STATA FUNCTIONS REFERENCE MANUAL

RELEASE 17

A Stata Press Publication
StataCorp LLC
College Station, Texas

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

ISBN-10: 1-59718-330-X
ISBN-13: 978-1-59718-330-7

The suggested citation for this software is

StataCorp. 2021. *Stata: Release 17*. Statistical Software. College Station, TX: StataCorp LLC.

Contents

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

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**

Title

Functions by category

Contents

Date and time functions

`age(`$e_{d\,\mathrm{DOB}}$`,`e_d [`,`s_{nl}]`)`	the age in integer years on e_d for date of birth $e_{d\,\mathrm{DOB}}$ with s_{nl} the nonleap-year birthday for 29feb birthdates
`age_frac(`$e_{d\,\mathrm{DOB}}$`,`e_d [`,`s_{nl}]`)`	the age in years, including the fractional part, on e_d for date of birth $e_{d\,\mathrm{DOB}}$ with s_{nl} the nonleap-year birthday for 29feb birthdates
`birthday(`$e_{d\,\mathrm{DOB}}$`,`Y [`,`s_{nl}]`)`	the e_d date of the birthday in year Y for date of birth $e_{d\,\mathrm{DOB}}$ with s_{nl} the nonleap-year birthday for 29feb birthdates
`bofd("`*cal*`",`e_d`)`	the e_b business date corresponding to e_d
`Cdhms(`e_d`,`h`,`m`,`s`)`	the e_{tC} datetime (ms. with leap seconds since 01jan1960 00:00:00.000) corresponding to e_d, h, m, s
`Chms(`h`,`m`,`s`)`	the e_{tC} datetime (ms. with leap seconds since 01jan1960 00:00:00.000) corresponding to h, m, s on 01jan1960
`Clock(`s_1`,`s_2 [`,`Y]`)`	the e_{tC} datetime (ms. with leap seconds since 01jan1960 00:00:00.000) corresponding to s_1 based on s_2 and Y
`clock(`s_1`,`s_2 [`,`Y]`)`	the e_{tc} datetime (ms. since 01jan1960 00:00:00.000) corresponding to s_1 based on s_2 and Y
`Clockdiff(`e_{tC1}`, `e_{tC2}`,`s_u`)`	the e_{tC} datetime difference, rounded down to an integer, from e_{tC1} to e_{tC2} in s_u units of days, hours, minutes, seconds, or milliseconds
`clockdiff(`e_{tc1}`,`e_{tc2}`,`s_u`)`	the e_{tc} datetime difference, rounded down to an integer, from e_{tc1} to e_{tc2} in s_u units of days, hours, minutes, seconds, or milliseconds
`Clockdiff_frac(`e_{tC1}`,`e_{tC2}`,`s_u`)`	the e_{tC} datetime difference, including the fractional part, from e_{tC1} to e_{tC2} in s_u units of days, hours, minutes, seconds, or milliseconds
`clockdiff_frac(`e_{tc1}`,`e_{tc2}`,`s_u`)`	the e_{tc} datetime difference, including the fractional part, from e_{tc1} to e_{tc2} in s_u units of days, hours, minutes, seconds, or milliseconds
`Clockpart(`e_{tC}`,`s_u`)`	the integer year, month, day, hour, minute, second, or millisecond of e_{tC} with s_u specifying which time part
`clockpart(`e_{tc}`,`s_u`)`	the integer year, month, day, hour, minute, second, or millisecond of e_{tc} with s_u specifying which time part

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

`Cofc(`e_{tc}`)` 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)

`cofC(`e_{tC}`)` 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)

`Cofd(`e_d`)` the e_{tC} datetime (ms. with leap seconds since 01jan1960 00:00:00.000) of date e_d at time 00:00:00.000

`cofd(`e_d`)` the e_{tc} datetime (ms. since 01jan1960 00:00:00.000) of date e_d at time 00:00:00.000

`daily(`s_1`,`s_2`[,`Y`])` a synonym for `date(`s_1`,`s_2`[,`Y`])`

`date(`s_1`,`s_2`[,`Y`])` the e_d date (days since 01jan1960) corresponding to s_1 based on s_2 and Y

`datediff(`e_{d1}`,`e_{d2}`,`s_u`[,`s_{nl}`])` the difference, rounded down to an integer, from e_{d1} to e_{d2} in s_u units of days, months, or years with s_{nl} the nonleap-year anniversary for e_{d1} on 29feb

`datediff_frac(`e_{d1}`,`e_{d2}`,`s_u`[,`s_{nl}`])` the difference, including the fractional part, from e_{d1} to e_{d2} in s_u units of days, months, or years with s_{nl} the nonleap-year anniversary for e_{d1} on 29feb

`datepart(`e_d`,`s_u`)` the integer year, month, or day of e_d with s_u specifying year, month, or day

`day(`e_d`)` the numeric day of the month corresponding to e_d

`daysinmonth(`e_d`)` the number of days in the month of e_d

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

`dofb(`e_b`,"`*cal*`")` the e_d datetime corresponding to e_b

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

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

`dofh(`e_h`)` the e_d date (days since 01jan1960) of the start of half-year e_h

`dofm(`e_m`)` the e_d date (days since 01jan1960) of the start of month e_m

`dofq(`e_q`)` the e_d date (days since 01jan1960) of the start of quarter e_q

`dofw(`e_w`)` the e_d date (days since 01jan1960) of the start of week e_w

`dofy(`e_y`)` the e_d date (days since 01jan1960) of 01jan in year e_y

`dow(`e_d`)` the numeric day of the week corresponding to date e_d; $0 =$ Sunday, $1 =$ Monday, $\dots$, $6 =$ Saturday

`doy(`e_d`)` the numeric day of the year corresponding to date e_d

`firstdayofmonth(`e_d`)` the e_d date of the first day of the month of e_d

`halfyear(`e_d`)` the numeric half of the year corresponding to date e_d

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

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

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

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

hofd(e_d) — the e_h half-yearly date (half years since 1960h1) containing date e_d

hours(ms) — $ms/3{,}600{,}000$

isleapsecond(e_{tC}) — `1` if e_{tC} is a leap second; otherwise, `0`

isleapyear(Y) — `1` if Y is a leap year; otherwise, `0`

lastdayofmonth(e_d) — the e_d date of the last day of the month of e_d

mdy(M,D,Y) — the e_d date (days since 01jan1960) corresponding to M, D, Y

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

minutes(ms) — $ms/60{,}000$

mm(e_{tc}) — the minute corresponding to datetime e_{tc} (ms. since 01jan1960 00:00:00.000)

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

mofd(e_d) — the e_m monthly date (months since 1960m1) containing date e_d

month(e_d) — the numeric month corresponding to date e_d

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

msofhours(h) — $h \times 3{,}600{,}000$

msofminutes(m) — $m \times 60{,}000$

msofseconds(s) — $s \times 1{,}000$

nextbirthday($e_{d\,\mathrm{DOB}}$,e_d[,s_{nl}]) — the e_d date of the first birthday after e_d for date of birth $e_{d\,\mathrm{DOB}}$ with s_{nl} the nonleap-year birthday for 29feb birthdates

nextleapyear(Y) — the first leap year after year Y

now() — the current e_{tc} datetime

previousbirthday($e_{d\,\mathrm{DOB}}$,e_d[,s_{nl}]) — the e_d date of the birthday immediately before e_d for date of birth $e_{d\,\mathrm{DOB}}$ with s_{nl} the nonleap-year birthday for 29feb birthdates

previousleapyear(Y) — the leap year immediately before year Y

qofd(e_d) — the e_q quarterly date (quarters since 1960q1) containing date e_d

quarter(e_d) — the numeric quarter of the year corresponding to date e_d

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

seconds(ms) — $ms/1{,}000$

ss(e_{tc}) — the second corresponding to datetime e_{tc} (ms. since 01jan1960 00:00:00.000)

