_1, df_2, np, f) = p$, then $invnF(df_1, df_2, np, p) = f$ Domain df_1 : 1e-6 to 1e+6 (may be nonintegral)Domain df_2 : 1e-6 to 1e+6 (may be nonintegral)Domain np : 0 to 10,000Domain p : 0 to 1

Range: 0 to 8e+307

invnFtail(df_1, df_2, np, p)Description: the inverse reverse cumulative (upper tail or survivor) noncentral F distribution: if $nFtail(df_1, df_2, np, f) = p$, then $invnFtail(df_1, df_2, np, p) = f$ Domain df_1 : 1e-323 to 8e+307 (may be nonintegral)Domain df_2 : 1e-323 to 8e+307 (may be nonintegral)Domain np : 0 to 1,000Domain p : 0 to 1

Range: 0 to 8e+307

npnF(df_1, df_2, f, p)Description: the noncentrality parameter, np , for the noncentral F : if $nF(df_1, df_2, np, f) = p$, then $npnF(df_1, df_2, f, p) = np$ Domain df_1 : 2e-10 to 1e+6 (may be nonintegral)Domain df_2 : 2e-10 to 1e+6 (may be nonintegral)Domain f : 0 to 8e+307Domain p : 0 to 1

Range: 0 to 1,000

Gamma distribution

gammaden(a, b, g, x)Description: the probability density function of the gamma distribution; 0 if $x < g$

The probability density function of the gamma distribution is defined by

$$\frac{1}{\Gamma(a)b^a}(x-g)^{a-1}e^{-(x-g)/b}$$

where a is the shape parameter, b is the scale parameter, and g is the location parameter.Domain a : 1e-323 to 8e+307Domain b : 1e-323 to 8e+307Domain g : -8e+307 to 8e+307Domain x : -8e+307 to 8e+307; interesting domain is $x \geq g$

Range: 0 to 8e+307

gammap(*a*,*x*)

Description: the cumulative gamma distribution with shape parameter *a*; 0 if $x < 0$

The cumulative gamma distribution with shape parameter *a* is defined by

$$\frac{1}{\Gamma(a)} \int_0^x e^{-t} t^{a-1} dt$$

The cumulative Poisson (the probability of observing *k* or fewer events if the expected is *x*) can be evaluated as `1-gammap(k+1,x)`. The reverse cumulative (the probability of observing *k* or more events) can be evaluated as `gammap(k,x)`. See [Press et al. \(2007, 259–266\)](#) for a more complete description and for suggested uses for this function.

`gammap()` is also known as the incomplete gamma function (ratio).

Probabilities for the three-parameter gamma distribution (see `gammaden()`) can be calculated by shifting and scaling *x*; that is, `gammap(a, (x - g)/b)`.

Domain *a*: 1e-10 to 1e+17

Domain *x*: -8e+307 to 8e+307; interesting domain is $x \geq 0$

Range: 0 to 1

gammaptail(*a*,*x*)

Description: the reverse cumulative (upper tail or survivor) gamma distribution with shape parameter *a*; 1 if $x < 0$

The reverse cumulative (upper tail or survivor) gamma distribution with shape parameter *a* is defined by

$$\text{gammaptail}(a, x) = 1 - \text{gammap}(a, x) = \int_x^\infty \text{gammaden}(a, t) dt$$

`gammaptail()` is also known as the complement to the incomplete gamma function (ratio).

Domain *a*: 1e-10 to 1e+17

Domain *x*: -8e+307 to 8e+307; interesting domain is $x \geq 0$

Range: 0 to 1

invgammap(*a*,*p*)

Description: the inverse cumulative gamma distribution: if `gammap(a,x) = p`, then `invgammap(a,p) = x`

Domain *a*: 1e-10 to 1e+17

Domain *p*: 0 to 1

Range: 0 to 8e+307

invgammaptail(*a*,*p*)

Description: the inverse reverse cumulative (upper tail or survivor) gamma distribution: if $\text{gammaptail}(a, x) = p$, then $\text{invgammaptail}(a, p) = x$
Domain *a*: 1e-10 to 1e+17
Domain *p*: 0 to 1
Range: 0 to 8e+307

dgammapda(*a*,*x*)

Description: $\frac{\partial P(a, x)}{\partial a}$, where $P(a, x) = \text{gammap}(a, x)$; 0 if $x < 0$
Domain *a*: 1e-7 to 1e+17
Domain *x*: -8e+307 to 8e+307; interesting domain is $x \geq 0$
Range: -16 to 0

dgammapdada(*a*,*x*)

Description: $\frac{\partial^2 P(a, x)}{\partial a^2}$, where $P(a, x) = \text{gammap}(a, x)$; 0 if $x < 0$
Domain *a*: 1e-7 to 1e+17
Domain *x*: -8e+307 to 8e+307; interesting domain is $x \geq 0$
Range: -0.02 to 4.77e+5

dgammapdadx(*a*,*x*)

Description: $\frac{\partial^2 P(a, x)}{\partial a \partial x}$, where $P(a, x) = \text{gammap}(a, x)$; 0 if $x < 0$
Domain *a*: 1e-7 to 1e+17
Domain *x*: -8e+307 to 8e+307; interesting domain is $x \geq 0$
Range: -0.04 to 8e+307

dgammapdx(*a*,*x*)

Description: $\frac{\partial P(a, x)}{\partial x}$, where $P(a, x) = \text{gammap}(a, x)$; 0 if $x < 0$
Domain *a*: 1e-10 to 1e+17
Domain *x*: -8e+307 to 8e+307; interesting domain is $x \geq 0$
Range: 0 to 8e+307

dgammapdxdx(*a*,*x*)

Description: $\frac{\partial^2 P(a, x)}{\partial x^2}$, where $P(a, x) = \text{gammap}(a, x)$; 0 if $x < 0$
Domain *a*: 1e-10 to 1e+17
Domain *x*: -8e+307 to 8e+307; interesting domain is $x \geq 0$
Range: 0 to 1e+40

lnigammaden(*a*,*b*,*x*)

Description: the natural logarithm of the inverse gamma density, where *a* is the shape parameter and *b* is the scale parameter
Domain *a*: 1e-300 to 1e+300
Domain *b*: 1e-300 to 1e+300
Domain *x*: -8e+307 to 8e+307
Range: 1e-300 to 8e+307

Hypergeometric distribution

`hypergeometricp(N, K, n, k)`

Description: the hypergeometric probability of k successes out of a sample of size n , from a population of size N containing K elements that have the attribute of interest

Success is obtaining an element with the attribute of interest.

Domain N : 2 to $1e+5$

Domain K : 1 to $N-1$

Domain n : 1 to $N-1$

Domain k : $\max(0, n - N + K)$ to $\min(K, n)$

Range: 0 to 1 (right exclusive)

`hypergeometric(N, K, n, k)`

Description: the cumulative probability of the hypergeometric distribution

N is the population size, K is the number of elements in the population that have the attribute of interest, and n is the sample size. Returned is the probability of observing k or fewer elements from a sample of size n that have the attribute of interest.

Domain N : 2 to $1e+5$

Domain K : 1 to $N-1$

Domain n : 1 to $N-1$

Domain k : $\max(0, n - N + K)$ to $\min(K, n)$

Range: 0 to 1

Inverse Gaussian distribution

`igaussianden(m, a, x)`

Description: the probability density of the inverse Gaussian distribution with mean m and shape parameter a ; 0 if $x \leq 0$

Domain m : $1e-323$ to $8e+307$

Domain a : $1e-323$ to $8e+307$

Domain x : $-8e+307$ to $8e+307$

Range: 0 to $8e+307$

`igaussian(m, a, x)`

Description: the cumulative inverse Gaussian distribution with mean m and shape parameter a ; 0 if $x \leq 0$

Domain m : $1e-323$ to $8e+307$

Domain a : $1e-323$ to $8e+307$

Domain x : $-8e+307$ to $8e+307$

Range: 0 to 1

`igausstail(m, a, x)`

Description: the reverse cumulative (upper tail or survivor) inverse Gaussian distribution with mean m and shape parameter a ; 1 if $x \leq 0$

$\text{igausstail}(m, a, x) = 1 - \text{igaussian}(m, a, x)$

Domain m : $1e-323$ to $8e+307$

Domain a : $1e-323$ to $8e+307$

Domain x : $-8e+307$ to $8e+307$

Range: 0 to 1

invgaussian(*m,a,p*)

Description: the inverse of **igaussian()**: if
 $\text{igaussian}(m,a,x) = p$, then $\text{invgaussian}(m,a,p) = x$
Domain *m*: 1e-323 to 8e+307
Domain *a*: 1e-323 to 1e+8
Domain *p*: 0 to 1 (exclusive)
Range: 0 to 8e+307

invgaussiantail(*m,a,p*)

Description: the inverse of **igaussiantail()**: if
 $\text{igaussiantail}(m,a,x) = p$, then
 $\text{invgaussiantail}(m,a,p) = x$
Domain *m*: 1e-323 to 8e+307
Domain *a*: 1e-323 to 1e+8
Domain *p*: 0 to 1 (exclusive)
Range: 0 to 1

lnigaussianden(*m,a,x*)

Description: the natural logarithm of the inverse Gaussian density with mean *m* and shape parameter *a*
Domain *m*: 1e-323 to 8e+307
Domain *a*: 1e-323 to 8e+307
Domain *x*: 1e-323 to 8e+307
Range: -8e+307 to 8e+307

Laplace distribution

laplaceden(*m,b,x*)

Description: the probability density of the Laplace distribution with mean *m* and scale parameter *b*
Domain *m*: -8e+307 to 8e+307
Domain *b*: 1e-307 to 8e+307
Domain *x*: -8e+307 to 8e+307
Range: 0 to 8e+307

laplace(*m,b,x*)

Description: the cumulative Laplace distribution with mean *m* and scale parameter *b*
Domain *m*: -8e+307 to 8e+307
Domain *b*: 1e-307 to 8e+307
Domain *x*: -8e+307 to 8e+307
Range: 0 to 1

laplacetail(*m,b,x*)

Description: the reverse cumulative (upper tail or survivor) Laplace distribution with mean *m* and scale parameter *b*
 $\text{laplacetail}(m,b,x) = 1 - \text{laplace}(m,b,x)$
Domain *m*: -8e+307 to 8e+307
Domain *b*: 1e-307 to 8e+307
Domain *x*: -8e+307 to 8e+307
Range: 0 to 1

invlaplace(m, b, p)

Description: the inverse of `laplace()`: if `laplace(m, b, x) = p` , then
`invlaplace(m, b, p) = x`

Domain m : $-8\text{e}+307$ to $8\text{e}+307$

Domain b : $1\text{e}-307$ to $8\text{e}+307$

Domain p : 0 to 1 (exclusive)

Range: $-8\text{e}+307$ to $8\text{e}+307$

invlaplacetail(m, b, p)

Description: the inverse of `laplacetail()`: if `laplacetail(m, b, x) = p` ,
then `invlaplacetail(m, b, p) = x`

Domain m : $-8\text{e}+307$ to $8\text{e}+307$

Domain b : $1\text{e}-307$ to $8\text{e}+307$

Domain p : 0 to 1 (exclusive)

Range: $-8\text{e}+307$ to $8\text{e}+307$

lnlaplaceden(m, b, x)

Description: the natural logarithm of the density of the Laplace distribution with mean m and
scale parameter b

Domain m : $-8\text{e}+307$ to $8\text{e}+307$

Domain b : $1\text{e}-307$ to $8\text{e}+307$

Domain x : $-8\text{e}+307$ to $8\text{e}+307$

Range: $-8\text{e}+307$ to 707

Logistic distribution**logisticden(x)**

Description: the density of the logistic distribution with mean 0 and standard deviation $\pi/\sqrt{3}$

`logisticden(x) = logisticden(1, x) = logisticden(0,1, x)`, where x is
the value of a logistic random variable.

Domain x : $-8\text{e}+307$ to $8\text{e}+307$

Range: 0 to 0.25

logisticden(s, x)

Description: the density of the logistic distribution with mean 0, scale s , and standard deviation
 $s\pi/\sqrt{3}$

`logisticden(s, x) = logisticden(0, s, x)`, where s is the scale and x is the
value of a logistic random variable.

Domain s : $1\text{e}-323$ to $8\text{e}+307$

Domain x : $-8\text{e}+307$ to $8\text{e}+307$

Range: 0 to $8\text{e}+307$

logisticden(*m,s,x*)

Description: the density of the logistic distribution with mean *m*, scale *s*, and standard deviation $s\pi/\sqrt{3}$

The density of the logistic distribution is defined as

$$\frac{\exp\{-(x-m)/s\}}{s[1 + \exp\{-(x-m)/s\}]^2}$$

where *m* is the mean, *s* is the scale, and *x* is the value of a logistic random variable.

Domain *m*: $-8\text{e}+307$ to $8\text{e}+307$

Domain *s*: $1\text{e}-323$ to $8\text{e}+307$

Domain *x*: $-8\text{e}+307$ to $8\text{e}+307$

Range: 0 to $8\text{e}+307$

logistic(*x*)

Description: the cumulative logistic distribution with mean 0 and standard deviation $\pi/\sqrt{3}$

$\text{logistic}(x) = \text{logistic}(1,x) = \text{logistic}(0,1,x)$, where *x* is the value of a logistic random variable.

Domain *x*: $-8\text{e}+307$ to $8\text{e}+307$

Range: 0 to 1

logistic(*s,x*)

Description: the cumulative logistic distribution with mean 0, scale *s*, and standard deviation $s\pi/\sqrt{3}$

$\text{logistic}(s, x) = \text{logistic}(0,s,x)$, where *s* is the scale and *x* is the value of a logistic random variable.

Domain *s*: $1\text{e}-323$ to $8\text{e}+307$

Domain *x*: $-8\text{e}+307$ to $8\text{e}+307$

Range: 0 to 1

logistic(*m,s,x*)

Description: the cumulative logistic distribution with mean *m*, scale *s*, and standard deviation $s\pi/\sqrt{3}$

The cumulative logistic distribution is defined as

$$[1 + \exp\{-(x-m)/s\}]^{-1}$$

where *m* is the mean, *s* is the scale, and *x* is the value of a logistic random variable.

Domain *m*: $-8\text{e}+307$ to $8\text{e}+307$

Domain *s*: $1\text{e}-323$ to $8\text{e}+307$

Domain *x*: $-8\text{e}+307$ to $8\text{e}+307$

Range: 0 to 1

logistictail(x)

Description: the reverse cumulative logistic distribution with mean 0 and standard deviation $\pi/\sqrt{3}$

$\text{logistictail}(x) = \text{logistictail}(1,x) = \text{logistictail}(0,1,x)$, where x is the value of a logistic random variable.