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

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

Mathematical functions

`abs(`x`)`	the absolute value of x
`ceil(`x`)`	the unique integer n such that $n-1<x\le n$; x (not ".") if x is missing, meaning that `ceil(.a)` $=$ `.a`
`cloglog(`x`)`	the complementary log-log of x
`comb(`n`,`k`)`	the combinatorial function $n!/\{k!(n-k)!\}$
`digamma(`x`)`	the `digamma()` function, $d\ln\Gamma(x)/dx$
`exp(`x`)`	the exponential function e^x
`expm1(`x`)`	e^x-1 with higher precision than `exp(`x`)` -1 for small values of $\lvert x\rvert$
`floor(`x`)`	the unique integer n such that $n\le x<n+1$; x (not ".") if x is missing, meaning that `floor(.a)` $=$ `.a`

`int(`x`)`	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`
`invcloglog(`x`)`	the inverse of the complementary log-log function of x
`invlogit(`x`)`	the inverse of the logit function of x
`ln(`x`)`	the natural logarithm, $\ln(x)$
`ln1m(`x`)`	the natural logarithm of $1-x$ with higher precision than `ln(`$1-x$`)` for small values of $\|x\|$
`ln1p(`x`)`	the natural logarithm of $1+x$ with higher precision than `ln(`$1+x$`)` for small values of $\|x\|$
`lnfactorial(`n`)`	the natural log of n factorial = $\ln(n!)$
`lngamma(`x`)`	$\ln\{\Gamma(x)\}$
`log(`x`)`	a synonym for `ln(`x`)`
`log10(`x`)`	the base-10 logarithm of x
`log1m(`x`)`	a synonym for `ln1m(`x`)`
`log1p(`x`)`	a synonym for `ln1p(`x`)`
`logit(`x`)`	the log of the odds ratio of x, `logit(`x`)` $= \ln\{x/(1-x)\}$
`max(`x_1`,`x_2`,...,`x_n`)`	the maximum value of $x_1, x_2, \ldots, x_n$
`min(`x_1`,`x_2`,...,`x_n`)`	the minimum value of $x_1, x_2, \ldots, x_n$
`mod(`x`,`y`)`	the modulus of x with respect to y
`reldif(`x`,`y`)`	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*
`round(`x`,`y`)` or `round(`x`)`	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
`sign(`x`)`	the sign of x: -1 if $x<0$, 0 if $x=0$, 1 if $x>0$, or *missing* if x is missing
`sqrt(`x`)`	the square root of x
`sum(`x`)`	the running sum of x, treating missing values as zero
`trigamma(`x`)`	the second derivative of `lngamma(`x`)` = $d^2 \ln\Gamma(x)/dx^2$
`trunc(`x`)`	a synonym for `int(`x`)`

Matrix functions

`cholesky(`M`)`	the Cholesky decomposition of the matrix: if $R =$ `cholesky(`S`)`, then $RR^T = S$
`coleqnumb(`M`,`s`)`	the equation number of M associated with column equation s; *missing* if the column equation cannot be found
`colnfreeparms(`M`)`	the number of free parameters in columns of M
`colnumb(`M`,`s`)`	the column number of M associated with column name s; *missing* if the column cannot be found

colsof(M)	the number of columns of M
corr(M)	the correlation matrix of the variance matrix
det(M)	the determinant of matrix M
diag(M)	the square, diagonal matrix created from the row or column vector
diag0cnt(M)	the number of zeros on the diagonal of M
el(s,i,j)	s[floor(i),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
get(*systemname*)	a copy of Stata internal system matrix *systemname*
hadamard(M,N)	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)
I(n)	an $n \times n$ identity matrix if n is an integer; otherwise, a round(n) $\times$ round(n) identity matrix
inv(M)	the inverse of the matrix M
invsym(M)	the inverse of M if M is positive definite
issymmetric(M)	1 if the matrix is symmetric; otherwise, 0
J(r,c,z)	the $r \times c$ matrix containing elements z
matmissing(M)	1 if any elements of the matrix are missing; otherwise, 0
matuniform(r,c)	the $r \times c$ matrices containing uniformly distributed pseudorandom numbers on the interval $(0, 1)$
mreldif(X,Y)	the relative difference of X and Y, where the relative difference is defined as $\max_{i,j}\{\lvert x_{ij} - y_{ij}\rvert/(\lvert y_{ij}\rvert + 1)\}$
nullmat(*matname*)	use with the row-join (,) and column-join (\) operators
roweqnumb(M,s)	the equation number of M associated with row equation s; *missing* if the row equation cannot be found
rownfreeparms(M)	the number of free parameters in rows of M
rownumb(M,s)	the row number of M associated with row name s; *missing* if the row cannot be found
rowsof(M)	the number of rows of M
sweep(M,i)	matrix M with ith row/column swept
trace(M)	the trace of matrix M
vec(M)	a column vector formed by listing the elements of M, starting with the first column and proceeding column by column
vecdiag(M)	the row vector containing the diagonal of matrix M

Programming functions

autocode(x,n,x_0,x_1)	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 or the upper bound of the first or last interval if $x < x_0$ or $x > x_1$, respectively
byteorder()	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
c(*name*)	the value of the system or constant result c(*name*) (see [P] **creturn**)

`_caller()`	`version` of the program or session that invoked the currently running program; see [P] **version**
`chop(`x`, `ϵ`)`	`round(`x`)` if `abs(`x $-$ `round(`x`))` $< \epsilon$; otherwise, x; or x if x is missing
`clip(`x`,`a`,`b`)`	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
`cond(`x`,`a`,`b[`,`c]`)`	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*
`e(`*name*`)`	the value of stored result `e(`*name*`)`; see [U] **18.8 Accessing results calculated by other programs**
`e(sample)`	`1` if the observation is in the estimation sample and `0` otherwise
`epsdouble()`	the machine precision of a double-precision number
`epsfloat()`	the machine precision of a floating-point number
`fileexists(`f`)`	`1` if the file specified by f exists; otherwise, `0`
`fileread(`f`)`	the contents of the file specified by f
`filereaderror(`s`)`	`0` or positive integer, said value having the interpretation of a return code
`filewrite(`f`,`s[`,`r]`)`	writes the string specified by s to the file specified by f and returns the number of bytes in the resulting file
`float(`x`)`	the value of x rounded to `float` precision
`fmtwidth(`*fmtstr*`)`	the output length of the %*fmt* contained in *fmtstr*; *missing* if *fmtstr* does not contain a valid %*fmt*
`frval()`	returns values of variables stored in other frames
`_frval()`	programmer's version of `frval()`
`has_eprop(`*name*`)`	`1` if *name* appears as a word in `e(properties)`; otherwise, `0`
`inlist(`z`,`a`,`b`,...)`	`1` if z is a member of the remaining arguments; otherwise, `0`
`inrange(`z`,`a`,`b`)`	`1` if it is known that $a \leq z \leq b$; otherwise, `0`
`irecode(`x`,`x_1`,...,`x_n`)`	*missing* if x is missing or $x_1, \ldots, x_n$ 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 $x > x_n$
`matrix(`*exp*`)`	restricts name interpretation to scalars and matrices; see `scalar()`
`maxbyte()`	the largest value that can be stored in storage type `byte`
`maxdouble()`	the largest value that can be stored in storage type `double`
`maxfloat()`	the largest value that can be stored in storage type `float`
`maxint()`	the largest value that can be stored in storage type `int`
`maxlong()`	the largest value that can be stored in storage type `long`
`mi(`x_1`,`x_2`,...,`x_n`)`	a synonym for `missing(`x_1`,`x_2`,...,`x_n`)`
`minbyte()`	the smallest value that can be stored in storage type `byte`
`mindouble()`	the smallest value that can be stored in storage type `double`
`minfloat()`	the smallest value that can be stored in storage type `float`
`minint()`	the smallest value that can be stored in storage type `int`
`minlong()`	the smallest value that can be stored in storage type `long`
`missing(`x_1`,`x_2`,...,`x_n`)`	`1` if any x_i evaluates to *missing*; otherwise, `0`

`r(`*name*`)` — the value of the stored result `r(`*name*`)`; see **[U] 18.8 Accessing results calculated by other programs**

`recode(`x`,`x_1`,`...`,`x_n`)` — *missing* if $x_1, x_2, \ldots, x_n$ is not weakly increasing; x if x is missing; x_1 if $x \le x_1$; x_2 if $x \le x_2$, ...; otherwise, x_n if $x > x_1$, x_2, ..., x_{n-1}. $x_i \ge .$ is interpreted as $x_i = +\infty$

`replay()` — `1` if the first nonblank character of local macro `` `0' `` is a comma, or if `` `0' `` is empty

`return(`*name*`)` — the value of the to-be-stored result `r(`*name*`)`; see **[P] return**

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

`scalar(`*exp*`)` — restricts name interpretation to scalars and matrices

`smallestdouble()` — the smallest double-precision number greater than zero

Random-number functions

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

`rbinomial(`n`,`p`)` — binomial(n,p) random variates, where n is the number of trials and p is the success probability

`rcauchy(`a`,`b`)` — Cauchy(a,b) random variates, where a is the location parameter and b is the scale parameter

`rchi2(`*df*`)` — χ^2, with *df* degrees of freedom, random variates

`rexponential(`b`)` — exponential random variates with scale b

`rgamma(`a`,`b`)` — gamma(a,b) random variates, where a is the gamma shape parameter and b is the scale parameter

`rhypergeometric(`N`,`K`,`n`)` — hypergeometric random variates

`rigaussian(`m`,`a`)` — inverse Gaussian random variates with mean m and shape parameter a

`rlaplace(`m`,`b`)` — Laplace(m,b) random variates with mean m and scale parameter b

`rlogistic()` — logistic variates with mean 0 and standard deviation $\pi/\sqrt{3}$

`rlogistic(`s`)` — logistic variates with mean 0, scale s, and standard deviation $s\pi/\sqrt{3}$

`rlogistic(`m`,`s`)` — logistic variates with mean m, scale s, and standard deviation $s\pi/\sqrt{3}$

`rnbinomial(`n`,`p`)` — negative binomial random variates

`rnormal()` — standard normal (Gaussian) random variates, that is, variates from a normal distribution with a mean of 0 and a standard deviation of 1

`rnormal(`m`)` — normal(m,1) (Gaussian) random variates, where m is the mean and the standard deviation is 1

`rnormal(`m`,`s`)` — normal(m,s) (Gaussian) random variates, where m is the mean and s is the standard deviation

`rpoisson(`m`)` — Poisson(m) random variates, where m is the distribution mean

`rt(`*df*`)` — Student's t random variates, where *df* is the degrees of freedom

`runiform()` — uniformly distributed random variates over the interval $(0, 1)$

`runiform(`a`,`b`)` — uniformly distributed random variates over the interval (a, b)

runiformint(a,b)	uniformly distributed random integer variates on the interval $[a, b]$
rweibull(a,b)	Weibull variates with shape a and scale b
rweibull(a,b,g)	Weibull variates with shape a, scale b, and location g
rweibullph(a,b)	Weibull (proportional hazards) variates with shape a and scale b
rweibullph(a,b,g)	Weibull (proportional hazards) variates with shape a, scale b, and location g

Selecting time-span functions

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

Statistical functions

betaden(a,b,x)	the probability density of the beta distribution, where a and b are the shape parameters; 0 if $x < 0$ or $x > 1$
binomial(n,k,θ)	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 $k > n$
binomialp(n,k,p)	the probability of observing floor(k) successes in floor(n) trials when the probability of a success on one trial is p
binomialtail(n,k,θ)	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 $k > n$
binormal(h,k,ρ)	the joint cumulative distribution $\Phi(h, k, \rho)$ of bivariate normal with correlation ρ
cauchy(a,b,x)	the cumulative Cauchy distribution with location parameter a and scale parameter b
cauchyden(a,b,x)	the probability density of the Cauchy distribution with location parameter a and scale parameter b
cauchytail(a,b,x)	the reverse cumulative (upper tail or survivor) Cauchy distribution with location parameter a and scale parameter b
chi2(df,x)	the cumulative χ^2 distribution with df degrees of freedom; 0 if $x < 0$
chi2den(df,x)	the probability density of the χ^2 distribution with df degrees of freedom; 0 if $x < 0$
chi2tail(df,x)	the reverse cumulative (upper tail or survivor) χ^2 distribution with df degrees of freedom; 1 if $x < 0$
dgammapda(a,x)	$\frac{\partial P(a,x)}{\partial a}$, where $P(a, x) =$ gammap(a,x); 0 if $x < 0$
dgammapdada(a,x)	$\frac{\partial^2 P(a,x)}{\partial a^2}$, where $P(a, x) =$ gammap(a,x); 0 if $x < 0$
dgammapdadx(a,x)	$\frac{\partial^2 P(a,x)}{\partial a \partial x}$, where $P(a, x) =$ gammap(a,x); 0 if $x < 0$
dgammapdx(a,x)	$\frac{\partial P(a,x)}{\partial x}$, where $P(a, x) =$ gammap(a,x); 0 if $x < 0$
dgammapdxdx(a,x)	$\frac{\partial^2 P(a,x)}{\partial x^2}$, where $P(a, x) =$ gammap(a,x); 0 if $x < 0$

dunnettprob(k,df,x)	the cumulative multiple range distribution that is used in Dunnett's multiple-comparison method with k ranges and df degrees of freedom; 0 if $x < 0$
exponential(b,x)	the cumulative exponential distribution with scale b
exponentialden(b,x)	the probability density function of the exponential distribution with scale b
exponentialtail(b,x)	the reverse cumulative exponential distribution with scale b
F(df_1,df_2,f)	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 $f < 0$
Fden(df_1,df_2,f)	the probability density function of the F distribution with df_1 numerator and df_2 denominator degrees of freedom; 0 if $f < 0$
Ftail(df_1,df_2,f)	the reverse cumulative (upper tail or survivor) F distribution with df_1 numerator and df_2 denominator degrees of freedom; 1 if $f < 0$
gammaden(a,b,g,x)	the probability density function of the gamma distribution; 0 if $x < g$
gammap(a,x)	the cumulative gamma distribution with shape parameter a; 0 if $x < 0$
gammaptail(a,x)	the reverse cumulative (upper tail or survivor) gamma distribution with shape parameter a; 1 if $x < 0$
hypergeometric(N,K,n,k)	the cumulative probability of the hypergeometric distribution
hypergeometricp(N,K,n,k)	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
ibeta(a,b,x)	the cumulative beta distribution with shape parameters a and b; 0 if $x < 0$; or 1 if $x > 1$
ibetatail(a,b,x)	the reverse cumulative (upper tail or survivor) beta distribution with shape parameters a and b; 1 if $x < 0$; or 0 if $x > 1$
igaussian(m,a,x)	the cumulative inverse Gaussian distribution with mean m and shape parameter a; 0 if $x \leq 0$
igaussianden(m,a,x)	the probability density of the inverse Gaussian distribution with mean m and shape parameter a; 0 if $x \leq 0$
igaussiantail(m,a,x)	the reverse cumulative (upper tail or survivor) inverse Gaussian distribution with mean m and shape parameter a; 1 if $x \leq 0$
invbinomial(n,k,p)	the inverse of the cumulative binomial; that is, θ ($\theta =$ probability of success on one trial) such that the probability of observing floor(k) or fewer successes in floor(n) trials is p
invbinomialtail(n,k,p)	the inverse of the right cumulative binomial; that is, θ ($\theta =$ probability of success on one trial) such that the probability of observing floor(k) or more successes in floor(n) trials is p
invcauchy(a,b,p)	the inverse of cauchy(): if cauchy(a,b,x) $= p$, then invcauchy(a,b,p) $= x$
invcauchytail(a,b,p)	the inverse of cauchytail(): if cauchytail(a,b,x) $= p$, then invcauchytail(a,b,p) $= x$
invchi2(df,p)	the inverse of chi2(): if chi2(df,x) $= p$, then invchi2(df,p) $= x$
invchi2tail(df,p)	the inverse of chi2tail(): if chi2tail(df,x) $= p$, then invchi2tail(df,p) $= x$