Domain x : $-8\text{e}+307$ to $8\text{e}+307$

Range: 0 to 1

logistictail(s,x)

Description: the reverse cumulative logistic distribution with mean 0, scale s , and standard deviation $s\pi/\sqrt{3}$

$\text{logistictail}(s,x) = \text{logistictail}(0,s,x)$, where s is the scale and x is the value of a logistic random variable.

Domain s : $1\text{e}-323$ to $8\text{e}+307$

Domain x : $-8\text{e}+307$ to $8\text{e}+307$

Range: 0 to 1

logistictail(m,s,x)

Description: the reverse cumulative logistic distribution with mean m , scale s , and standard deviation $s\pi/\sqrt{3}$

The reverse cumulative logistic distribution is defined as

$$[1 + \exp\{(x - m)/s\}]^{-1}$$

where m is the mean, s is the scale, and x is the value of a logistic random variable.

Domain m : $-8\text{e}+307$ to $8\text{e}+307$

Domain s : $1\text{e}-323$ to $8\text{e}+307$

Domain x : $-8\text{e}+307$ to $8\text{e}+307$

Range: 0 to 1

invlogistic(p)

Description: the inverse cumulative logistic distribution: if $\text{logistic}(x) = p$, then $\text{invlogistic}(p) = x$

Domain p : 0 to 1

Range: $-8\text{e}+307$ to $8\text{e}+307$

invlogistic(s,p)

Description: the inverse cumulative logistic distribution: if $\text{logistic}(s,x) = p$, then $\text{invlogistic}(s,p) = x$

Domain s : $1\text{e}-323$ to $8\text{e}+307$

Domain p : 0 to 1

Range: $-8\text{e}+307$ to $8\text{e}+307$

invlogistic(m, s, p)

Description: the inverse cumulative logistic distribution: if $\text{logistic}(m, s, x) = p$, then $\text{invlogistic}(m, s, p) = x$

Domain m : $-8\text{e}+307$ to $8\text{e}+307$

Domain s : $1\text{e}-323$ to $8\text{e}+307$

Domain p : 0 to 1

Range: $-8\text{e}+307$ to $8\text{e}+307$

invlogistictail(p)

Description: the inverse reverse cumulative logistic distribution: if $\text{logistictail}(x) = p$, then $\text{invlogistictail}(p) = x$

Domain p : 0 to 1

Range: $-8\text{e}+307$ to $8\text{e}+307$

invlogistictail(s, p)

Description: the inverse reverse cumulative logistic distribution: if $\text{logistictail}(s, x) = p$, then $\text{invlogistictail}(s, p) = x$

Domain s : $1\text{e}-323$ to $8\text{e}+307$

Domain p : 0 to 1

Range: $-8\text{e}+307$ to $8\text{e}+307$

invlogistictail(m, s, p)

Description: the inverse reverse cumulative logistic distribution: if $\text{logistictail}(m, s, x) = p$, then $\text{invlogistictail}(m, s, p) = x$

Domain m : $-8\text{e}+307$ to $8\text{e}+307$

Domain s : $1\text{e}-323$ to $8\text{e}+307$

Domain p : 0 to 1

Range: $-8\text{e}+307$ to $8\text{e}+307$

Negative binomial distribution

nbinomialp(n, k, p)

Description: the negative binomial probability

When n is an integer, `nbinomialp()` returns the probability of observing exactly `floor(k)` failures before the n th success when the probability of a success on one trial is p .

Domain n : $1\text{e}-10$ to $1\text{e}+6$ (can be nonintegral)

Domain k : 0 to $1\text{e}+10$

Domain p : 0 to 1 (left exclusive)

Range: 0 to 1

nbinomial(n, k, p)

Description: the cumulative probability of the negative binomial distribution

n can be nonintegral. When n is an integer, **nbinomial()** returns the probability of observing k or fewer failures before the n th success, when the probability of a success on one trial is p .

The negative binomial distribution function is evaluated using **ibeta()**.

Domain n : $1\text{e}-10$ to $1\text{e}+17$ (can be nonintegral)

Domain k : 0 to $2^{53} - 1$

Domain p : 0 to 1 (left exclusive)

Range: 0 to 1

nbinomialtail(n, k, p)

Description: the reverse cumulative probability of the negative binomial distribution

When n is an integer, **nbinomialtail()** returns the probability of observing k or more failures before the n th success, when the probability of a success on one trial is p .

The reverse negative binomial distribution function is evaluated using **ibetatail()**.

Domain n : $1\text{e}-10$ to $1\text{e}+17$ (can be nonintegral)

Domain k : 0 to $2^{53} - 1$

Domain p : 0 to 1 (left exclusive)

Range: 0 to 1

invnbinomial(n, k, q)

Description: the value of the negative binomial parameter, p , such that $q = \text{nbinomial}(n, k, p)$

invnbinomial() is evaluated using **invibeta()**.

Domain n : $1\text{e}-10$ to $1\text{e}+17$ (can be nonintegral)

Domain k : 0 to $2^{53} - 1$

Domain q : 0 to 1 (exclusive)

Range: 0 to 1

invnbinomialtail(n, k, q)

Description: the value of the negative binomial parameter, p , such that $q = \text{nbinomialtail}(n, k, p)$

invnbinomialtail() is evaluated using **invibetatail()**.

Domain n : $1\text{e}-10$ to $1\text{e}+17$ (can be nonintegral)

Domain k : 1 to $2^{53} - 1$

Domain q : 0 to 1 (exclusive)

Range: 0 to 1 (exclusive)

Normal (Gaussian), binormal, and multivariate normal distributions

normalden(z)

Description: the standard normal density, $N(0, 1)$

Domain: $-8\text{e}+307$ to $8\text{e}+307$

Range: 0 to $0.39894 \dots$

normalden(x, σ)

Description: the normal density with mean 0 and standard deviation σ

$$\begin{aligned}\text{normalden}(x, 1) &= \text{normalden}(x) \text{ and} \\ \text{normalden}(x, \sigma) &= \text{normalden}(x/\sigma)/\sigma.\end{aligned}$$

Domain x : $-8\text{e}+307$ to $8\text{e}+307$

Domain σ : $1\text{e}-308$ to $8\text{e}+307$

Range: 0 to $8\text{e}+307$

normalden(x, μ, σ)

Description: the normal density with mean μ and standard deviation σ , $N(\mu, \sigma^2)$

$$\begin{aligned}\text{normalden}(x, 0, s) &= \text{normalden}(x, s) \text{ and} \\ \text{normalden}(x, \mu, \sigma) &= \text{normalden}((x - \mu)/\sigma)/\sigma. \text{ In general,}\end{aligned}$$

$$\text{normalden}(z, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left\{\frac{(z-\mu)}{\sigma}\right\}^2}$$

Domain x : $-8\text{e}+307$ to $8\text{e}+307$

Domain μ : $-8\text{e}+307$ to $8\text{e}+307$

Domain σ : $1\text{e}-308$ to $8\text{e}+307$

Range: 0 to $8\text{e}+307$

normal(z)

Description: the cumulative standard normal distribution

$$\text{normal}(z) = \int_{-\infty}^z \frac{1}{\sqrt{2\pi}} e^{-x^2/2} dx$$

Domain: $-8\text{e}+307$ to $8\text{e}+307$

Range: 0 to 1

invnormal(p)

Description: the inverse cumulative standard normal distribution: if $\text{normal}(z) = p$, then

$$\text{invnormal}(p) = z$$

Domain: $1\text{e}-323$ to $1 - 2^{-53}$

Range: -38.449394 to 8.2095362

lnnormalden(z)

Description: the natural logarithm of the standard normal density, $N(0, 1)$

Domain: $-1\text{e}+154$ to $1\text{e}+154$

Range: $-5\text{e}+307$ to $-0.91893853 = \text{lnnormalden}(0)$

lnnormalden(x, σ)

Description: the natural logarithm of the normal density with mean 0 and standard deviation σ

$$\begin{aligned}\text{lnnormalden}(x, 1) &= \text{lnnormalden}(x) \text{ and} \\ \text{lnnormalden}(x, \sigma) &= \text{lnnormalden}(x/\sigma) - \ln(\sigma).\end{aligned}$$

Domain x : $-8\text{e}+307$ to $8\text{e}+307$

Domain σ : $1\text{e}-323$ to $8\text{e}+307$

Range: $-5\text{e}+307$ to 742.82799

lnnormalden(x, μ, σ)

Description: the natural logarithm of the normal density with mean μ and standard deviation σ , $N(\mu, \sigma^2)$

$\text{lnnormalden}(x, 0, s) = \text{lnnormalden}(x, s)$ and

$\text{lnnormalden}(x, \mu, \sigma) = \text{lnnormalden}((x - \mu)/\sigma) - \ln(\sigma)$. In general,

$$\text{lnnormalden}(z, \mu, \sigma) = \ln \left[\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2} \left\{ \frac{(z-\mu)}{\sigma} \right\}^2} \right]$$

Domain x : $-8\text{e}+307$ to $8\text{e}+307$

Domain μ : $-8\text{e}+307$ to $8\text{e}+307$

Domain σ : $1\text{e}-323$ to $8\text{e}+307$

Range: $1\text{e}-323$ to $8\text{e}+307$

lnnormal(z)

Description: the natural logarithm of the cumulative standard normal distribution

$$\text{lnnormal}(z) = \ln \left(\int_{-\infty}^z \frac{1}{\sqrt{2\pi}} e^{-x^2/2} dx \right)$$

Domain: $-1\text{e}+99$ to $8\text{e}+307$

Range: $-5\text{e}+197$ to 0

binormal(h, k, ρ)

Description: the joint cumulative distribution $\Phi(h, k, \rho)$ of bivariate normal with correlation ρ

Cumulative over $(-\infty, h] \times (-\infty, k]$:

$$\Phi(h, k, \rho) = \frac{1}{2\pi\sqrt{1-\rho^2}} \int_{-\infty}^h \int_{-\infty}^k \exp \left\{ -\frac{1}{2(1-\rho^2)} (x_1^2 - 2\rho x_1 x_2 + x_2^2) \right\} dx_1 dx_2$$

Domain h : $-8\text{e}+307$ to $8\text{e}+307$

Domain k : $-8\text{e}+307$ to $8\text{e}+307$

Domain ρ : -1 to 1

Range: 0 to 1

lnmvnormalden(M, V, X)

Description: the natural logarithm of the multivariate normal density

M is the mean vector, V is the covariance matrix, and X is the random vector.

Domain M : $1 \times n$ and $n \times 1$ vectors

Domain V : $n \times n$, positive-definite, symmetric matrices

Domain X : $1 \times n$ and $n \times 1$ vectors

Range: $-8\text{e}+307$ to $8\text{e}+307$

Poisson distribution

`poissonp(m,k)`

Description: the probability of observing `floor(k)` outcomes that are distributed as Poisson with mean *m*

The Poisson probability function is evaluated using `gammaden()`.

Domain *m*: 1e-10 to 1e+8

Domain *k*: 0 to 1e+9

Range: 0 to 1

`poisson(m,k)`

Description: the probability of observing `floor(k)` or fewer outcomes that are distributed as Poisson with mean *m*

The Poisson distribution function is evaluated using `gammaptail()`.

Domain *m*: 1e-10 to $2^{53} - 1$

Domain *k*: 0 to $2^{53} - 1$

Range: 0 to 1

`poisontail(m,k)`

Description: the probability of observing `floor(k)` or more outcomes that are distributed as Poisson with mean *m*

The reverse cumulative Poisson distribution function is evaluated using `gammap()`.

Domain *m*: 1e-10 to $2^{53} - 1$

Domain *k*: 0 to $2^{53} - 1$

Range: 0 to 1

`invpoisson(k,p)`

Description: the Poisson mean such that the cumulative Poisson distribution evaluated at *k* is *p*: if `poisson(m,k) = p`, then `invpoisson(k,p) = m`

The inverse Poisson distribution function is evaluated using `invgammaptail()`.

Domain *k*: 0 to $2^{53} - 1$

Domain *p*: 0 to 1 (exclusive)

Range: 1.110e-16 to 2^{53}

`invpoisontail(k,q)`

Description: the Poisson mean such that the reverse cumulative Poisson distribution evaluated at *k* is *q*: if `poisontail(m,k) = q`, then `invpoisontail(k,q) = m`

The inverse of the reverse cumulative Poisson distribution function is evaluated using `invgammap()`.

Domain *k*: 0 to $2^{53} - 1$

Domain *q*: 0 to 1 (exclusive)

Range: 0 to 2^{53} (left exclusive)

Student's t and noncentral Student's t distributions

tden(df, t)

Description: the probability density function of Student's t distribution

$$\text{tden}(df, t) = \frac{\Gamma\{(df+1)/2\}}{\sqrt{\pi df} \Gamma(df/2)} \cdot (1 + t^2/df)^{-(df+1)/2}$$

Domain df : 1e-323 to 8e+307 (may be nonintegral)

Domain t : -8e+307 to 8e+307

Range: 0 to 0.39894 ...

t(df, t)

Description: the cumulative Student's t distribution with df degrees of freedom

Domain df : 2e-10 to 2e+17 (may be nonintegral)

Domain t : -8e+307 to 8e+307

Range: 0 to 1

ttail(df, t)

Description: the reverse cumulative (upper tail or survivor) Student's t distribution; the probability $T > t$

$$\text{ttail}(df, t) = \int_t^{\infty} \frac{\Gamma\{(df+1)/2\}}{\sqrt{\pi df} \Gamma(df/2)} \cdot (1 + x^2/df)^{-(df+1)/2} dx$$

Domain df : 2e-10 to 2e+17 (may be nonintegral)

Domain t : -8e+307 to 8e+307

Range: 0 to 1

invnt(df, p)

Description: the inverse cumulative Student's t distribution: if $\text{t}(df, t) = p$, then $\text{invnt}(df, p) = t$

Domain df : 2e-10 to 2e+17 (may be nonintegral)

Domain p : 0 to 1

Range: -8e+307 to 8e+307

invttail(df, p)

Description: the inverse reverse cumulative (upper tail or survivor) Student's t distribution: if $\text{ttail}(df, t) = p$, then $\text{invttail}(df, p) = t$

Domain df : 2e-10 to 2e+17 (may be nonintegral)

Domain p : 0 to 1

Range: -8e+307 to 8e+307

invnnt(df, np, p)

Description: the inverse cumulative noncentral Student's t distribution: if $\text{nt}(df, np, t) = p$, then $\text{invnnt}(df, np, p) = t$

Domain df : 1 to 1e+6 (may be nonintegral)

Domain np : -1,000 to 1,000

Domain p : 0 to 1

Range: -8e+307 to 8e+307

invnttail(*df*,*np*,*p*)

Description: the inverse reverse cumulative (upper tail or survivor) noncentral Student's t distribution: if $\text{nttail}(df, np, t) = p$, then $\text{invnttail}(df, np, p) = t$
Domain df : 1 to $1\text{e}+6$ (may be nonintegral)
Domain np : $-1,000$ to $1,000$
Domain p : 0 to 1
Range: $-8\text{e}+10$ to $8\text{e}+10$

ntden(*df*,*np*,*t*)

Description: the probability density function of the noncentral Student's t distribution with df degrees of freedom and noncentrality parameter np
Domain df : $1\text{e}-100$ to $1\text{e}+10$ (may be nonintegral)
Domain np : $-1,000$ to $1,000$
Domain t : $-8\text{e}+307$ to $8\text{e}+307$
Range: 0 to 0.39894 ...

nt(*df*,*np*,*t*)

Description: the cumulative noncentral Student's t distribution with df degrees of freedom and noncentrality parameter np
 $\text{nt}(df, 0, t) = \text{t}(df, t)$.
Domain df : $1\text{e}-100$ to $1\text{e}+10$ (may be nonintegral)
Domain np : $-1,000$ to $1,000$
Domain t : $-8\text{e}+307$ to $8\text{e}+307$
Range: 0 to 1

nttail(*df*,*np*,*t*)

Description: the reverse cumulative (upper tail or survivor) noncentral Student's t distribution with df degrees of freedom and noncentrality parameter np
Domain df : $1\text{e}-100$ to $1\text{e}+10$ (may be nonintegral)
Domain np : $-1,000$ to $1,000$
Domain t : $-8\text{e}+307$ to $8\text{e}+307$
Range: 0 to 1

npnt(*df*,*t*,*p*)

Description: the noncentrality parameter, np , for the noncentral Student's t distribution: if $\text{nt}(df, np, t) = p$, then $\text{npnt}(df, t, p) = np$
Domain df : $1\text{e}-100$ to $1\text{e}+8$ (may be nonintegral)
Domain t : $-8\text{e}+307$ to $8\text{e}+307$
Domain p : 0 to 1
Range: $-1,000$ to $1,000$

Tukey's Studentized range distribution

`tukeyprob(k, df, x)`

Description: the cumulative Tukey's Studentized range distribution with k ranges and df degrees of freedom; 0 if $x < 0$

If df is a missing value, then the normal distribution is used instead of Student's t .

`tukeyprob()` is computed using an algorithm described in [Miller \(1981\)](#).

Domain k : 2 to 1e+6

Domain df : 2 to 1e+6

Domain x : $-8e+307$ to $8e+307$

Range: 0 to 1

`invtukeyprob(k, df, p)`

Description: the inverse cumulative Tukey's Studentized range distribution with k ranges and df degrees of freedom

If df is a missing value, then the normal distribution is used instead of Student's t .

If `tukeyprob(k, df, x) = p`, then `invtukeyprob(k, df, p) = x`.

`invtukeyprob()` is computed using an algorithm described in [Miller \(1981\)](#).

Domain k : 2 to 1e+6

Domain df : 2 to 1e+6

Domain p : 0 to 1

Range: 0 to $8e+307$

Weibull distribution

`weibullden(a, b, x)`

Description: the probability density function of the Weibull distribution with shape a and scale b

`weibullden(a, b, x) = weibullden(a, b, 0, x)`, where a is the shape, b is the scale, and x is the value of Weibull random variable.

Domain a : 1e-323 to $8e+307$

Domain b : 1e-323 to $8e+307$

Domain x : 1e-323 to $8e+307$

Range: 0 to $8e+307$

weibullden(*a,b,g,x*)

Description: the probability density function of the Weibull distribution with shape *a*, scale *b*, and location *g*

The probability density function of the generalized Weibull distribution is defined as

$$\frac{a}{b} \left(\frac{x-g}{b} \right)^{a-1} \exp \left\{ - \left(\frac{x-g}{b} \right)^a \right\}$$

for $x \geq g$ and 0 for $x < g$, where *a* is the shape, *b* is the scale, *g* is the location parameter, and *x* is the value of a generalized Weibull random variable.

Domain *a*: 1e-323 to 8e+307

Domain *b*: 1e-323 to 8e+307

Domain *g*: -8e+307 to 8e+307

Domain *x*: -8e+307 to 8e+307; interesting domain is $x \geq g$

Range: 0 to 8e+307

weibull(*a,b,x*)

Description: the cumulative Weibull distribution with shape *a* and scale *b*

weibull(*a,b,x*) = **weibull**(*a, b, 0, x*), where *a* is the shape, *b* is the scale, and *x* is the value of Weibull random variable.

Domain *a*: 1e-323 to 8e+307

Domain *b*: 1e-323 to 8e+307

Domain *x*: 1e-323 to 8e+307

Range: 0 to 1

weibull(*a,b,g,x*)

Description: the cumulative Weibull distribution with shape *a*, scale *b*, and location *g*

The cumulative Weibull distribution is defined as

$$1 - \exp \left[- \left(\frac{x-g}{b} \right)^a \right]$$

for $x \geq g$ and 0 for $x < g$, where *a* is the shape, *b* is the scale, *g* is the location parameter, and *x* is the value of a Weibull random variable.

The mean of the Weibull distribution is $g + b\Gamma\{(a+1)/a\}$ and its variance is $b^2 (\Gamma\{(a+2)/a\} - [\Gamma\{(a+1)/a\}]^2)$ where $\Gamma()$ is the gamma function described in [lngamma\(\)](#).

Domain *a*: 1e-323 to 8e+307

Domain *b*: 1e-323 to 8e+307

Domain *g*: -8e+307 to 8e+307

Domain *x*: -8e+307 to 8e+307; interesting domain is $x \geq g$

Range: 0 to 1

weibulltail(*a*,*b*,*x*)

Description: the reverse cumulative Weibull distribution with shape *a* and scale *b*

$\text{weibulltail}(a,b,x) = \text{weibulltail}(a,b,0,x)$, where *a* is the shape, *b* is the scale, and *x* is the value of a Weibull random variable.

Domain *a*: 1e-323 to 8e+307

Domain *b*: 1e-323 to 8e+307

Domain *x*: 1e-323 to 8e+307

Range: 0 to 1

weibulltail(*a*,*b*,*g*,*x*)

Description: the reverse cumulative Weibull distribution with shape *a*, scale *b*, and location *g*

The reverse cumulative Weibull distribution is defined as

$$\exp \left\{ - \left(\frac{x - g}{b} \right)^a \right\}$$

for $x \geq g$ and 0 if $x < g$, where *a* is the shape, *b* is the scale, *g* is the location parameter, and *x* is the value of a generalized Weibull random variable.

Domain *a*: 1e-323 to 8e+307

Domain *b*: 1e-323 to 8e+307

Domain *g*: -8e+307 to 8e+307

Domain *x*: -8e+307 to 8e+307; interesting domain is $x \geq g$

Range: 0 to 1

invweibull(*a*,*b*,*p*)

Description: the inverse cumulative Weibull distribution with shape *a* and scale *b*: if

$\text{weibull}(a,b,x) = p$, then $\text{invweibull}(a,b,p) = x$

Domain *a*: 1e-323 to 8e+307

Domain *b*: 1e-323 to 8e+307

Domain *p*: 0 to 1

Range: 1e-323 to 8e+307

invweibull(*a*,*b*,*g*,*p*)

Description: the inverse cumulative Weibull distribution with shape *a*, scale *b*, and location *g*: if

$\text{weibull}(a,b,g,x) = p$, then

$\text{invweibull}(a,b,g,p) = x$

Domain *a*: 1e-323 to 8e+307

Domain *b*: 1e-323 to 8e+307

Domain *g*: -8e+307 to 8e+307

Domain *p*: 0 to 1

Range: $g + c(\text{epsdouble})$ to 8e+307

invweibulltail(a, b, p)

Description: the inverse reverse cumulative Weibull distribution with shape a and scale b : if $\text{weibulltail}(a, b, x) = p$, then $\text{invweibulltail}(a, b, p) = x$

Domain a : 1e-323 to 8e+307Domain b : 1e-323 to 8e+307Domain p : 0 to 1

Range: 1e-323 to 8e+307

invweibulltail(a, b, g, p)

Description: the inverse reverse cumulative Weibull distribution with shape a , scale b , and location g : if $\text{weibulltail}(a, b, g, x) = p$, then $\text{invweibulltail}(a, b, g, p) = x$

Domain a : 1e-323 to 8e+307Domain b : 1e-323 to 8e+307Domain g : -8e+307 to 8e+307Domain p : 0 to 1Range: $g + c(\text{epsdouble})$ to 8e+307

Weibull (proportional hazards) distribution

weibullphden(a, b, x)

Description: the probability density function of the Weibull (proportional hazards) distribution with shape a and scale b

$\text{weibullphden}(a, b, x) = \text{weibullphden}(a, b, 0, x)$, where a is the shape, b is the scale, and x is the value of Weibull (proportional hazards) random variable.

Domain a : 1e-323 to 8e+307Domain b : 1e-323 to 8e+307Domain x : 1e-323 to 8e+307

Range: 0 to 8e+307

weibullphden(a, b, g, x)

Description: the probability density function of the Weibull (proportional hazards) distribution with shape a , scale b , and location g

The probability density function of the Weibull (proportional hazards) distribution is defined as

$$ba(x - g)^{a-1} \exp \{-b(x - g)^a\}$$

for $x \geq g$ and 0 for $x < g$, where a is the shape, b is the scale, g is the location parameter, and x is the value of a Weibull (proportional hazards) random variable.

Domain a : 1e-323 to 8e+307Domain b : 1e-323 to 8e+307Domain g : -8e+307 to 8e+307Domain x : -8e+307 to 8e+307; interesting domain is $x \geq g$

Range: 0 to 8e+307

weibullph(*a*,*b*,*x*)

Description: the cumulative Weibull (proportional hazards) distribution with shape *a* and scale *b*

$\text{weibullph}(a,b,x) = \text{weibullph}(a, b, 0, x)$, where *a* is the shape, *b* is the scale, and *x* is the value of Weibull random variable.

Domain *a*: 1e-323 to 8e+307

Domain *b*: 1e-323 to 8e+307

Domain *x*: 1e-323 to 8e+307

Range: 0 to 1

weibullph(*a*,*b*,*g*,*x*)

Description: the cumulative Weibull (proportional hazards) distribution with shape *a*, scale *b*, and location *g*

The cumulative Weibull (proportional hazards) distribution is defined as

$$1 - \exp \{-b(x - g)^a\}$$

for $x \geq g$ and 0 if $x < g$, where *a* is the shape, *b* is the scale, *g* is the location parameter, and *x* is the value of a Weibull (proportional hazards) random variable. The mean of the Weibull (proportional hazards) distribution is

$$g + b^{-\frac{1}{a}} \Gamma\{(a+1)/a\}$$

and its variance is

$$b^{-\frac{2}{a}} (\Gamma\{(a+2)/a\} - [\Gamma\{(a+1)/a\}]^2)$$

where $\Gamma()$ is the gamma function described in [lngamma\(*x*\)](#) .

Domain *a*: 1e-323 to 8e+307

Domain *b*: 1e-323 to 8e+307

Domain *g*: -8e+307 to 8e+307

Domain *x*: -8e+307 to 8e+307; interesting domain is $x \geq g$

Range: 0 to 1

weibullphtail(*a*,*b*,*x*)

Description: the reverse cumulative Weibull (proportional hazards) distribution with shape *a* and scale *b*

$\text{weibullphtail}(a,b,x) = \text{weibullphtail}(a,b,0,x)$, where *a* is the shape, *b* is the scale, and *x* is the value of a Weibull (proportional hazards) random variable.

Domain *a*: 1e-323 to 8e+307

Domain *b*: 1e-323 to 8e+307

Domain *x*: 1e-323 to 8e+307

Range: 0 to 1

`weibullphtail(a,b,g,x)`

Description: the reverse cumulative Weibull (proportional hazards) distribution with shape a , scale b , and location g

The reverse cumulative Weibull (proportional hazards) distribution is defined as

$$\exp\{-b(x-g)^a\}$$

for $x \geq g$ and 0 of $x < g$, where a is the shape, b is the scale, g is the location parameter, and x is the value of a Weibull (proportional hazards) random variable.

Domain a : 1e-323 to 8e+307

Domain b : 1e-323 to 8e+307

Domain g : -8e+307 to 8e+307

Domain x : -8e+307 to 8e+307; interesting domain is $x \geq g$

Range: 0 to 1

`invweibullph(a,b,p)`

Description: the inverse cumulative Weibull (proportional hazards) distribution with shape a and scale b : if `weibullph(a,b,x) = p`, then `invweibullph(a,b,p) = x`

Domain a : 1e-323 to 8e+307

Domain b : 1e-323 to 8e+307

Domain p : 0 to 1

Range: 1e-323 to 8e+307

`invweibullph(a,b,g,p)`

Description: the inverse cumulative Weibull (proportional hazards) distribution with shape a , scale b , and location g : if `weibullph(a,b,g,x) = p`, then `invweibullph(a,b,g,p) = x`

Domain a : 1e-323 to 8e+307

Domain b : 1e-323 to 8e+307

Domain g : -8e+307 to 8e+307

Domain p : 0 to 1

Range: $g + c(\text{epsdouble})$ to 8e+307

`invweibullphtail(a,b,p)`

Description: the inverse reverse cumulative Weibull (proportional hazards) distribution with shape a and scale b : if `weibullphtail(a,b,x) = p`, then `invweibullphtail(a,b,p) = x`

Domain a : 1e-323 to 8e+307

Domain b : 1e-323 to 8e+307

Domain p : 0 to 1

Range: 1e-323 to 8e+307

`invweibullphtail(a,b,g,p)`

Description: the inverse reverse cumulative Weibull (proportional hazards) distribution with shape a , scale b , and location g : if `weibullphtail(a,b,g,x) = p`, then
`invweibullphtail(a,b,g,p) = x`

Domain a : $1\text{e}-323$ to $8\text{e}+307$

Domain b : $1\text{e}-323$ to $8\text{e}+307$

Domain g : $-8\text{e}+307$ to $8\text{e}+307$

Domain p : 0 to 1

Range: $g + c(\text{epsdouble})$ to $8\text{e}+307$

Wishart distribution

`lnwishartden(df,V,X)`

Description: the natural logarithm of the density of the Wishart distribution; missing if $df \leq n - 1$
 df denotes the degrees of freedom, V is the scale matrix, and X is the Wishart random matrix.

Domain df : 1 to $1\text{e}+100$ (may be nonintegral)

Domain V : $n \times n$, positive-definite, symmetric matrices

Domain X : $n \times n$, positive-definite, symmetric matrices

Range: $-8\text{e}+307$ to $8\text{e}+307$

`lniwishartden(df,V,X)`

Description: the natural logarithm of the density of the inverse Wishart distribution; missing if $df \leq n - 1$

df denotes the degrees of freedom, V is the scale matrix, and X is the inverse Wishart random matrix.