invdunnettprob(k,df,p) — the inverse cumulative multiple range distribution that is used in Dunnett's multiple-comparison method with k ranges and df degrees of freedom

invexponential(b,p) — the inverse cumulative exponential distribution with scale b: if exponential(b,x) $= p$, then invexponential(b,p) $= x$

invexponentialtail(b,p) — the inverse reverse cumulative exponential distribution with scale b: if exponentialtail(b,x) $= p$, then invexponentialtail(b,p) $= x$

invF(df_1,df_2,p) — the inverse cumulative F distribution: if F(df_1,df_2,f) $= p$, then invF(df_1,df_2,p) $= f$

invFtail(df_1,df_2,p) — 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) $= f$

invgammap(a,p) — the inverse cumulative gamma distribution: if gammap(a,x) $= p$, then invgammap(a,p) $= x$

invgammaptail(a,p) — the inverse reverse cumulative (upper tail or survivor) gamma distribution: if gammaptail(a,x) $= p$, then invgammaptail(a,p) $= x$

invibeta(a,b,p) — the inverse cumulative beta distribution: if ibeta(a,b,x) $= p$, then invibeta(a,b,p) $= x$

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

invigaussian(m,a,p) — the inverse of igaussian(): if igaussian(m,a,x) $= p$, then invigaussian(m,a,p) $= x$

invigaussiantail(m,a,p) — the inverse of igaussiantail(): if igaussiantail(m,a,x) $= p$, then invigaussiantail(m,a,p) $= x$

invlaplace(m,b,p) — the inverse of laplace(): if laplace(m,b,x) $= p$, then invlaplace(m,b,p) $= x$

invlaplacetail(m,b,p) — the inverse of laplacetail(): if laplacetail(m,b,x) $= p$, then invlaplacetail(m,b,p) $= x$

invlogistic(p) — the inverse cumulative logistic distribution: if logistic(x) $= p$, then invlogistic(p) $= x$

invlogistic(s,p) — the inverse cumulative logistic distribution: if logistic(s,x) $= p$, then invlogistic(s,p) $= x$

invlogistic(m,s,p) — the inverse cumulative logistic distribution: if logistic(m,s,x) $= p$, then invlogistic(m,s,p) $= x$

invlogistictail(p) — the inverse reverse cumulative logistic distribution: if logistictail(x) $= p$, then invlogistictail(p) $= x$

invlogistictail(s,p) — the inverse reverse cumulative logistic distribution: if logistictail(s,x) $= p$, then invlogistictail(s,p) $= x$

invlogistictail(m,s,p) — the inverse reverse cumulative logistic distribution: if logistictail(m,s,x) $= p$, then invlogistictail(m,s,p) $= x$

invnbinomial(n,k,q) — the value of the negative binomial parameter, p, such that $q =$ nbinomial(n,k,p)

invnbinomialtail(n,k,q) — the value of the negative binomial parameter, p, such that $q =$ nbinomialtail(n,k,p)

invnchi2(df,np,p)	the inverse cumulative noncentral χ^2 distribution: if nchi2(df,np,x) $= p$, then invnchi2(df,np,p) $= x$
invnchi2tail(df,np,p)	the inverse reverse cumulative (upper tail or survivor) noncentral χ^2 distribution: if nchi2tail(df,np,x) $= p$, then invnchi2tail(df,np,p) $= x$
invnF(df_1,df_2,np,p)	the inverse cumulative noncentral F distribution: if nF(df_1,df_2,np,f) $= p$, then invnF(df_1,df_2,np,p) $= f$
invnFtail(df_1,df_2,np,p)	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) $= f$
invnibeta(a,b,np,p)	the inverse cumulative noncentral beta distribution: if nibeta(a,b,np,x) $= p$, then invibeta(a,b,np,p) $= x$
invnormal(p)	the inverse cumulative standard normal distribution: if normal(z) $= p$, then invnormal(p) $= z$
invnt(df,np,p)	the inverse cumulative noncentral Student's t distribution: if nt(df,np,t) $= p$, then invnt(df,np,p) $= t$
invnttail(df,np,p)	the inverse reverse cumulative (upper tail or survivor) noncentral Student's t distribution: if nttail(df,np,t) $= p$, then invnttail(df,np,p) $= t$
invpoisson(k,p)	the Poisson mean such that the cumulative Poisson distribution evaluated at k is p: if poisson(m,k) $= p$, then invpoisson(k,p) $= m$
invpoissontail(k,q)	the Poisson mean such that the reverse cumulative Poisson distribution evaluated at k is q: if poissontail(m,k) $= q$, then invpoissontail(k,q) $= m$
invt(df,p)	the inverse cumulative Student's t distribution: if t(df,t) $= p$, then invt(df,p) $= t$
invttail(df,p)	the inverse reverse cumulative (upper tail or survivor) Student's t distribution: if ttail(df,t) $= p$, then invttail(df,p) $= t$
invtukeyprob(k,df,p)	the inverse cumulative Tukey's Studentized range distribution with k ranges and df degrees of freedom
invweibull(a,b,p)	the inverse cumulative Weibull distribution with shape a and scale b: if weibull(a,b,x) $= p$, then invweibull(a,b,p) $= x$
invweibull(a,b,g,p)	the inverse cumulative Weibull distribution with shape a, scale b, and location g: if weibull(a,b,g,x) $= p$, then invweibull(a,b,g,p) $= x$
invweibullph(a,b,p)	the inverse cumulative Weibull (proportional hazards) distribution with shape a and scale b: if weibullph(a,b,x) $= p$, then invweibullph(a,b,p) $= x$
invweibullph(a,b,g,p)	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) $= x$
invweibullphtail(a,b,p)	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) $= x$