Domain df : 1 to $1\text{e}+100$ (may be nonintegral)

Domain V : $n \times n$, positive-definite, symmetric matrices

Domain X : $n \times n$, positive-definite, symmetric matrices

Range: $-8\text{e}+307$ to $8\text{e}+307$

John Wishart (1898–1956) was born in Montrose, Scotland. He obtained a degree in mathematics and physics from the University of Edinburgh. He learned mathematics from E. T. Whittaker, upon whose recommendation he became Karl Pearson's research assistant. During his apprenticeship, he worked on approximations to the incomplete beta function and published multiple papers on this topic. He is best known for deriving the generalized product moment distribution, which was consequently named the Wishart distribution. This distribution is a critical component in the calculation of covariance matrices and Bayesian statistics.

Wishart served in both world wars, fighting with the Black Watch regiment in the first and working for the Intelligence Corps in the second. Upon his return from World War II, he resumed his involvement with the Royal Statistical Society, becoming chairman of the Research Section in 1945. A few years later, he also served as Associate Editor for the journal *Biometrika*.

He taught courses in statistics and agriculture at Cambridge and became the Head of the Statistical Laboratory. He published multiple papers applying statistical methods to agricultural research and was involved with the United Nations Food and Agriculture Organization. He was in Mexico to establish an agricultural research center on behalf of this organization when he died.

References

- Dunnett, C. W. 1955. A multiple comparison for comparing several treatments with a control. *Journal of the American Statistical Association* 50: 1096–1121.
- Johnson, N. L., S. Kotz, and N. Balakrishnan. 1995. *Continuous Univariate Distributions, Vol. 2*. 2nd ed. New York: Wiley.
- Miller, R. G., Jr. 1981. *Simultaneous Statistical Inference*. 2nd ed. New York: Springer.
- Moore, R. J. 1982. Algorithm AS 187: Derivatives of the incomplete gamma integral. *Applied Statistics* 31: 330–335.
- Posten, H. O. 1993. An effective algorithm for the noncentral beta distribution function. *American Statistician* 47: 129–131.
- Press, W. H., S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. 2007. *Numerical Recipes: The Art of Scientific Computing*. 3rd ed. New York: Cambridge University Press.
- Tamhane, A. C. 2008. Eulogy to Charles Dunnett. *Biometrical Journal* 50: 636–637.

Also see

- [FN] **Functions by category**
- [D] **egen** — Extensions to generate
- [D] **generate** — Create or change contents of variable
- [M-4] **Statistical** — Statistical functions
- [U] **13.3 Functions**

Contents

<code>abbrev(<i>s</i>,<i>n</i>)</code>	name <i>s</i> , abbreviated to a length of <i>n</i>
<code>char(<i>n</i>)</code>	the character corresponding to ASCII or extended ASCII code <i>n</i> ; "" if <i>n</i> is not in the domain
<code>collatorlocale(<i>loc</i>,<i>type</i>)</code>	the most closely related locale supported by ICU from <i>loc</i> if <i>type</i> is 1; the actual locale where the collation data comes from if <i>type</i> is 2
<code>collatorversion(<i>loc</i>)</code>	the version string of a collator based on locale <i>loc</i>
<code>indexnot(<i>s</i>₁,<i>s</i>₂)</code>	the position in ASCII string <i>s</i> ₁ of the first character of <i>s</i> ₁ not found in ASCII string <i>s</i> ₂ , or 0 if all characters of <i>s</i> ₁ are found in <i>s</i> ₂
<code>plural(<i>n</i>,<i>s</i>)</code>	the plural of <i>s</i> if <i>n</i> ≠ ±1
<code>plural(<i>n</i>,<i>s</i>₁,<i>s</i>₂)</code>	the plural of <i>s</i> ₁ , as modified by or replaced with <i>s</i> ₂ , if <i>n</i> ≠ ±1
<code>real(<i>s</i>)</code>	<i>s</i> converted to numeric or <i>missing</i>
<code>regexm(<i>s</i>,<i>re</i>)</code>	performs a match of a regular expression and evaluates to 1 if regular expression <i>re</i> is satisfied by the ASCII string <i>s</i> ; otherwise, 0
<code>regexpr(<i>s</i>₁,<i>re</i>,<i>s</i>₂)</code>	replaces the first substring within ASCII string <i>s</i> ₁ that matches <i>re</i> with ASCII string <i>s</i> ₂ and returns the resulting string
<code>regexs(<i>n</i>)</code>	subexpression <i>n</i> from a previous <code>regexm()</code> match, where 0 ≤ <i>n</i> < 10
<code>soundex(<i>s</i>)</code>	the soundex code for a string, <i>s</i>
<code>soundex_nara(<i>s</i>)</code>	the U.S. Census soundex code for a string, <i>s</i>
<code>strcat(<i>s</i>₁,<i>s</i>₂)</code>	there is no <code>strcat()</code> function; instead the addition operator is used to concatenate strings
<code>strdup(<i>s</i>₁,<i>n</i>)</code>	there is no <code>strdup()</code> function; instead the multiplication operator is used to create multiple copies of strings
<code>string(<i>n</i>)</code>	a synonym for <code>strofreal(<i>n</i>)</code>
<code>string(<i>n</i>,<i>s</i>)</code>	a synonym for <code>strofreal(<i>n</i>,<i>s</i>)</code>
<code>stritrim(<i>s</i>)</code>	<i>s</i> with multiple, consecutive internal blanks (ASCII space character <code>char(32)</code>) collapsed to one blank
<code>strlen(<i>s</i>)</code>	the number of characters in ASCII <i>s</i> or length in bytes
<code>strlower(<i>s</i>)</code>	lowercase ASCII characters in string <i>s</i>
<code>strltrim(<i>s</i>)</code>	<i>s</i> without leading blanks (ASCII space character <code>char(32)</code>)
<code>strmatch(<i>s</i>₁,<i>s</i>₂)</code>	1 if <i>s</i> ₁ matches the pattern <i>s</i> ₂ ; otherwise, 0
<code>strofreal(<i>n</i>)</code>	<i>n</i> converted to a string
<code>strofreal(<i>n</i>,<i>s</i>)</code>	<i>n</i> converted to a string using the specified display format
<code>strpos(<i>s</i>₁,<i>s</i>₂)</code>	the position in <i>s</i> ₁ at which <i>s</i> ₂ is first found; otherwise, 0
<code>strproper(<i>s</i>)</code>	a string with the first ASCII letter and any other letters immediately following characters that are not letters capitalized; all other ASCII letters converted to lowercase

<code>strreverse(<i>s</i>)</code>	reverses the ASCII string <i>s</i>
<code>strrpos(<i>s</i>₁,<i>s</i>₂)</code>	the position in <i>s</i> ₁ at which <i>s</i> ₂ is last found; otherwise, 0
<code>strrtrim(<i>s</i>)</code>	<i>s</i> without trailing blanks (ASCII space character <code>char(32)</code>)
<code>strtoname(<i>s</i>[,<i>p</i>])</code>	<i>s</i> translated into a Stata 13 compatible name
<code>strtrim(<i>s</i>)</code>	<i>s</i> without leading and trailing blanks (ASCII space character <code>char(32)</code>); equivalent to <code>strltrim(strrtrim(<i>s</i>))</code>
<code>strupper(<i>s</i>)</code>	uppercase ASCII characters in string <i>s</i>
<code>subinstr(<i>s</i>₁,<i>s</i>₂,<i>s</i>₃,<i>n</i>)</code>	<i>s</i> ₁ , where the first <i>n</i> occurrences in <i>s</i> ₁ of <i>s</i> ₂ have been replaced with <i>s</i> ₃
<code>subinword(<i>s</i>₁,<i>s</i>₂,<i>s</i>₃,<i>n</i>)</code>	<i>s</i> ₁ , where the first <i>n</i> occurrences in <i>s</i> ₁ of <i>s</i> ₂ as a word have been replaced with <i>s</i> ₃
<code>substr(<i>s</i>,<i>n</i>₁,<i>n</i>₂)</code>	the substring of <i>s</i> , starting at <i>n</i> ₁ , for a length of <i>n</i> ₂
<code>tobytes(<i>s</i>[,<i>n</i>])</code>	escaped decimal or hex digit strings of up to 200 bytes of <i>s</i>
<code>uchar(<i>n</i>)</code>	the Unicode character corresponding to Unicode code point <i>n</i> or an empty string if <i>n</i> is beyond the Unicode code-point range
<code>udstrlen(<i>s</i>)</code>	the number of display columns needed to display the Unicode string <i>s</i> in the Stata Results window
<code>udsubstr(<i>s</i>,<i>n</i>₁,<i>n</i>₂)</code>	the Unicode substring of <i>s</i> , starting at character <i>n</i> ₁ , for <i>n</i> ₂ display columns
<code>uisdigit(<i>s</i>)</code>	1 if the first Unicode character in <i>s</i> is a Unicode decimal digit; otherwise, 0
<code>uisletter(<i>s</i>)</code>	1 if the first Unicode character in <i>s</i> is a Unicode letter; otherwise, 0
<code>ustrcompare(<i>s</i>₁,<i>s</i>₂[,<i>loc</i>])</code>	compares two Unicode strings
<code>ustrcompareex(<i>s</i>₁,<i>s</i>₂,<i>loc</i>,<i>st</i>,<i>case</i>,<i>cslv</i>,<i>norm</i>,<i>num</i>,<i>alt</i>,<i>fr</i>)</code>	compares two Unicode strings
<code>ustrfix(<i>s</i>[,<i>rep</i>])</code>	replaces each invalid UTF-8 sequence with a Unicode character
<code>ustrfrom(<i>s</i>,<i>enc</i>,<i>mode</i>)</code>	converts the string <i>s</i> in encoding <i>enc</i> to a UTF-8 encoded Unicode string
<code>ustrinvalidcnt(<i>s</i>)</code>	the number of invalid UTF-8 sequences in <i>s</i>
<code>ustrleft(<i>s</i>,<i>n</i>)</code>	the first <i>n</i> Unicode characters of the Unicode string <i>s</i>
<code>ustrlen(<i>s</i>)</code>	the number of characters in the Unicode string <i>s</i>
<code>ustrlower(<i>s</i>[,<i>loc</i>])</code>	lowercase all characters of Unicode string <i>s</i> under the given locale <i>loc</i>
<code>ustrltrim(<i>s</i>)</code>	removes the leading Unicode whitespace characters and blanks from the Unicode string <i>s</i>
<code>ustrnormalize(<i>s</i>,<i>norm</i>)</code>	normalizes Unicode string <i>s</i> to one of the five normalization forms specified by <i>norm</i>
<code>ustrpos(<i>s</i>₁,<i>s</i>₂[,<i>n</i>])</code>	the position in <i>s</i> ₁ at which <i>s</i> ₂ is first found; otherwise, 0
<code>ustrregexm(<i>s</i>,<i>re</i>[,<i>noc</i>])</code>	performs a match of a regular expression and evaluates to 1 if regular expression <i>re</i> is satisfied by the Unicode string <i>s</i> ; otherwise, 0
<code>ustrregexra(<i>s</i>₁,<i>re</i>,<i>s</i>₂[,<i>noc</i>])</code>	replaces all substrings within the Unicode string <i>s</i> ₁ that match <i>re</i> with <i>s</i> ₂ and returns the resulting string
<code>ustrregexrf(<i>s</i>₁,<i>re</i>,<i>s</i>₂[,<i>noc</i>])</code>	replaces the first substring within the Unicode string <i>s</i> ₁ that matches <i>re</i> with <i>s</i> ₂ and returns the resulting string

<code>ustrregexs(<i>n</i>)</code>	subexpression <i>n</i> from a previous <code>ustrregexm()</code> match
<code>ustrreverse(<i>s</i>)</code>	reverses the Unicode string <i>s</i>
<code>ustrright(<i>s</i>,<i>n</i>)</code>	the last <i>n</i> Unicode characters of the Unicode string <i>s</i>
<code>ustrrpos(<i>s</i>₁,<i>s</i>₂[,<i>n</i>])</code>	the position in <i>s</i> ₁ at which <i>s</i> ₂ is last found; otherwise, 0
<code>ustrtrim(<i>s</i>)</code>	remove trailing Unicode whitespace characters and blanks from the Unicode string <i>s</i>
<code>ustrsortkey(<i>s</i>[,<i>loc</i>])</code>	generates a null-terminated byte array that can be used by the <code>sort</code> command to produce the same order as <code>ustrcompare()</code>
<code>ustrsortkeyex(<i>s</i>,<i>loc</i>,<i>st</i>,<i>case</i>,<i>cslv</i>,<i>norm</i>,<i>num</i>,<i>alt</i>,<i>fr</i>)</code>	generates a null-terminated byte array that can be used by the <code>sort</code> command to produce the same order as <code>ustrcompare()</code>
<code>ustrtitle(<i>s</i>[,<i>loc</i>])</code>	a string with the first characters of Unicode words titlecased and other characters lowercased
<code>ustrto(<i>s</i>,<i>enc</i>,<i>mode</i>)</code>	converts the Unicode string <i>s</i> in UTF-8 encoding to a string in encoding <i>enc</i>
<code>ustrtohex(<i>s</i>[,<i>n</i>])</code>	escaped hex digit string of <i>s</i> up to 200 Unicode characters
<code>ustrtoname(<i>s</i>[,<i>p</i>])</code>	string <i>s</i> translated into a Stata name
<code>ustrtrim(<i>s</i>)</code>	removes leading and trailing Unicode whitespace characters and blanks from the Unicode string <i>s</i>
<code>ustrunescape(<i>s</i>)</code>	the Unicode string corresponding to the escaped sequences of <i>s</i>
<code>ustrupper(<i>s</i>[,<i>loc</i>])</code>	uppercase all characters in string <i>s</i> under the given locale <i>loc</i>
<code>ustrword(<i>s</i>,<i>n</i>[,<i>loc</i>])</code>	the <i>n</i> th Unicode word in the Unicode string <i>s</i>
<code>ustrwordcount(<i>s</i>[,<i>loc</i>])</code>	the number of nonempty Unicode words in the Unicode string <i>s</i>
<code>usubinstr(<i>s</i>₁,<i>s</i>₂,<i>s</i>₃,<i>n</i>)</code>	replaces the first <i>n</i> occurrences of the Unicode string <i>s</i> ₂ with the Unicode string <i>s</i> ₃ in <i>s</i> ₁
<code>usubstr(<i>s</i>,<i>n</i>₁,<i>n</i>₂)</code>	the Unicode substring of <i>s</i> , starting at <i>n</i> ₁ , for a length of <i>n</i> ₂
<code>word(<i>s</i>,<i>n</i>)</code>	the <i>n</i> th word in <i>s</i> ; <i>missing</i> ("") if <i>n</i> is missing
<code>wordbreaklocale(<i>loc</i>,<i>type</i>)</code>	the most closely related locale supported by ICU from <i>loc</i> if <i>type</i> is 1, the actual locale where the word-boundary analysis data come from if <i>type</i> is 2; or an empty string is returned for any other <i>type</i>
<code>wordcount(<i>s</i>)</code>	the number of words in <i>s</i>

Functions

In the display below, *s* indicates a string subexpression (a string literal, a string variable, or another string expression) and *n* indicates a numeric subexpression (a number, a numeric variable, or another numeric expression).