`invweibullphtail(`a`,`b`,`g`,`p`)`	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`)` $= x$
`invweibulltail(`a`,`b`,`p`)`	the inverse reverse cumulative Weibull distribution with shape a and scale b: if `weibulltail(`a`,`b`,`x`)` $= p$, then `invweibulltail(`a`,`b`,`p`)` $= x$
`invweibulltail(`a`,`b`,`g`,`p`)`	the inverse reverse cumulative Weibull distribution with shape a, scale b, and location g: if `weibulltail(`a`,`b`,`g`,`x`)` $= p$, then `invweibulltail(`a`,`b`,`g`,`p`)` $= x$
`laplace(`m`,`b`,`x`)`	the cumulative Laplace distribution with mean m and scale parameter b
`laplaceden(`m`,`b`,`x`)`	the probability density of the Laplace distribution with mean m and scale parameter b
`laplacetail(`m`,`b`,`x`)`	the reverse cumulative (upper tail or survivor) Laplace distribution with mean m and scale parameter b
`lncauchyden(`a`,`b`,`x`)`	the natural logarithm of the density of the Cauchy distribution with location parameter a and scale parameter b
`lnigammaden(`a`,`b`,`x`)`	the natural logarithm of the inverse gamma density, where a is the shape parameter and b is the scale parameter
`lnigaussianden(`m`,`a`,`x`)`	the natural logarithm of the inverse Gaussian density with mean m and shape parameter a
`lniwishartden(`df`,`V`,`X`)`	the natural logarithm of the density of the inverse Wishart distribution; missing if $df \le n - 1$
`lnlaplaceden(`m`,`b`,`x`)`	the natural logarithm of the density of the Laplace distribution with mean m and scale parameter b
`lnmvnormalden(`M`,`V`,`X`)`	the natural logarithm of the multivariate normal density
`lnnormal(`z`)`	the natural logarithm of the cumulative standard normal distribution
`lnnormalden(`z`)`	the natural logarithm of the standard normal density, $N(0, 1)$
`lnnormalden(`x`,`σ`)`	the natural logarithm of the normal density with mean 0 and standard deviation σ
`lnnormalden(`x`,`μ`,`σ`)`	the natural logarithm of the normal density with mean μ and standard deviation σ, $N(\mu, \sigma^2)$
`lnwishartden(`df`,`V`,`X`)`	the natural logarithm of the density of the Wishart distribution; missing if $df \le n - 1$
`logistic(`x`)`	the cumulative logistic distribution with mean 0 and standard deviation $\pi/\sqrt{3}$
`logistic(`s`,`x`)`	the cumulative logistic distribution with mean 0, scale s, and standard deviation $s\pi/\sqrt{3}$
`logistic(`m`,`s`,`x`)`	the cumulative logistic distribution with mean m, scale s, and standard deviation $s\pi/\sqrt{3}$
`logisticden(`x`)`	the density of the logistic distribution with mean 0 and standard deviation $\pi/\sqrt{3}$
`logisticden(`s`,`x`)`	the density of the logistic distribution with mean 0, scale s, and standard deviation $s\pi/\sqrt{3}$
`logisticden(`m`,`s`,`x`)`	the density of the logistic distribution with mean m, scale s, and standard deviation $s\pi/\sqrt{3}$

logistictail(x)	the reverse cumulative logistic distribution with mean 0 and standard deviation $\pi/\sqrt{3}$
logistictail(s,x)	the reverse cumulative logistic distribution with mean 0, scale s, and standard deviation $s\pi/\sqrt{3}$
logistictail(m,s,x)	the reverse cumulative logistic distribution with mean m, scale s, and standard deviation $s\pi/\sqrt{3}$
nbetaden(a,b,np,x)	the probability density function of the noncentral beta distribution; 0 if $x < 0$ or $x > 1$
nbinomial(n,k,p)	the cumulative probability of the negative binomial distribution
nbinomialp(n,k,p)	the negative binomial probability
nbinomialtail(n,k,p)	the reverse cumulative probability of the negative binomial distribution
nchi2(df,np,x)	the cumulative noncentral χ^2 distribution; 0 if $x < 0$
nchi2den(df,np,x)	the probability density of the noncentral χ^2 distribution; 0 if $x < 0$
nchi2tail(df,np,x)	the reverse cumulative (upper tail or survivor) noncentral χ^2 distribution; 1 if $x < 0$
nF(df_1,df_2,np,f)	the cumulative 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,np,f)	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 $f < 0$
nFtail(df_1,df_2,np,f)	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 $f < 0$
nibeta(a,b,np,x)	the cumulative noncentral beta distribution; 0 if $x < 0$; or 1 if $x > 1$
normal(z)	the cumulative standard normal distribution
normalden(z)	the standard normal density, $N(0,1)$
normalden(x,σ)	the normal density with mean 0 and standard deviation σ
normalden(x,μ,σ)	the normal density with mean μ and standard deviation σ, $N(\mu,\sigma^2)$
npnchi2(df,x,p)	the noncentrality parameter, np, for noncentral χ^2: if nchi2(df,np,x) $= p$, then npnchi2(df,x,p) $= np$
npnF(df_1,df_2,f,p)	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) $= np$
npnt(df,t,p)	the noncentrality parameter, np, for the noncentral Student's t distribution: if nt(df,np,t) $= p$, then npnt(df,t,p) $= np$
nt(df,np,t)	the cumulative noncentral Student's t distribution with df degrees of freedom and noncentrality parameter np
ntden(df,np,t)	the probability density function of the noncentral Student's t distribution with df degrees of freedom and noncentrality parameter np
nttail(df,np,t)	the reverse cumulative (upper tail or survivor) noncentral Student's t distribution with df degrees of freedom and noncentrality parameter np
poisson(m,k)	the probability of observing floor(k) or fewer outcomes that are distributed as Poisson with mean m

`poissonp(`m`,`k`)`	the probability of observing `floor(`k`)` outcomes that are distributed as Poisson with mean m
`poissontail(`m`,`k`)`	the probability of observing `floor(`k`)` or more outcomes that are distributed as Poisson with mean m
`t(`*df*`,`t`)`	the cumulative Student's t distribution with *df* degrees of freedom
`tden(`*df*`,`t`)`	the probability density function of Student's t distribution
`ttail(`*df*`,`t`)`	the reverse cumulative (upper tail or survivor) Student's t distribution; the probability $T > t$
`tukeyprob(`k`,`*df*`,`x`)`	the cumulative Tukey's Studentized range distribution with k ranges and *df* degrees of freedom; `0` if $x < 0$
`weibull(`a`,`b`,`x`)`	the cumulative Weibull distribution with shape a and scale b
`weibull(`a`,`b`,`g`,`x`)`	the cumulative Weibull distribution with shape a, scale b, and location g
`weibullden(`a`,`b`,`x`)`	the probability density function of the Weibull distribution with shape a and scale b
`weibullden(`a`,`b`,`g`,`x`)`	the probability density function of the Weibull distribution with shape a, scale b, and location g
`weibullph(`a`,`b`,`x`)`	the cumulative Weibull (proportional hazards) distribution with shape a and scale b
`weibullph(`a`,`b`,`g`,`x`)`	the cumulative Weibull (proportional hazards) distribution with shape a, scale b, and location g
`weibullphden(`a`,`b`,`x`)`	the probability density function of the Weibull (proportional hazards) distribution with shape a and scale b
`weibullphden(`a`,`b`,`g`,`x`)`	the probability density function of the Weibull (proportional hazards) distribution with shape a, scale b, and location g
`weibullphtail(`a`,`b`,`x`)`	the reverse cumulative Weibull (proportional hazards) distribution with shape a and scale b
`weibullphtail(`a`,`b`,`g`,`x`)`	the reverse cumulative Weibull (proportional hazards) distribution with shape a, scale b, and location g
`weibulltail(`a`,`b`,`x`)`	the reverse cumulative Weibull distribution with shape a and scale b
`weibulltail(`a`,`b`,`g`,`x`)`	the reverse cumulative Weibull distribution with shape a, scale b, and location g

String functions

`abbrev(`s`,`n`)`	name s, abbreviated to a length of n
`char(`n`)`	the character corresponding to ASCII or extended ASCII code n; `""` if n is not in the domain
`collatorlocale(`*loc*`,`*type*`)`	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`
`collatorversion(`*loc*`)`	the version string of a collator based on locale *loc*
`indexnot(`s_1`,`s_2`)`	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
`plural(`n`,`s`)`	the plural of s if $n \neq \pm 1$

plural(n,s_1,s_2)	the plural of s_1, as modified by or replaced with s_2, if $n \neq \pm 1$
real(s)	s converted to numeric or *missing*
regexm(s,re)	performs a match of a regular expression and evaluates to 1 if regular expression re is satisfied by the ASCII string s; otherwise, 0
regexr(s_1,re,s_2)	replaces the first substring within ASCII string s_1 that matches re with ASCII string s_2 and returns the resulting string
regexs(n)	subexpression n from a previous regexm() match, where $0 \leq n < 10$
soundex(s)	the soundex code for a string, s
soundex_nara(s)	the U.S. Census soundex code for a string, s
strcat(s_1,s_2)	there is no strcat() function; instead the addition operator is used to concatenate strings
strdup(s_1,n)	there is no strdup() function; instead the multiplication operator is used to create multiple copies of strings
string(n)	a synonym for strofreal(n)
string(n,s)	a synonym for strofreal(n,s)
stritrim(s)	s with multiple, consecutive internal blanks (ASCII space character char(32)) collapsed to one blank
strlen(s)	the number of characters in ASCII s or length in bytes
strlower(s)	lowercase ASCII characters in string s
strltrim(s)	s without leading blanks (ASCII space character char(32))
strmatch(s_1,s_2)	1 if s_1 matches the pattern s_2; otherwise, 0
strofreal(n)	n converted to a string
strofreal(n,s)	n converted to a string using the specified display format
strpos(s_1,s_2)	the position in s_1 at which s_2 is first found, 0 if s_2 does not occur, and 1 if s_2 is empty
strproper(s)	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
strreverse(s)	reverses the ASCII string s
strrpos(s_1,s_2)	the position in s_1 at which s_2 is last found, 0 if s_2 does not occur, and 1 if s_2 is empty
strrtrim(s)	s without trailing blanks (ASCII space character char(32))
strtoname(s[,p])	s translated into a Stata 13 compatible name
strtrim(s)	s without leading and trailing blanks (ASCII space character char(32)); equivalent to strltrim(strrtrim(s))
strupper(s)	uppercase ASCII characters in string s
subinstr(s_1,s_2,s_3,n)	s_1, where the first n occurrences in s_1 of s_2 have been replaced with s_3
subinword(s_1,s_2,s_3,n)	s_1, where the first n occurrences in s_1 of s_2 as a word have been replaced with s_3
substr(s,n_1,n_2)	the substring of s, starting at n_1, for a length of n_2
tobytes(s[,n])	escaped decimal or hex digit strings of up to 200 bytes of s
uchar(n)	the Unicode character corresponding to Unicode code point n or an empty string if n is beyond the Unicode code-point range

`udstrlen(`s`)`	the number of display columns needed to display the Unicode string s in the Stata Results window
`udsubstr(`s,n_1,n_2`)`	the Unicode substring of s, starting at character n_1, for n_2 display columns
`uisdigit(`s`)`	`1` if the first Unicode character in s is a Unicode decimal digit; otherwise, `0`
`uisletter(`s`)`	`1` if the first Unicode character in s is a Unicode letter; otherwise, `0`
`ustrcompare(`s_1,s_2[,*loc*]`)`	compares two Unicode strings
`ustrcompareex(`s_1,s_2,*loc*,*st*,*case*,*cslv*,*norm*,*num*,*alt*,*fr*`)`	compares two Unicode strings
`ustrfix(`s[,*rep*]`)`	replaces each invalid UTF-8 sequence with a Unicode character
`ustrfrom(`s,*enc*,*mode*`)`	converts the string s in encoding *enc* to a UTF-8 encoded Unicode string
`ustrinvalidcnt(`s`)`	the number of invalid UTF-8 sequences in s
`ustrleft(`s,n`)`	the first n Unicode characters of the Unicode string s
`ustrlen(`s`)`	the number of characters in the Unicode string s
`ustrlower(`s[,*loc*]`)`	lowercase all characters of Unicode string s under the given locale *loc*
`ustrltrim(`s`)`	removes the leading Unicode whitespace characters and blanks from the Unicode string s
`ustrnormalize(`s,*norm*`)`	normalizes Unicode string s to one of the five normalization forms specified by *norm*
`ustrpos(`s_1,s_2[,n]`)`	the position in s_1 at which s_2 is first found; otherwise, `0`
`ustrregexm(`s,*re*[,*noc*]`)`	performs a match of a regular expression and evaluates to `1` if regular expression *re* is satisfied by the Unicode string s; otherwise, `0`
`ustrregexra(`s_1,*re*,s_2[,*noc*]`)`	replaces all substrings within the Unicode string s_1 that match *re* with s_2 and returns the resulting string
`ustrregexrf(`s_1,*re*,s_2[,*noc*]`)`	replaces the first substring within the Unicode string s_1 that matches *re* with s_2 and returns the resulting string
`ustrregexs(`n`)`	subexpression n from a previous `ustrregexm()` match
`ustrreverse(`s`)`	reverses the Unicode string s
`ustrright(`s,n`)`	the last n Unicode characters of the Unicode string s
`ustrrpos(`s_1,s_2[,n]`)`	the position in s_1 at which s_2 is last found; otherwise, `0`
`ustrrtrim(`s`)`	remove trailing Unicode whitespace characters and blanks from the Unicode string s
`ustrsortkey(`s[,*loc*]`)`	generates a null-terminated byte array that can be used by the `sort` command to produce the same order as `ustrcompare()`
`ustrsortkeyex(`s,*loc*,*st*,*case*,*cslv*,*norm*,*num*,*alt*,*fr*`)`	generates a null-terminated byte array that can be used by the `sort` command to produce the same order as `ustrcompare()`
`ustrtitle(`s[,*loc*]`)`	a string with the first characters of Unicode words titlecased and other characters lowercased
`ustrto(`s,*enc*,*mode*`)`	converts the Unicode string s in UTF-8 encoding to a string in encoding *enc*
`ustrtohex(`s[,n]`)`	escaped hex digit string of s up to 200 Unicode characters

`ustrtoname(`s[`,`p]`)`	string s translated into a Stata name
`ustrtrim(`s`)`	removes leading and trailing Unicode whitespace characters and blanks from the Unicode string s
`ustrunescape(`s`)`	the Unicode string corresponding to the escaped sequences of s
`ustrupper(`s[`,`*loc*]`)`	uppercase all characters in string s under the given locale *loc*
`ustrword(`s`,`n[`,`*loc*]`)`	the nth Unicode word in the Unicode string s
`ustrwordcount(`s[`,`*loc*]`)`	the number of nonempty Unicode words in the Unicode string s
`usubinstr(`s_1`,`s_2`,`s_3`,`n`)`	replaces the first n occurrences of the Unicode string s_2 with the Unicode string s_3 in s_1
`usubstr(`s`,`n_1`,`n_2`)`	the Unicode substring of s, starting at n_1, for a length of n_2
`word(`s`,`n`)`	the nth word in s; *missing* (`""`) if n is missing
`wordbreaklocale(`*loc*`,`*type*`)`	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*
`wordcount(`s`)`	the number of words in s

Trigonometric functions

`acos(`x`)`	the radian value of the arccosine of x
`acosh(`x`)`	the inverse hyperbolic cosine of x
`asin(`x`)`	the radian value of the arcsine of x
`asinh(`x`)`	the inverse hyperbolic sine of x
`atan(`x`)`	the radian value of the arctangent of x
`atan2(`y`, `x`)`	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
`atanh(`x`)`	the inverse hyperbolic tangent of x
`cos(`x`)`	the cosine of x, where x is in radians
`cosh(`x`)`	the hyperbolic cosine of x
`sin(`x`)`	the sine of x, where x is in radians
`sinh(`x`)`	the hyperbolic sine of x
`tan(`x`)`	the tangent of x, where x is in radians
`tanh(`x`)`	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**