If your strings contain Unicode characters or you are writing programs that will be used by others who might use Unicode strings, read [\[U\] 12.4.2 Handling Unicode strings](#).

abbrev(*s*,*n*)

Description: name *s*, abbreviated to a length of *n*

Length is measured in the number of [display columns](#), not in the number of characters. For most users, the number of display columns equals the number of characters. For a detailed discussion of display columns, see [\[U\] 12.4.2.2 Displaying Unicode characters](#).

If any of the characters of *s* are a period, “.”, and *n* < 8, then the value of *n* defaults to a value of 8. Otherwise, if *n* < 5, then *n* defaults to a value of 5. If *n* is *missing*, `abbrev()` will return the entire string *s*. `abbrev()` is typically used with variable names and variable names with factor-variable or time-series operators (the period case).

`abbrev("displacement",8)` is `displa~t`.

Domain *s*: strings

Domain *n*: integers 5 to 32

Range: strings

char(*n*)

Description: the character corresponding to ASCII or extended ASCII code *n*; "" if *n* is not in the domain

Note: ASCII codes are from 0 to 127; extended ASCII codes are from 128 to 255. Prior to Stata 14, the display of extended ASCII characters was encoding dependent. For example, `char(128)` on Microsoft Windows using Windows-1252 encoding displayed the Euro symbol, but on Linux using ISO-Latin-1 encoding, `char(128)` displayed an invalid character symbol. Beginning with Stata 14, Stata's display encoding is UTF-8 on all platforms. The `char(128)` function is an invalid UTF-8 sequence and thus will display a question mark. There are two Unicode functions corresponding to `char()`: `uchar()` and `ustrunescape()`. You can use `uchar(8364)` or `ustrunescape("\u20AC")` to display a Euro sign on all platforms.

Domain *n*: integers 0 to 255

Range: ASCII characters

uchar(*n*)

Description: the Unicode character corresponding to Unicode code point *n* or an empty string if *n* is beyond the Unicode code-point range

Note that `uchar()` takes the decimal value of the Unicode [code point](#). `ustrunescape()` takes an escaped hex digit string of the Unicode code point. For example, both `uchar(8364)` and `ustrunescape("\u20ac")` produce the Euro sign.

Domain *n*: integers ≥ 0

Range: Unicode characters

collatorlocale(*loc*, *type*)

Description: the most closely related locale supported by ICU from *loc* if *type* is 1; the actual locale where the collation data comes from if *type* is 2

For any other *type*, *loc* is returned in a canonicalized form.

```
collatorlocale("en_us_texas", 0) = en_US_TEXAS
```

```
collatorlocale("en_us_texas", 1) = en_US
```

```
collatorlocale("en_us_texas", 2) = root
```

Domain *loc*: strings of locale name

Domain *type*: integers

Range: strings

collatorversion(*loc*)

Description: the version string of a collator based on locale *loc*

The Unicode standard is constantly adding more characters and the sort key format may change as well. This can cause `ustrsortkey()` and `ustrsortkeyex()` to produce incompatible sort keys between different versions of International Components for Unicode. The version string can be used for versioning the sort keys to indicate when saved sort keys must be regenerated.

Range: strings

indexnot(*s*₁, *s*₂)

Description: the position in ASCII string *s*₁ of the first character of *s*₁ not found in ASCII string *s*₂, or 0 if all characters of *s*₁ are found in *s*₂

`indexnot()` is intended for use with only [plain ASCII](#) strings. For Unicode characters beyond the plain ASCII range, the position and character are given in [bytes](#), not characters.

Domain *s*₁: ASCII strings (to be searched)

Domain *s*₂: ASCII strings (to search for)

Range: integers ≥ 0

plural(*n*, *s*)

Description: the plural of *s* if $n \neq \pm 1$

The plural is formed by adding “s” to *s*.

```
plural(1, "horse") = "horse"
```

```
plural(2, "horse") = "horses"
```

Domain *n*: real numbers

Domain *s*: strings

Range: strings

plural(*n*, *s*₁, *s*₂)

Description: the plural of *s*₁, as modified by or replaced with *s*₂, if *n* ≠ ±1

If *s*₂ begins with the character “+”, the plural is formed by adding the remainder of *s*₂ to *s*₁. If *s*₂ begins with the character “-”, the plural is formed by subtracting the remainder of *s*₂ from *s*₁. If *s*₂ begins with neither “+” nor “-”, then the plural is formed by returning *s*₂.

```
plural(2, "glass", "+es") = "glasses"
plural(1, "mouse", "mice") = "mouse"
plural(2, "mouse", "mice") = "mice"
plural(2, "abcdefg", "-efg") = "abcd"
```

Domain *n*: real numbers

Domain *s*₁: strings

Domain *s*₂: strings

Range: strings

real(*s*)

Description: *s* converted to numeric or *missing*

Also see [strofreal\(\)](#).

```
real("5.2")+1 = 6.2
```

```
real("hello") = .
```

Domain *s*: strings

Range: -8e+307 to 8e+307 or *missing*

regextm(*s*, *re*)

Description: performs a match of a regular expression and evaluates to 1 if regular expression *re* is satisfied by the ASCII string *s*; otherwise, 0

Regular expression syntax is based on Henry Spencer’s NFA algorithm, and this is nearly identical to the POSIX.2 standard. *s* and *re* may not contain binary 0 (\0).

regextm() is intended for use with only [plain ASCII](#) characters. For Unicode characters beyond the plain ASCII range, the match is based on [bytes](#). For a character-based match, see [ustrregextm\(\)](#).

Domain *s*: ASCII strings

Domain *re*: regular expressions

Range: ASCII strings

regexr(*s*₁, *re*, *s*₂)

Description: replaces the first substring within ASCII string *s*₁ that matches *re* with ASCII string *s*₂ and returns the resulting string

If *s*₁ contains no substring that matches *re*, the unaltered *s*₁ is returned. *s*₁ and the result of **regexr()** may be at most 1,100,000 characters long. *s*₁, *re*, and *s*₂ may not contain binary 0 (\0).

regexr() is intended for use with only **plain ASCII** characters. For Unicode characters beyond the plain ASCII range, the match is based on **bytes** and the result is restricted to 1,100,000 bytes. For a character-based match, see **ustrregextf()** or **ustrregexra()**.

Domain *s*₁: ASCII strings

Domain *re*: regular expressions

Domain *s*₂: ASCII strings

Range: ASCII strings

regexts(*n*)

Description: subexpression *n* from a previous **regextm()** match, where $0 \leq n < 10$

Subexpression 0 is reserved for the entire string that satisfied the regular expression. The returned subexpression may be at most 1,100,000 characters (bytes) long.

Domain *n*: 0 to 9

Range: ASCII strings

ustrregextm(*s*, *re*[, *noc*])

Description: performs a match of a regular expression and evaluates to 1 if regular expression *re* is satisfied by the Unicode string *s*; otherwise, 0

If *noc* is specified and not 0, a case-insensitive match is performed. The function may return a negative integer if an error occurs.

ustrregextm("12345", "([0-9]){5}") = 1

ustrregextm("de TRÈS près", "rès") = 1

ustrregextm("de TRÈS près", "Rès") = 0

ustrregextm("de TRÈS près", "Rès", 1) = 1

Domain *s*: Unicode strings

Domain *re*: Unicode regular expressions

Domain *noc*: integers

Range: integers

ustrregexrf(*s*₁, *re*, *s*₂[, *noc*])

Description: replaces the first substring within the Unicode string *s*₁ that matches *re* with *s*₂ and returns the resulting string

If *noc* is specified and not 0, a case-insensitive match is performed. The function may return an empty string if an error occurs.

```
ustrregexrf("très près", "rès", "X") = "tX près"
ustrregexrf("TRÈS près", "Rès", "X") = "TRÈS près"
ustrregexrf("TRÈS près", "Rès", "X", 1) = "TX près"
```

Domain *s*₁: Unicode strings
Domain *re*: Unicode regular expressions
Domain *s*₂: Unicode strings
Domain *noc*: integers
Range: Unicode strings

ustrregexra(*s*₁, *re*, *s*₂[, *noc*])

Description: replaces all substrings within the Unicode string *s*₁ that match *re* with *s*₂ and returns the resulting string

If *noc* is specified and not 0, a case-insensitive match is performed. The function may return an empty string if an error occurs.

```
ustrregexra("très près", "rès", "X") = "tX pX"
ustrregexra("TRÈS près", "Rès", "X") = "TRÈS près"
ustrregexra("TRÈS près", "Rès", "X", 1) = "TX pX"
```

Domain *s*₁: Unicode strings
Domain *re*: Unicode regular expressions
Domain *s*₂: Unicode strings
Domain *noc*: integers
Range: Unicode strings

ustrregexs(*n*)

Description: subexpression *n* from a previous **ustrregextm()** match

Subexpression 0 is reserved for the entire string that satisfied the regular expression. The function may return an empty string if *n* is larger than the maximum count of subexpressions from the previous match or if an error occurs.

Domain *n*: integers ≥ 0
Range: strings

soundex(*s*)

Description: the soundex code for a string, *s*

The soundex code consists of a letter followed by three numbers: the letter is the first ASCII letter of the name and the numbers encode the remaining consonants. Similar sounding consonants are encoded by the same number. Unicode characters beyond the [plain ASCII](#) range are ignored.

```
soundex("Ashcraft") = "A226"
```

```
soundex("Robert") = "R163"
```

```
soundex("Rupert") = "R163"
```

Domain *s*: strings

Range: strings

soundex_nara(*s*)

Description: the U.S. Census soundex code for a string, *s*

The soundex code consists of a letter followed by three numbers: the letter is the first ASCII letter of the name and the numbers encode the remaining consonants. Similar sounding consonants are encoded by the same number. Unicode characters beyond the [plain ASCII](#) range are ignored.

```
soundex_nara("Ashcraft") = "A261"
```

Domain *s*: strings

Range: strings

strcat(*s*₁, *s*₂)

Description: there is no `strcat()` function; instead the addition operator is used to concatenate strings

```
"hello " + "world" = "hello world"
```

```
"a" + "b" = "ab"
```

```
"Café " + "de Flore" = "Café de Flore"
```

Domain *s*₁: strings

Domain *s*₂: strings

Range: strings

strdup(*s*₁, *n*)

Description: there is no `strdup()` function; instead the multiplication operator is used to create multiple copies of strings

```
"hello" * 3 = "hellohellohello"
```

```
3 * "hello" = "hellohellohello"
```

```
0 * "hello" = ""
```

```
"hello" * 1 = "hello"
```

```
"Здравствуй" * 2 = "Здравствуй Здравствуй"
```

Domain *s*₁: strings

Domain *n*: nonnegative integers 0, 1, 2, ...

Range: strings

string(*n*)Description: a synonym for **stprofreal(*n*)****string(*n*,*s*)**Description: a synonym for **stprofreal(*n*,*s*)****stritrim(*s*)**Description: *s* with multiple, consecutive internal blanks (ASCII space character **char(32)**) collapsed to one blank**stritrim("hello there") = "hello there"**Domain *s*: strings

Range: strings with no multiple, consecutive internal blanks

strlen(*s*)Description: the number of characters in ASCII *s* or length in bytes

strlen() is intended for use with only **plain ASCII** characters and for use by programmers who want to obtain the byte-length of a string. Note that any Unicode character beyond ASCII range (code point greater than 127) takes more than 1 byte in the UTF-8 encoding; for example, **é** takes 2 bytes.

For the number of characters in a **Unicode string**, see **ustrlen()**.

strlen("ab") = 2**strlen("é") = 2**Domain *s*: stringsRange: integers ≥ 0 **ustrlen(*s*)**Description: the number of characters in the Unicode string *s*

An invalid UTF-8 sequence is counted as one Unicode character. An invalid UTF-8 sequence may contain one byte or multiple bytes. Note that any Unicode character beyond the **plain ASCII** range (code point greater than 127) takes more than 1 byte in the UTF-8 encoding; for example, **é** takes 2 bytes.

ustrlen("médiane") = 7**strlen("médiane") = 8**Domain *s*: Unicode stringsRange: integers ≥ 0

udstrlen(*s*)

Description: the number of display columns needed to display the Unicode string *s* in the Stata Results window

A Unicode character in the CJK (Chinese, Japanese, and Korean) encoding usually requires two [display columns](#); a Latin character usually requires one column. Any invalid UTF-8 sequence requires one column.

```
udstrlen("中值") = 4
```

```
ustrlen("中值") = 2
```

```
strlen("中值") = 6
```

Domain *s*: Unicode strings

Range: integers ≥ 0

strlower(*s*)

Description: lowercase ASCII characters in string *s*

Unicode characters beyond the [plain ASCII](#) range are ignored.

```
strlower("THIS") = "this"
```

```
strlower("CAFÉ") = "café"
```

Domain *s*: strings

Range: strings with lowercased characters

ustrlower(*s*[*,loc*])

Description: lowercase all characters of Unicode string *s* under the given locale *loc*

If *loc* is not specified, the [default locale](#) is used. The same *s* but different *loc* may produce different results; for example, the lowercase letter of “İ” is “ı” in English but a dotless “i” in Turkish. The same Unicode character can be mapped to different Unicode characters based on its surrounding characters; for example, Greek capital letter sigma Σ has two lowercases: ς, if it is the final character of a word, or σ. The result can be longer or shorter than the input Unicode string in bytes.

```
ustrlower("MÉDIANE","fr") = "médiane"
```

```
ustrlower("ISTANBUL","tr") = "ıstanbul"
```

```
ustrlower("ΟΔΥΣΣΕΥΣ") = "ὀδυσσεὺς"
```

Domain *s*: Unicode strings

Domain *loc*: locale name

Range: Unicode strings

strltrim(*s*)

Description: *s* without leading blanks (ASCII space character `char(32)`)

```
strltrim(" this") = "this"
```

Domain *s*: strings

Range: strings without leading blanks

ustrltrim(*x*)

Description: removes the leading Unicode whitespace characters and blanks from the Unicode string *s*

Note that, in addition to `char(32)`, ASCII characters `char(9)`, `char(10)`, `char(11)`, `char(12)`, and `char(13)` are whitespace characters in Unicode standard.

```
ustrltrim(" this") = "this"
```

```
ustrltrim(char(9)+"this") = "this"
```

```
ustrltrim(ustrunescape("\u1680")+" this") = "this"
```

Domain *s*: Unicode strings

Range: Unicode strings

strmatch(*s*₁, *s*₂)

Description: 1 if *s*₁ matches the pattern *s*₂; otherwise, 0

`strmatch("17.4", "1??4")` returns 1. In *s*₂, "?" means that one character goes here, and "*" means that zero or more bytes go here. Note that a [Unicode character](#) may contain multiple bytes; thus, using "*" with Unicode characters can infrequently result in matches that do not occur at a character boundary.