Title

Functions by name

abbrev(s,n)	name s, abbreviated to a length of n
abs(x)	the absolute value of x
acos(x)	the radian value of the arccosine of x
acosh(x)	the inverse hyperbolic cosine of x
age($e_{d\,\mathrm{DOB}}$,e_d[,s_{nl}])	the age in integer years on e_d for date of birth $e_{d\,\mathrm{DOB}}$ with s_{nl} the nonleap-year birthday for 29feb birthdates
age_frac($e_{d\,\mathrm{DOB}}$,e_d[,s_{nl}])	the age in years, including the fractional part, on e_d for date of birth $e_{d\,\mathrm{DOB}}$ with s_{nl} the nonleap-year birthday for 29feb birthdates
asin(x)	the radian value of the arcsine of x
asinh(x)	the inverse hyperbolic sine of x
atan(x)	the radian value of the arctangent of x
atan2(y, x)	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
atanh(x)	the inverse hyperbolic tangent of x
autocode(x,n,x_0,x_1)	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 or the upper bound of the first or last interval if $x < x_0$ or $x > x_1$, respectively
betaden(a,b,x)	the probability density of the beta distribution, where a and b are the shape parameters; 0 if $x < 0$ or $x > 1$
binomial(n,k,θ)	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 $k > n$
binomialp(n,k,p)	the probability of observing floor(k) successes in floor(n) trials when the probability of a success on one trial is p
binomialtail(n,k,θ)	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 $k > n$
binormal(h,k,ρ)	the joint cumulative distribution $\Phi(h, k, \rho)$ of bivariate normal with correlation ρ
birthday($e_{d\,\mathrm{DOB}}$,Y[,s_{nl}])	the e_d date of the birthday in year Y for date of birth $e_{d\,\mathrm{DOB}}$ with s_{nl} the nonleap-year birthday for 29feb birthdates
bofd("*cal*",e_d)	the e_b business date corresponding to e_d
byteorder()	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
c(*name*)	the value of the system or constant result c(*name*) (see [P] **creturn**)
_caller()	version of the program or session that invoked the currently running program; see [P] **version**
cauchy(a,b,x)	the cumulative Cauchy distribution with location parameter a and scale parameter b

`cauchyden(`a`,`b`,`x`)` the probability density of the Cauchy distribution with location parameter a and scale parameter b

`cauchytail(`a`,`b`,`x`)` the reverse cumulative (upper tail or survivor) Cauchy distribution with location parameter a and scale parameter b

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

`ceil(`x`)` the unique integer n such that $n - 1 < x \leq n$; x (not ".") if x is missing, meaning that `ceil(.a)` = `.a`

`char(`n`)` the character corresponding to ASCII or extended ASCII code n; `""` if n is not in the domain

`chi2(`df`,`x`)` the cumulative χ^2 distribution with df degrees of freedom; `0` if $x < 0$

`chi2den(`df`,`x`)` the probability density of the χ^2 distribution with df degrees of freedom; `0` if $x < 0$

`chi2tail(`df`,`x`)` the reverse cumulative (upper tail or survivor) χ^2 distribution with df degrees of freedom; `1` if $x < 0$

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

`chop(`x`, `ϵ`)` `round(`x`)` if `abs(`x $-$ `round(`x`))` $< \epsilon$; otherwise, x; or x if x is missing

`cholesky(`M`)` the Cholesky decomposition of the matrix: if $R =$ `cholesky(`S`)`, then $RR^T = S$

`clip(`x`,`a`,`b`)` 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

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

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

`Clockdiff(`e_{tC1}`, `e_{tC2}`,`s_u`)` the e_{tC} datetime difference, rounded down to an integer, from e_{tC1} to e_{tC2} in s_u units of days, hours, minutes, seconds, or milliseconds

`clockdiff(`e_{tc1}`,`e_{tc2}`,`s_u`)` the e_{tc} datetime difference, rounded down to an integer, from e_{tc1} to e_{tc2} in s_u units of days, hours, minutes, seconds, or milliseconds

`Clockdiff_frac(`e_{tC1}`,`e_{tC2}`,`s_u`)` the e_{tC} datetime difference, including the fractional part, from e_{tC1} to e_{tC2} in s_u units of days, hours, minutes, seconds, or milliseconds

`clockdiff_frac(`e_{tc1}`,`e_{tc2}`,`s_u`)` the e_{tc} datetime difference, including the fractional part, from e_{tc1} to e_{tc2} in s_u units of days, hours, minutes, seconds, or milliseconds

`Clockpart(`e_{tC}`,`s_u`)` the integer year, month, day, hour, minute, second, or millisecond of e_{tC} with s_u specifying which time part

`clockpart(`e_{tc}`,`s_u`)` the integer year, month, day, hour, minute, second, or millisecond of e_{tc} with s_u specifying which time part

`cloglog(`x`)` the complementary log-log of x

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

`Cofc(`e_{tc}`)` 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)

`cofC(`e_{tC}`)`	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)
`Cofd(`e_d`)`	the e_{tC} datetime (ms. with leap seconds since 01jan1960 00:00:00.000) of date e_d at time 00:00:00.000
`cofd(`e_d`)`	the e_{tc} datetime (ms. since 01jan1960 00:00:00.000) of date e_d at time 00:00:00.000
`coleqnumb(`M`,`s`)`	the equation number of M associated with column equation s; *missing* if the column equation cannot be found
`collatorlocale(`*loc*`,`*type*`)`	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`
`collatorversion(`*loc*`)`	the version string of a collator based on locale *loc*
`colnfreeparms(`M`)`	the number of free parameters in columns of M
`colnumb(`M`,`s`)`	the column number of M associated with column name s; *missing* if the column cannot be found
`colsof(`M`)`	the number of columns of M
`comb(`n`,`k`)`	the combinatorial function $n!/\{k!(n-k)!\}$
`cond(`x`,`a`,`b[`,`c]`)`	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*
`corr(`M`)`	the correlation matrix of the variance matrix
`cos(`x`)`	the cosine of x, where x is in radians
`cosh(`x`)`	the hyperbolic cosine of x
`daily(`s_1`,`s_2[`,`Y]`)`	a synonym for `date(`s_1`,`s_2[`,`Y]`)`
`date(`s_1`,`s_2[`,`Y]`)`	the e_d date (days since 01jan1960) corresponding to s_1 based on s_2 and Y
`datediff(`e_{d1}`,`e_{d2}`,`s_u[`,`s_{nl}]`)`	the difference, rounded down to an integer, from e_{d1} to e_{d2} in s_u units of days, months, or years with s_{nl} the nonleap-year anniversary for e_{d1} on 29feb
`datediff_frac(`e_{d1}`,`e_{d2}`,`s_u[`,`s_{nl}]`)`	the difference, including the fractional part, from e_{d1} to e_{d2} in s_u units of days, months, or years with s_{nl} the nonleap-year anniversary for e_{d1} on 29feb
`datepart(`e_d`,`s_u`)`	the integer year, month, or day of e_d with s_u specifying year, month, or day
`day(`e_d`)`	the numeric day of the month corresponding to e_d
`daysinmonth(`e_d`)`	the number of days in the month of e_d
`det(`M`)`	the determinant of matrix M
`dgammapda(`a`,`x`)`	$\frac{\partial P(a,x)}{\partial a}$, where $P(a,x) =$ `gammap(`a`,`x`)`; `0` if $x<0$
`dgammapdada(`a`,`x`)`	$\frac{\partial^2 P(a,x)}{\partial a^2}$, where $P(a,x) =$ `gammap(`a`,`x`)`; `0` if $x<0$
`dgammapdadx(`a`,`x`)`	$\frac{\partial^2 P(a,x)}{\partial a \partial x}$, where $P(a,x) =$ `gammap(`a`,`x`)`; `0` if $x<0$
`dgammapdx(`a`,`x`)`	$\frac{\partial P(a,x)}{\partial x}$, where $P(a,x) =$ `gammap(`a`,`x`)`; `0` if $x<0$
`dgammapdxdx(`a`,`x`)`	$\frac{\partial^2 P(a,x)}{\partial x^2}$, where $P(a,x) =$ `gammap(`a`,`x`)`; `0` if $x<0$