Also see [regextm\(\)](#), [regexr\(\)](#), and [regexts\(\)](#).

```
strmatch("café", "caf?") = 1
```

Domain *s*₁: strings

Domain *s*₂: strings

Range: integers 0 or 1

stroofreal(*n*)

Description: *n* converted to a string

Also see [real\(\)](#).

```
stroofreal(4)+"F" = "4F"
```

```
stroofreal(1234567) = "1234567"
```

```
stroofreal(12345678) = "1.23e+07"
```

```
stroofreal(.) = "."
```

Domain *n*: $-8\text{e}+307$ to $8\text{e}+307$ or *missing*

Range: strings

stofreal(*n*,*s*)

Description: *n* converted to a string using the specified display format

Also see [real\(\)](#).

```
stofreal(4,"%9.2f") = "4.00"
stofreal(123456789,"%11.0g") = "123456789"
stofreal(123456789,"%13.0gc") = "123,456,789"
stofreal(0,"%td") = "01jan1960"
stofreal(225,"%tq") = "2016q2"
stofreal(225,"not a format") = ""
```

Domain *n*: $-8\text{e}+307$ to $8\text{e}+307$ or *missing*

Domain *s*: strings containing *%fmt* numeric display format

Range: strings

strpos(*s*₁,*s*₂)

Description: the position in *s*₁ at which *s*₂ is first found; otherwise, 0

`strpos()` is intended for use with only [plain ASCII](#) characters and for use by programmers who want to obtain the byte-position of *s*₂. Note that any Unicode character beyond ASCII range (code point greater than 127) takes more than 1 byte in the UTF-8 encoding; for example, *é* takes 2 bytes.

To find the character position of *s*₂ in a [Unicode string](#), see [ustrpos\(\)](#).

```
strpos("this","is") = 3
strpos("this","it") = 0
```

Domain *s*₁: strings (to be searched)

Domain *s*₂: strings (to search for)

Range: integers ≥ 0

ustrpos(*s*₁,*s*₂[,*n*])

Description: the position in *s*₁ at which *s*₂ is first found; otherwise, 0

If *n* is specified and is greater than 0, the search starts at the *n*th Unicode character of *s*₁. An invalid UTF-8 sequence in either *s*₁ or *s*₂ is replaced with a Unicode replacement character `\ufffd` before the search is performed.

```
ustrpos("médiane", "édi") = 2
ustrpos("médiane", "édi", 3) = 0
ustrpos("médiane", "éci") = 0
```

Domain *s*₁: Unicode strings (to be searched)

Domain *s*₂: Unicode strings (to search for)

Domain *n*: integers

Range: integers

strproper(*s*)

Description: a string with the first ASCII letter and any other letters immediately following characters that are not letters capitalized; all other ASCII letters converted to lowercase

strproper() implements a form of [titlecasing](#) and is intended for use with only [plain ASCII](#) strings. Unicode characters beyond ASCII are treated as characters that are not letters. To titlecase strings with Unicode characters beyond the plain ASCII range or to implement language-sensitive rules for titlecasing, see [ustrtitle\(\)](#).

```
strproper("mR. joHn a. sMitH") = "Mr. John A. Smith"  
strproper("jack o'reilly") = "Jack O'Reilly"  
strproper("2-cent's worth") = "2-Cent'S Worth"  
strproper("vous êtes") = "Vous êTes"
```

Domain *s*: strings

Range: strings

ustrtitle(*s*[, *loc*])

Description: a string with the first characters of Unicode words titlecased and other characters lowercased

If *loc* is not specified, the [default locale](#) is used. Note that a Unicode word is different from a Stata word produced by function [word\(\)](#). The Stata word is a space-separated token. A Unicode word is a language unit based on either a set of [word-boundary rules](#) or dictionaries for some languages (Chinese, Japanese, and Thai). The titlecase is also locale dependent and context sensitive; for example, lowercase “ij” is considered a digraph in Dutch. Its titlecase is “IJ”.

```
ustrtitle("vous êtes", "fr") = "Vous Êtes"  
ustrtitle("mR. joHn a. sMitH") = "Mr. John A. Smith"  
ustrtitle("ijmuiden", "en") = "Ijmuiden"  
ustrtitle("ijmuiden", "nl") = "IJmuiden"
```

Domain *s*: Unicode strings

Domain *loc*: Unicode strings

Range: Unicode strings

strreverse(*s*)

Description: reverses the ASCII string *s*

strreverse() is intended for use with only [plain ASCII](#) characters. For Unicode characters beyond ASCII range (code point greater than 127), the [encoded](#) bytes are reversed.

To reverse the characters of [Unicode string](#), see [ustrreverse\(\)](#).

```
strreverse("hello") = "olleh"
```

Domain *s*: ASCII strings

Range: ASCII reversed strings

ustrreverse(*s*)

Description: reverses the Unicode string *s*

The function does not take Unicode character equivalence into consideration. Hence, a Unicode character in a decomposed form will not be reversed as one unit. An invalid UTF-8 sequence is replaced with a Unicode replacement character `\ufffd`.

```
ustrreverse("médiane") = "enaidém"
```

Domain *s*: Unicode strings

Range: reversed Unicode strings

strrpos(*s*₁, *s*₂)

Description: the position in *s*₁ at which *s*₂ is last found; otherwise, 0

`strrpos()` is intended for use with only [plain ASCII](#) characters and for use by programmers who want to obtain the last byte-position of *s*₂. Note that any Unicode character beyond ASCII range (code point greater than 127) takes more than 1 byte in the UTF-8 encoding; for example, `é` takes 2 bytes.

To find the last character position of *s*₂ in a [Unicode string](#), see `ustrrpos()`.

```
strrpos("this", "is") = 3
```

```
strrpos("this is", "is") = 6
```

```
strrpos("this is", "it") = 0
```

Domain *s*₁: strings (to be searched)

Domain *s*₂: strings (to search for)

Range: integers ≥ 0

ustrrpos(*s*₁, *s*₂[*n*])

Description: the position in *s*₁ at which *s*₂ is last found; otherwise, 0

If *n* is specified and is greater than 0, only the part between the first Unicode character and the *n*th Unicode character of *s*₁ is searched. An invalid UTF-8 sequence in either *s*₁ or *s*₂ is replaced with a Unicode replacement character `\ufffd` before the search is performed.

```
ustrrpos("enchanté", "n") = 6
```

```
ustrrpos("enchanté", "n", 5) = 2
```

```
ustrrpos("enchanté", "n", 6) = 6
```

```
ustrrpos("enchanté", "ne") = 0
```

Domain *s*₁: Unicode strings (to be searched)

Domain *s*₂: Unicode strings (to search for)

Domain *n*: integers

Range: integers

strrtrim(*s*)

Description: *s* without trailing blanks (ASCII space character `char(32)`)

```
strrtrim("this ") = "this"
```

Domain *s*: strings

Range: strings without trailing blanks

ustrrrtrim(*s*)

Description: remove trailing Unicode whitespace characters and blanks from the Unicode string *s*

Note that, in addition to `char(32)`, ASCII characters `char(9)`, `char(10)`, `char(11)`, `char(12)`, and `char(13)` are considered whitespace characters in the Unicode standard.

```
ustrrrtrim("this ") = "this"
```

```
ustrltrim("this"+char(10)) = "this"
```

```
ustrrrtrim("this "+ustrunescape("\u2000")) = "this"
```

Domain *s*: Unicode strings

Range: Unicode strings

strtoname(*s*[,*p*])

Description: *s* translated into a Stata 13 compatible name

`strtoname()` results in a name that is truncated to 32 bytes. Each character in *s* that is not allowed in a Stata name is converted to an underscore character, `_`. If the first character in *s* is a numeric character and *p* is not 0, then the result is prefixed with an underscore. Stata 14 names may be 32 characters; see [\[U\] 11.3 Naming conventions](#).

```
strtoname("name") = "name"
```

```
strtoname("a name") = "a_name"
```

```
strtoname("5",1) = "_5"
```

```
strtoname("5:30",1) = "_5_30"
```

```
strtoname("5",0) = "5"
```

```
strtoname("5:30",0) = "5_30"
```

Domain *s*: strings

Domain *p*: integers 0 or 1

Range: strings

ustrtoname(*s*[,*p*])

Description: string *s* translated into a Stata name

`ustrtoname()` results in a name that is truncated to 32 characters. Each character in *s* that is not allowed in a Stata name is converted to an underscore character, `_`. If the first character in *s* is a numeric character and *p* is not 0, then the result is prefixed with an underscore.

```
ustrtoname("name",1) = "name"
```

```
ustrtoname("the médiane") = "the_médiane"
```

```
ustrtoname("Omédiane") = "_Omédiane"
```

```
ustrtoname("Omédiane", 1) = "_Omédiane"
```

```
ustrtoname("Omédiane", 0) = "Omédiane"
```

Domain *s*: Unicode strings

Domain *p*: integers 0 or 1

Range: Unicode strings

strtrim(*s*)

Description: *s* without leading and trailing blanks (ASCII space character `char(32)`); equivalent to `strltrim(strrtrim(s))`

```
strtrim(" this ") = "this"
```

Domain *s*: strings

Range: strings without leading or trailing blanks

ustrtrim(*s*)

Description: removes leading and trailing Unicode whitespace characters and blanks from the Unicode string *s*

Note that, in addition to `char(32)`, ASCII characters `char(9)`, `char(10)`, `char(11)`, `char(12)`, and `char(13)` are considered whitespace characters in the Unicode standard.

```
ustrtrim(" this ") = "this"
```

```
ustrtrim(char(11)+" this ")+char(13) = "this"
```

```
ustrtrim(" this "+ustrunescape("\u2000")) = "this"
```

Domain *s*: Unicode strings

Range: Unicode strings

strupper(*s*)

Description: uppercase ASCII characters in string *s*

Unicode characters beyond the [plain ASCII](#) range are ignored.

```
strupper("this") = "THIS"
```

```
strupper("café") = "CAFÉ"
```

Domain *s*: strings

Range: strings with uppercased characters

ustrupper(*s*[, *loc*])

Description: uppercase all characters in string *s* under the given locale *loc*

If *loc* is not specified, the [default locale](#) is used. The same *s* but a different *loc* may produce different results; for example, the uppercase letter of “i” is “I” in English, but “İ” with a dot in Turkish. The result can be longer or shorter than the input string in bytes; for example, the uppercase form of the German letter ß (code point `\u00df`) is two capital letters “SS”.

```
ustrupper("médiane", "fr") = "MÉDIANE"
```

```
ustrupper("Rußland", "de") = "RUSSLAND"
```

```
ustrupper("istanbul", "tr") = "İSTANBUL"
```

Domain *s*: Unicode strings

Domain *loc*: locale name

Range: Unicode strings

substr(*s*₁,*s*₂,*s*₃,*n*)

Description: *s*₁, where the first *n* occurrences in *s*₁ of *s*₂ have been replaced with *s*₃

substr() is intended for use with only **plain ASCII** characters and for use by programmers who want to perform byte-based substitution. Note that any Unicode character beyond ASCII range (code point greater than 127) takes more than 1 byte in the UTF-8 encoding; for example, **é** takes 2 bytes.

To perform character-based replacement in **Unicode strings**, see **usubstr()**.

If *n* is *missing*, all occurrences are replaced.

Also see **regextm()**, **regext()**, and **regexts()**.

```
substr("this is the day","is","X",1) = "thX is the day"
substr("this is the hour","is","X",2) = "thX X the hour"
substr("this is this","is","X",.) = "thX X thX"
```

Domain *s*₁: strings (to be substituted into)
Domain *s*₂: strings (to be substituted from)
Domain *s*₃: strings (to be substituted with)
Domain *n*: integers ≥ 0 or *missing*
Range: strings

usubstr(*s*₁,*s*₂,*s*₃,*n*)

Description: replaces the first *n* occurrences of the Unicode string *s*₂ with the Unicode string *s*₃ in *s*₁

If *n* is *missing*, all occurrences are replaced. An invalid UTF-8 sequence in *s*₁, *s*₂, or *s*₃ is replaced with a Unicode replacement character **\ufffd** before replacement is performed.

```
usubstr("de très près","ès","es",1) = "de tres près"
usubstr("de très pr'es","ès","X",2) = "de trX prX"
```

Domain *s*₁: Unicode strings (to be substituted into)
Domain *s*₂: Unicode strings (to be substituted from)
Domain *s*₃: Unicode strings (to be substituted with)
Domain *n*: integers ≥ 0 or *missing*
Range: Unicode strings

subinword(s_1, s_2, s_3, n)

Description: s_1 , where the first n occurrences in s_1 of s_2 as a word have been replaced with s_3

A word is defined as a space-separated token. A token at the beginning or end of s_1 is considered space-separated. This is different from a Unicode word, which is a language unit based on either a set of [word-boundary rules](#) or dictionaries for several languages (Chinese, Japanese, and Thai). If n is *missing*, all occurrences are replaced.

Also see [regexm\(\)](#), [regexpr\(\)](#), and [regexs\(\)](#).

```
subinword("this is the day","is","X",1) = "this X the day"
subinword("this is the hour","is","X",.) = "this X the hour"
subinword("this is this","th","X",.) = "this is this"
```

Domain s_1 : strings (to be substituted for)
 Domain s_2 : strings (to be substituted from)
 Domain s_3 : strings (to be substituted with)
 Domain n : integers ≥ 0 or *missing*
 Range: strings

substr(s, n_1, n_2)

Description: the substring of s , starting at n_1 , for a length of n_2

substr() is intended for use with only [plain ASCII](#) characters and for use by programmers who want to extract a subset of bytes from a string. For those with plain ASCII text, n_1 is the starting character, and n_2 is the length of the string in characters. For programmers, **substr()** is technically a byte-based function. For plain ASCII characters, the two are equivalent but you can operate on byte values beyond that range. Note that any Unicode character beyond ASCII range (code point greater than 127) takes more than 1 byte in the UTF-8 encoding; for example, **é** takes 2 bytes.

To obtain substrings of [Unicode strings](#), see [usubstr\(\)](#).

If $n_1 < 0$, n_1 is interpreted as the distance from the end of the string; if $n_2 = .$ (*missing*), the remaining portion of the string is returned.

```
substr("abcdef",2,3) = "bcd"
substr("abcdef",-3,2) = "de"
substr("abcdef",2,.) = "bcdef"
substr("abcdef",-3,.) = "def"
substr("abcdef",2,0) = ""
substr("abcdef",15,2) = ""
```

Domain s : strings
 Domain n_1 : integers ≥ 1 and ≤ -1
 Domain n_2 : integers ≥ 1
 Range: strings

usubstr(*s*,*n*₁,*n*₂)

Description: the Unicode substring of *s*, starting at *n*₁, for a length of *n*₂

If *n*₁ < 0, *n*₁ is interpreted as the distance from the last character of the *s*; if *n*₂ = . (*missing*), the remaining portion of the Unicode string is returned.

`usubstr("médiane",2,3) = "édi"`
`usubstr("médiane",-3,2) = "an"`
`usubstr("médiane",2,.) = "édiane"`

Domain *s*: Unicode strings
Domain *n*₁: integers ≥ 1 and ≤ -1
Domain *n*₂: integers ≥ 1
Range: Unicode strings

udsubstr(*s*,*n*₁,*n*₂)

Description: the Unicode substring of *s*, starting at character *n*₁, for *n*₂ display columns

If *n*₂ = . (*missing*), the remaining portion of the Unicode string is returned. If *n*₂ **display columns** from *n*₁ is in the middle of a Unicode character, the substring stops at the previous Unicode character.

`udsubstr("médiane",2,3) = "édi"`
`udsubstr("中值",1,1) = ""`
`udsubstr("中值",1,2) = "中"`

Domain *s*: Unicode strings
Domain *n*₁: integers ≥ 1
Domain *n*₂: integers ≥ 1
Range: Unicode strings

tobytes(*s*[,*n*])

Description: escaped decimal or hex digit strings of up to 200 bytes of *s*

The escaped decimal digit string is in the form of \dDDD. The escaped hex digit string is in the form of \xhh. If *n* is not specified or is 0, the decimal form is produced. Otherwise, the hex form is produced.

`tobytes("abc") = "\d097\d098\d099"`
`tobytes("abc", 1) = "\x61\x62\x63"`
`tobytes("café") = "\d099\d097\d102\d195\d169"`

Domain *s*: Unicode strings
Domain *n*: integers
Range: strings

uisdigit(*s*)

Description: 1 if the first Unicode character in *s* is a Unicode decimal digit; otherwise, 0

A Unicode decimal digit is a Unicode character with the character property Nd according to the Unicode standard. The function returns -1 if the string starts with an invalid UTF-8 sequence.

Domain *s*: Unicode strings
Range: integers

uisletter(*s*)

Description: 1 if the first Unicode character in *s* is a Unicode letter; otherwise, 0

A Unicode letter is a Unicode character with the character property L according to the Unicode standard. The function returns -1 if the string starts with an invalid UTF-8 sequence.

Domain *s*: Unicode strings

Range: integers

ustrcompare(*s*₁, *s*₂ [, *loc*])

Description: compares two Unicode strings

The function returns -1, 1, or 0 if *s*₁ is less than, greater than, or equal to *s*₂. The function may return a negative number other than -1 if an error happens. The comparison is locale dependent. For example, *z* < *ö* in Swedish but *ö* < *z* in German. If *loc* is not specified, the [default locale](#) is used. The comparison is diacritic and case sensitive. If you need different behavior, for example, case-insensitive comparison, you should use the extended comparison function [ustrcompareex\(\)](#). [Unicode string comparison](#) compares Unicode strings in a language-sensitive manner. On the other hand, the [sort](#) command compares strings in code-point (binary) order. For example, uppercase “Z” (code-point value 90) comes before lowercase “a” (code-point value 97) in code-point order but comes after “a” in any English dictionary.

```
ustrcompare("z", "ö", "sv") = -1
ustrcompare("z", "ö", "de") = 1
```

Domain *s*₁: Unicode strings

Domain *s*₂: Unicode strings

Domain *loc*: Unicode strings

Range: integers

ustrcompareex(*s*₁, *s*₂, *loc*, *st*, *case*, *cslv*, *norm*, *num*, *alt*, *fr*)

Description: compares two Unicode strings

The function returns -1, 1, or 0 if *s*₁ is less than, greater than, or equal to *s*₂. The function may return a negative number other than -1 if an error occurs. The comparison is locale dependent. For example, *z* < *ö* in Swedish but *ö* < *z* in German. If *loc* is not specified, the [default locale](#) is used.

st controls the strength of the comparison. Possible values are 1 (primary), 2 (secondary), 3 (tertiary), 4 (quaternary), or 5 (identical). -1 means to use the default value for the locale. Any other numbers are treated as tertiary. The primary difference represents base letter differences; for example, letter “a” and letter “b” have primary differences. The secondary difference represents diacritical differences on the same base letter; for example, letters “a” and “ä” have secondary differences. The tertiary difference represents case differences of the same base letter; for example, letters “a” and “A” have tertiary differences. Quaternary strength is useful to distinguish between Katakana and Hiragana for the JIS 4061 collation standard. Identical strength is essentially the code-point order of the string, hence, is rarely useful.

```
ustrcompareex("café", "cafe", "fr", 1, -1, -1, -1, -1, -1, -1) = 0
ustrcompareex("café", "cafe", "fr", 2, -1, -1, -1, -1, -1, -1) = 1
ustrcompareex("Café", "café", "fr", 3, -1, -1, -1, -1, -1, -1) = 1
```

case controls the uppercase and lowercase letter order. Possible values are 0 (use order specified in tertiary strength), 1 (uppercase first), or 2 (lowercase first). -1 means to use the default value for the locale. Any other values are treated as 0.

```
ustrcompareex("Caf  ", "caf  ", "fr", -1, 1, -1, -1, -1, -1) = -1
ustrcompareex("Caf  ", "caf  ", "fr", -1, 2, -1, -1, -1, -1) = 1
```

cslv controls whether an extra case level between the secondary level and the tertiary level is generated. Possible values are 0 (off) or 1 (on). -1 means to use the default value for the locale. Any other values are treated as 0. Combining this setting to be “on” and the strength setting to be primary can achieve the effect of ignoring the diacritical differences but preserving the case differences. If the setting is “on”, the result is also affected by the *case* setting.

```
ustrcompareex("caf  ", "Cafe", "fr", 1, -1, 1, -1, -1, -1) = -1
ustrcompareex("caf  ", "Cafe", "fr", 1, 1, 1, -1, -1, -1) = 1
```

norm controls whether the normalization check and normalizations are performed. Possible values are 0 (off) or 1 (on). -1 means to use the default value for the locale. Any other values are treated as 0. Most languages do not require normalization for comparison. Normalization is needed in languages that use multiple combining characters such as Arabic, ancient Greek, or Hebrew.

num controls how contiguous digit substrings are sorted. Possible values are 0 (off) or 1 (on). -1 means to use the default value for the locale. Any other values are treated as 0. If the setting is “on”, substrings consisting of digits are sorted based on the numeric value. For example, “100” is after value “20” instead of before it. Note that the digit substring is limited to 254 digits, and plus/minus signs, decimals, or exponents are not supported.

```
ustrcompareex("100", "20", "en", -1, -1, -1, -1, 0, -1, -1) = -1
ustrcompareex("100", "20", "en", -1, -1, -1, -1, 1, -1, -1) = 1
```

alt controls how spaces and punctuation characters are handled. Possible values are 0 (use primary strength) or 1 (alternative handling). Any other values are treated as 0. If the setting is 1 (alternative handling), “onsite”, “on-site”, and “on site” are considered equals.

```
ustrcompareex("onsite", "on-site", "en",
               -1, -1, -1, -1, -1, 1, -1) = 0
ustrcompareex("onsite", "on site", "en",
               -1, -1, -1, -1, -1, 1, -1) = 0
ustrcompareex("onsite", "on-site", "en",
               -1, -1, -1, -1, -1, 0, -1) = 1
```

fr controls the direction of the secondary strength. Possible values are 0 (off) or 1 (on). -1 means to use the default value for the locale. All other values are treated as “off”. If the setting is “on”, the diacritical letters are sorted backward. Note that the setting is “on” by default only for Canadian French (locale *fr_CA*).

```
ustrcompareex("cot  ", "c  te", "fr_CA", -1, -1, -1, -1, -1, -1, 0) = -1
ustrcompareex("cot  ", "c  te", "fr_CA", -1, -1, -1, -1, -1, -1, 1) = 1
ustrcompareex("cot  ", "c  te", "fr_CA", -1, -1, -1, -1, -1, -1, -1) = 1
ustrcompareex("cot  ", "c  te", "fr", -1, -1, -1, -1, -1, -1, -1) = 1
```

Domain s_1 : Unicode strings
 Domain s_2 : Unicode strings
 Domain loc : Unicode strings
 Domain st : integers
 Domain $case$: integers
 Domain $cslv$: integers
 Domain $norm$: integers
 Domain num : integers
 Domain alt : integers
 Domain fr : integers
 Range: integers

`ustrfix(s [, rep])`

Description: replaces each invalid UTF-8 sequence with a Unicode character

In the one-argument case, the Unicode replacement character `\ufffd` is used. In the two-argument case, the first Unicode character of rep is used. If rep starts with an invalid UTF-8 sequence, then Unicode replacement character `\ufffd` is used. Note that an invalid UTF-8 sequence can contain one byte or multiple bytes.

```
ustrfix(char(200)) = ustrunescape("\ufffd")
ustrfix("ab"+char(200)+"cdé", "") = "abcdé"
ustrfix("ab"+char(229)+char(174)+"cdé", "é") = "abécdé"
```

Domain s : Unicode strings
 Domain rep : Unicode character
 Range: Unicode strings

`ustrfrom(s , enc , $mode$)`

Description: converts the string s in encoding enc to a UTF-8 encoded Unicode string

$mode$ controls how invalid byte sequences in s are handled. The possible values are 1, which substitutes an invalid byte sequence with a Unicode replacement character `\ufffd`; 2, which skips any invalid byte sequences; 3, which stops at the first invalid byte sequence and returns an empty string; or 4, which replaces any byte in an invalid sequence with an escaped hex digit sequence `%Xhh`. Any other values are treated as 1. A good use of value 4 is to check what invalid bytes a Unicode string ust contains by examining the result of `ustrfrom(ust, "utf-8", 4)`.

Also see `ustrto()`.

```
ustrfrom("caf"+char(233), "latin1", 1) = "café"
ustrfrom("caf"+char(233), "utf-8", 1) =
    "caf"+ustrunescape("\ufffd")
ustrfrom("caf"+char(233), "utf-8", 2) = "caf"
ustrfrom("caf"+char(233), "utf-8", 3) = ""
ustrfrom("caf"+char(233), "utf-8", 4) = "caf%XE9"
```

Domain s : strings in encoding enc
 Domain enc : Unicode strings
 Domain $mode$: integers
 Range: Unicode strings

ustrinvalidcnt(*s*)

Description: the number of invalid UTF-8 sequences in *s*

An invalid UTF-8 sequence may contain one byte or multiple bytes.

```
ustrinvalidcnt("médiane") = 0
ustrinvalidcnt("médiane"+char(229)) = 1
ustrinvalidcnt("médiane"+char(229)+char(174)) = 1
ustrinvalidcnt("médiane"+char(174)+char(158)) = 2
```

Domain *s*: Unicode strings

Range: integers

ustrleft(*s*,*n*)

Description: the first *n* Unicode characters of the Unicode string *s*

An invalid UTF-8 sequence is replaced with a Unicode replacement character `\ufffd`.

```
ustrleft("Экспериментальные",3) = "Экс"
ustrleft("Экспериментальные",5) = "Экспе"
```

Domain *s*: Unicode strings

Domain *n*: integers

Range: Unicode strings

ustrnormalize(*s*,*norm*)

Description: *normalizes* Unicode string *s* to one of the five normalization forms specified by *norm*

The normalization forms are `nfc`, `nfd`, `nfkc`, `nfkd`, or `nfkcc`. The function returns an empty string for any other value of *norm*. Unicode normalization removes the Unicode string differences caused by Unicode character equivalence. `nfc` specifies Normalization Form C, which normalizes decomposed Unicode code points to a composited form. `nfd` specifies Normalization Form D, which normalizes composited Unicode code points to a decomposed form. `nfc` and `nfd` produce canonical equivalent form. `nfkc` and `nfkd` are similar to `nfc` and `nfd` but produce compatibility equivalent forms. `nfkcc` specifies `nfkc` with casefolding. This normalization and casefolding implement the [Unicode Character Database](#).

In the Unicode standard, both “i” (`\u0069` followed by a diaeresis `\u0308`) and the composite character `\u00ef` represent “i” with 2 dots as in “naïve”. Hence, the code-point sequence `\u0069\u0308` and the code point `\u00ef` are considered Unicode equivalent. According to the Unicode standard, they should be treated as the same single character in Unicode string operations, such as in display, comparison, and selection. However, Stata does not support multiple code-point characters; each code point is considered a separate Unicode character. Hence, `\u0069\u0308` is displayed as two characters in the Results window. `ustrnormalize()` can be used with “`nfc`” to normalize `\u0069\u0308` to the canonical equivalent composited code point `\u00ef`.

```
ustrnormalize(ustrunescape("\u0069\u0308"), "nfc") = "i"
```

The decomposed form `nfd` can be used to removed diacritical marks from base letters. First, normalize the Unicode string to canonical decomposed form, and then call `ustrto()` with mode `skip` to skip all non-ASCII characters.

Also see `ustrfrom()`.

```
ustrto(ustrnormalize("café", "nfd"), "ascii", 2) = "cafe"
```

Domain *s*: Unicode strings

Domain *norm*: Unicode strings

Range: Unicode strings

`ustrright(s,n)`

Description: the last *n* Unicode characters of the Unicode string *s*

An invalid UTF-8 sequence is replaced with a Unicode replacement character `\ufffd`.

```
ustrright("Экспериментальные",3) = "ные"
```

```
ustrright("Экспериментальные",5) = "льные"
```

Domain *s*: Unicode strings

Domain *n*: integers

Range: Unicode strings

`ustrsortkey(s[,loc])`

Description: generates a null-terminated byte array that can be used by the `sort` command to produce the same order as `ustrcompare()`

The function may return an empty array if an error occurs. The result is locale dependent. If *loc* is not specified, the `default locale` is used. The result is also diacritic and case sensitive. If you need different behavior, for example, case-insensitive results, you should use the extended function `ustrsortkeyex()`. See [U] **12.4.2.5 Sorting strings containing Unicode characters** for details and examples.

Domain *s*: Unicode strings

Domain *loc*: Unicode strings

Range: null-terminated byte array

`ustrsortkeyex(s, loc, case, cslv, norm, num, alt, fr)`

Description: generates a null-terminated byte array that can be used by the `sort` command to produce the same order as `ustrcompare()`

The function may return an empty array if an error occurs. The result is locale dependent. If *loc* is not specified, the **default locale** is used. See [U] [12.4.2.5 Sorting strings containing Unicode characters](#) for details and examples.

st controls the strength of the comparison. Possible values are 1 (primary), 2 (secondary), 3 (tertiary), 4 (quaternary), or 5 (identical). -1 means to use the default value for the locale. Any other numbers are treated as tertiary. The primary difference represents base letter differences; for example, letter “a” and letter “b” have primary differences. The secondary difference represents diacritical differences on the same base letter; for example, letters “a” and “ä” have secondary differences. The tertiary difference represents case differences of the same base letters; for example, letters “a” and “A” have tertiary differences. Quaternary strength is useful to distinguish between Katakana and Hiragana for the JIS 4061 collation standard. Identical strength is essentially the code-point order of the string and, hence, is rarely useful.

case controls the uppercase and lowercase letter order. Possible values are 0 (use order specified in tertiary strength), 1 (uppercase first), or 2 (lowercase first). -1 means to use the default value for the locale. Any other values are treated as 0.

cslv controls if an extra case level between the secondary level and the tertiary level is generated. Possible values are 0 (off) or 1 (on). -1 means to use the default value for the locale. Any other values are treated as 0. Combining this setting to be “on” and the strength setting to be primary can achieve the effect of ignoring the diacritical differences but preserving the case differences. If the setting is “on”, the result is also affected by the *case* setting.

norm controls whether the normalization check and normalizations are performed. Possible values are 0 (off) or 1 (on). -1 means to use the default value for the locale. Any other values are treated as 0. Most languages do not require normalization for comparison. Normalization is needed in languages that use multiple combining characters such as Arabic, ancient Greek, or Hebrew.

num controls how contiguous digit substrings are sorted. Possible values are 0 (off) or 1 (on). -1 means to use the default value for the locale. Any other values are treated as 0. If the setting is “on”, substrings consisting of digits are sorted based on the numeric value. For example, “100” is after “20” instead of before it. Note that the digit substring is limited to 254 digits, and plus/minus signs, decimals, or exponents are not supported.

alt controls how spaces and punctuation characters are handled. Possible values are 0 (use primary strength) or 1 (alternative handling). Any other values are treated as 0. If the setting is 1 (alternative handling), “onsite”, “on-site”, and “on site” are considered equals.

fr controls the direction of the secondary strength. Possible values are 0 (off) or 1 (on). -1 means to use the default value for the locale. All other values are treated as “off”. If the setting is “on”, the diacritical letters are sorted backward. Note that the setting is “on” by default only for Canadian French (locale `fr_CA`).

Domain *s*: Unicode strings
 Domain *loc*: Unicode strings
 Domain *st*: integers
 Domain *case*: integers
 Domain *cslv*: integers
 Domain *norm*: integers
 Domain *num*: integers
 Domain *alt*: integers
 Domain *fr*: integers
 Range: null-terminated byte array

`ustrto(s, enc, mode)`

Description: converts the Unicode string *s* in UTF-8 encoding to a string in encoding *enc*

See [D] [unicode encoding](#) for details on available encodings. Any invalid sequence in *s* is replaced with a Unicode replacement character `\ufffd`. *mode* controls how unsupported Unicode characters in the encoding *enc* are handled. The possible values are 1, which substitutes any unsupported characters with the *enc*’s substitution strings (the substitution character for both `ascii` and `latin1` is `char(26)`); 2, which skips any unsupported characters; 3, which stops at the first unsupported character and returns an empty string; or 4, which replaces any unsupported character with an escaped hex digit sequence `\uhhhh` or `\Uhhhhhhhh`. The hex digit sequence contains either 4 or 8 hex digits, depending if the Unicode character’s code-point value is less than or greater than `\uffff`. Any other values are treated as 1.

```
ustrto("café", "ascii", 1) = "caf"+char(26)
ustrto("café", "ascii", 2) = "caf"
ustrto("café", "ascii", 3) = ""
ustrto("café", "ascii", 4) = "caf\u00E9"
```

`ustrto()` can be used to removed diacritical marks from base letters. First, normalize the Unicode string to NFD form using `ustrnormalize()`, and then call `ustrto()` with value 2 to skip all non-ASCII characters.

Also see `ustrfrom()`.

```
ustrto(ustrnormalize("café", "nfd"), "ascii", 2) = "cafe"
```

Domain *s*: Unicode strings
 Domain *enc*: Unicode strings
 Domain *mode*: integers
 Range: strings in encoding *enc*

ustrtohex(*s*[, *n*])Description: escaped hex digit string of *s* up to 200 Unicode characters

The escaped hex digit string is in the form of `\uhhhh` for code points less than `\uffff` or `\Uhhhhhhh` for code points greater than `\uffff`. The function starts at the *n*th Unicode character of *s* if *n* is specified and larger than 0. Any invalid UTF-8 sequence is replaced with a Unicode replacement character `\ufffd`. Note that the null terminator `char(0)` is a valid Unicode character. Function `ustrunescape()` can be applied on the result to get back the original Unicode string *s* if *s* does not contain any invalid UTF-8 sequences.

Also see `ustrunescape()`.

```
ustrtohex("нүлю") = "\u043d\u0443\u043b\u044e"  
ustrtohex("нүлю", 2) = "\u0443\u043b\u044e"  
ustrtohex("i"+char(200)+char(0)+"s") =  
"\u0069\uufffd\u0000\u0073"
```

Domain *s*: Unicode stringsDomain *n*: integers ≥ 1

Range: strings

ustrunescape(*s*)Description: the Unicode string corresponding to the escaped sequences of *s*

The following escape sequences are recognized: 4 hex digit form `\uhhhh`; 8 hex digit form `\Uhhhhhhh`; 1–2 hex digit form `\xhh`; and 1–3 octal digit form `\ooo`, where *h* is `[0-9A-Fa-f]` and *o* is `[0-7]`. The standard ANSI C escapes `\a`, `\b`, `\t`, `\n`, `\v`, `\f`, `\r`, `\e`, `\"`, `\'`, `\?`, `\\` are recognized as well. The function returns an empty string if an escape sequence is badly formed. Note that the 8 hex digit form `\Uhhhhhhh` begins with a capital letter “U”.

Also see `ustrtohex()`.

```
ustrunescape("\u043d\u0443\u043b\u044e") = "нүлю"
```

Domain *s*: strings of escaped hex values

Range: Unicode strings

word(*s*, *n*)Description: the *n*th word in *s*; *missing* ("") if *n* is missing

Positive numbers count words from the beginning of *s*, and negative numbers count words from the end of *s*. (1 is the first word in *s*, and -1 is the last word in *s*.) A word is a set of characters that start and terminate with spaces. This is different from a Unicode word, which is a language unit based on either a set of [word-boundary rules](#) or dictionaries for several languages (Chinese, Japanese, and Thai).

Domain *s*: stringsDomain *n*: integers

Range: strings

`ustrword(s,n[,loc])`

Description: the *n*th Unicode word in the Unicode string *s*

Positive *n* counts Unicode words from the beginning of *s*, and negative *n* counts Unicode words from the end of *s*. For examples, *n* equal to 1 returns the first word in *s*, and *n* equal to -1 returns the last word in *s*. If *loc* is not specified, the [default locale](#) is used. A Unicode word is different from a Stata word produced by the `word()` function. A Stata word is a space-separated token. A Unicode word is a language unit based on either a set of [word-boundary rules](#) or dictionaries for some languages (Chinese, Japanese, and Thai). The function returns *missing* ("") if *n* is greater than *cnt* or less than $-cnt$, where *cnt* is the number of words *s* contains. *cnt* can be obtained from `ustrwordcount()`. The function also returns *missing* ("") if an error occurs.

```
ustrword("Parlez-vous français", 1, "fr") = "Parlez"
ustrword("Parlez-vous français", 2, "fr") = "-"
ustrword("Parlez-vous français",-1, "fr") = "français"
ustrword("Parlez-vous français",-2, "fr") = "vous"
```

Domain *s*: Unicode strings

Domain *loc*: Unicode strings

Domain *n*: integers

Range: Unicode strings

`wordbreaklocale(loc,type)`

Description: the most closely related locale supported by ICU from *loc* if *type* is 1, the actual locale where the word-boundary analysis data come from if *type* is 2; or an empty string is returned for any other *type*

```
wordbreaklocale("en_us_texas", 1) = en_US
wordbreaklocale("en_us_texas", 2) = root
```

Domain *loc*: strings of locale name

Domain *type*: integers

Range: strings

`wordcount(s)`

Description: the number of words in *s*

A word is a set of characters that starts and terminates with spaces, starts with the beginning of the string, or terminates with the end of the string. This is different from a Unicode word, which is a language unit based on either a set of [word-boundary rules](#) or dictionaries for several languages (Chinese, Japanese, and Thai).

Domain *s*: strings

Range: nonnegative integers 0, 1, 2, ...

`ustrwordcount(s[, loc])`

Description: the number of nonempty Unicode words in the Unicode string *s*

An empty Unicode word is a Unicode word consisting of only Unicode whitespace characters. If *loc* is not specified, the [default locale](#) is used. A Unicode word is different from a Stata word produced by the `word()` function. A Stata word is a space-separated token. A Unicode word is a language unit based on either a set of [word-boundary rules](#) or dictionaries for some languages (Chinese, Japanese, and Thai). The function may return a negative number if an error occurs.

`ustrwordcount("Parlez-vous français", "fr") = 4`

Domain *s*: Unicode strings

Domain *loc*: Unicode strings

Range: integers

References

- Cox, N. J. 2004. [Stata tip 6: Inserting awkward characters in the plot](#). *Stata Journal* 4: 95–96.
- . 2011. [Stata tip 98: Counting substrings within strings](#). *Stata Journal* 11: 318–320.
- Jeanty, P. W. 2013. [Dealing with identifier variables in data management and analysis](#). *Stata Journal* 13: 699–718.
- Koplenig, A. 2018. [Stata tip 129: Efficiently processing textual data with Stata's new Unicode features](#). *Stata Journal* 18: 287–289.
- Schwarz, C. 2019. [lsemantic: A command for text similarity based on latent semantic analysis](#). *Stata Journal* 19: 129–142.

Also see

[FN] [Functions by category](#)

[D] [egen](#) — Extensions to generate

[D] [generate](#) — Create or change contents of variable

[M-4] [String](#) — String manipulation functions

[U] [12.4.2 Handling Unicode strings](#)

[U] [13.2.2 String operators](#)

[U] [13.3 Functions](#)

Contents

<code>acos(x)</code>	the radian value of the arccosine of x
<code>acosh(x)</code>	the inverse hyperbolic cosine of x
<code>asin(x)</code>	the radian value of the arcsine of x
<code>asinh(x)</code>	the inverse hyperbolic sine of x
<code>atan(x)</code>	the radian value of the arctangent of x
<code>atan2(y, x)</code>	the radian value of the arctangent of y/x , where the signs of the parameters y and x are used to determine the quadrant of the answer
<code>atanh(x)</code>	the inverse hyperbolic tangent of x
<code>cos(x)</code>	the cosine of x , where x is in radians
<code>cosh(x)</code>	the hyperbolic cosine of x
<code>sin(x)</code>	the sine of x , where x is in radians
<code>sinh(x)</code>	the hyperbolic sine of x
<code>tan(x)</code>	the tangent of x , where x is in radians
<code>tanh(x)</code>	the hyperbolic tangent of x

Functions

`acos(x)`
Description: the radian value of the arccosine of x
Domain: -1 to 1
Range: 0 to π

`acosh(x)`
Description: the inverse hyperbolic cosine of x
$$\operatorname{acosh}(x) = \ln(x + \sqrt{x^2 - 1})$$

Domain: 1 to $8.9\text{e}+307$
Range: 0 to 709.77

`asin(x)`
Description: the radian value of the arcsine of x
Domain: -1 to 1
Range: $-\pi/2$ to $\pi/2$

`asinh(x)`
Description: the inverse hyperbolic sine of x
$$\operatorname{asinh}(x) = \ln(x + \sqrt{x^2 + 1})$$

Domain: $-8.9\text{e}+307$ to $8.9\text{e}+307$
Range: -709.77 to 709.77

atan(x)Description: the radian value of the arctangent of x Domain: $-8\text{e}+307$ to $8\text{e}+307$ Range: $-\pi/2$ to $\pi/2$ **atan2(y, x)**Description: the radian value of the arctangent of y/x , where the signs of the parameters y and x are used to determine the quadrant of the answerDomain y : $-8\text{e}+307$ to $8\text{e}+307$ Domain x : $-8\text{e}+307$ to $8\text{e}+307$ Range: $-\pi$ to π **atanh(x)**Description: the inverse hyperbolic tangent of x

$$\operatorname{atanh}(x) = \frac{1}{2} \{ \ln(1+x) - \ln(1-x) \}$$

Domain: -1 to 1 Range: $-8\text{e}+307$ to $8\text{e}+307$ **cos(x)**Description: the cosine of x , where x is in radiansDomain: $-1\text{e}+18$ to $1\text{e}+18$ Range: -1 to 1 **cosh(x)**Description: the hyperbolic cosine of x

$$\cosh(x) = \{ \exp(x) + \exp(-x) \} / 2$$

Domain: -709 to 709 Range: 1 to $4.11\text{e}+307$ **sin(x)**Description: the sine of x , where x is in radiansDomain: $-1\text{e}+18$ to $1\text{e}+18$ Range: -1 to 1 **sinh(x)**Description: the hyperbolic sine of x

$$\sinh(x) = \{ \exp(x) - \exp(-x) \} / 2$$

Domain: -709 to 709 Range: $-4.11\text{e}+307$ to $4.11\text{e}+307$ **tan(x)**Description: the tangent of x , where x is in radiansDomain: $-1\text{e}+18$ to $1\text{e}+18$ Range: $-1\text{e}+17$ to $1\text{e}+17$ or *missing***tanh(x)**Description: the hyperbolic tangent of x

$$\tanh(x) = \{ \exp(x) - \exp(-x) \} / \{ \exp(x) + \exp(-x) \}$$

Domain: $-8\text{e}+307$ to $8\text{e}+307$ Range: -1 to 1 or *missing*

□ Technical note

The trigonometric functions are defined in terms of *radians*. There are 2π radians in a circle. If you prefer to think in terms of *degrees*, because there are also 360 degrees in a circle, you may convert degrees into radians by using the formula $r = d\pi/180$, where d represents degrees and r represents radians. Stata includes the built-in constant `_pi`, equal to π to machine precision. Thus, to calculate the sine of `theta`, where `theta` is measured in degrees, you could type

```
sin(theta*_pi/180)
```

`atan()` similarly returns radians, not degrees. The arccotangent can be obtained as

```
acot(x) = _pi/2 - atan(x)
```



Reference

Oldham, K. B., J. C. Myland, and J. Spanier. 2009. *An Atlas of Functions*. 2nd ed. New York: Springer.

Also see

[FN] [Functions by category](#)

[D] [egen](#) — Extensions to generate

[D] [generate](#) — Create or change contents of variable

[M-5] [sin\(\)](#) — Trigonometric and hyperbolic functions

[U] [13.3 Functions](#)

Subject and author index

See the [combined subject index](#) and the [combined author index](#) in the *Glossary and Index*.