dhms(e_d,h,m,s)	the e_{tc} datetime (ms. since 01jan1960 00:00:00.000) corresponding to e_d, h, m, and s
diag(M)	the square, diagonal matrix created from the row or column vector
diag0cnt(M)	the number of zeros on the diagonal of M
digamma(x)	the digamma() function, $d\ln\Gamma(x)/dx$
dofb(e_b,"*cal*")	the e_d datetime corresponding to e_b
dofC(e_{tC})	the e_d date (days since 01jan1960) of datetime e_{tC} (ms. with leap seconds since 01jan1960 00:00:00.000)
dofc(e_{tc})	the e_d date (days since 01jan1960) of datetime e_{tc} (ms. since 01jan1960 00:00:00.000)
dofh(e_h)	the e_d date (days since 01jan1960) of the start of half-year e_h
dofm(e_m)	the e_d date (days since 01jan1960) of the start of month e_m
dofq(e_q)	the e_d date (days since 01jan1960) of the start of quarter e_q
dofw(e_w)	the e_d date (days since 01jan1960) of the start of week e_w
dofy(e_y)	the e_d date (days since 01jan1960) of 01jan in year e_y
dow(e_d)	the numeric day of the week corresponding to date e_d; 0 = Sunday, 1 = Monday, . . . , 6 = Saturday
doy(e_d)	the numeric day of the year corresponding to date e_d
dunnettprob(k,*df*,x)	the cumulative multiple range distribution that is used in Dunnett's multiple-comparison method with k ranges and *df* degrees of freedom; 0 if $x < 0$
e(*name*)	the value of stored result e(*name*); see **[U] 18.8 Accessing results calculated by other programs**
el(s,i,j)	s[floor(i),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
e(sample)	1 if the observation is in the estimation sample and 0 otherwise
epsdouble()	the machine precision of a double-precision number
epsfloat()	the machine precision of a floating-point number
exp(x)	the exponential function e^x
expm1(x)	$e^x - 1$ with higher precision than exp(x) $- 1$ for small values of $\lvert x\rvert$
exponential(b,x)	the cumulative exponential distribution with scale b
exponentialden(b,x)	the probability density function of the exponential distribution with scale b
exponentialtail(b,x)	the reverse cumulative exponential distribution with scale b
F(df_1,df_2,f)	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 $f < 0$
Fden(df_1,df_2,f)	the probability density function of the F distribution with df_1 numerator and df_2 denominator degrees of freedom; 0 if $f < 0$
fileexists(f)	1 if the file specified by f exists; otherwise, 0
fileread(f)	the contents of the file specified by f
filereaderror(s)	0 or positive integer, said value having the interpretation of a return code

`filewrite(`f`,`s[`,`r]`)`	writes the string specified by s to the file specified by f and returns the number of bytes in the resulting file
`firstdayofmonth(`e_d`)`	the e_d date of the first day of the month of e_d
`float(`x`)`	the value of x rounded to `float` precision
`floor(`x`)`	the unique integer n such that $n \leq x < n+1$; x (not ".") if x is missing, meaning that `floor(.a)` = `.a`
`fmtwidth(`*fmtstr*`)`	the output length of the *%fmt* contained in *fmtstr*; *missing* if *fmtstr* does not contain a valid *%fmt*
`frval()`	returns values of variables stored in other frames
`_frval()`	programmer's version of `frval()`
`Ftail(`df_1`,`df_2`,`f`)`	the reverse cumulative (upper tail or survivor) F distribution with df_1 numerator and df_2 denominator degrees of freedom; `1` if $f < 0$
`gammaden(`a`,`b`,`g`,`x`)`	the probability density function of the gamma distribution; `0` if $x < g$
`gammap(`a`,`x`)`	the cumulative gamma distribution with shape parameter a; `0` if $x < 0$
`gammaptail(`a`,`x`)`	the reverse cumulative (upper tail or survivor) gamma distribution with shape parameter a; `1` if $x < 0$
`get(`*systemname*`)`	a copy of Stata internal system matrix *systemname*
`hadamard(`M`,`N`)`	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)
`halfyear(`e_d`)`	the numeric half of the year corresponding to date e_d
`halfyearly(`s_1`,`s_2[`,`Y]`)`	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()`
`has_eprop(`*name*`)`	`1` if *name* appears as a word in `e(properties)`; otherwise, `0`
`hh(`e_{tc}`)`	the hour corresponding to datetime e_{tc} (ms. since 01jan1960 00:00:00.000)
`hhC(`e_{tC}`)`	the hour corresponding to datetime e_{tC} (ms. with leap seconds since 01jan1960 00:00:00.000)
`hms(`h`,`m`,`s`)`	the e_{tc} datetime (ms. since 01jan1960 00:00:00.000) corresponding to h, m, s on 01jan1960
`hofd(`e_d`)`	the e_h half-yearly date (half years since 1960h1) containing date e_d
`hours(`ms`)`	$ms/3{,}600{,}000$
`hypergeometric(`N`,`K`,`n`,`k`)`	the cumulative probability of the hypergeometric distribution
`hypergeometricp(`N`,`K`,`n`,`k`)`	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
`I(`n`)`	an $n \times n$ identity matrix if n is an integer; otherwise, a `round(`n`)` $\times$ `round(`n`)` identity matrix
`ibeta(`a`,`b`,`x`)`	the cumulative beta distribution with shape parameters a and b; `0` if $x < 0$; or `1` if $x > 1$
`ibetatail(`a`,`b`,`x`)`	the reverse cumulative (upper tail or survivor) beta distribution with shape parameters a and b; `1` if $x < 0$; or `0` if $x > 1$
`igaussian(`m`,`a`,`x`)`	the cumulative inverse Gaussian distribution with mean m and shape parameter a; `0` if $x \leq 0$

igaussianden(m,a,x)	the probability density of the inverse Gaussian distribution with mean m and shape parameter a; 0 if $x \leq 0$
igaussiantail(m,a,x)	the reverse cumulative (upper tail or survivor) inverse Gaussian distribution with mean m and shape parameter a; 1 if $x \leq 0$
indexnot(s_1,s_2)	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
inlist(z,a,b,...)	1 if z is a member of the remaining arguments; otherwise, 0
inrange(z,a,b)	1 if it is known that $a \leq z \leq b$; otherwise, 0
int(x)	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
inv(M)	the inverse of the matrix M
invbinomial(n,k,p)	the inverse of the cumulative binomial; that is, θ ($\theta =$ probability of success on one trial) such that the probability of observing floor(k) or fewer successes in floor(n) trials is p
invbinomialtail(n,k,p)	the inverse of the right cumulative binomial; that is, θ ($\theta =$ probability of success on one trial) such that the probability of observing floor(k) or more successes in floor(n) trials is p
invcauchy(a,b,p)	the inverse of cauchy(): if cauchy(a,b,x) $= p$, then invcauchy(a,b,p) $= x$
invcauchytail(a,b,p)	the inverse of cauchytail(): if cauchytail(a,b,x) $= p$, then invcauchytail(a,b,p) $= x$
invchi2(df,p)	the inverse of chi2(): if chi2(df,x) $= p$, then invchi2(df,p) $= x$
invchi2tail(df,p)	the inverse of chi2tail(): if chi2tail(df,x) $= p$, then invchi2tail(df,p) $= x$
invcloglog(x)	the inverse of the complementary log-log function of x
invdunnettprob(k,df,p)	the inverse cumulative multiple range distribution that is used in Dunnett's multiple-comparison method with k ranges and df degrees of freedom
invexponential(b,p)	the inverse cumulative exponential distribution with scale b: if exponential(b,x) $= p$, then invexponential(b,p) $= x$
invexponentialtail(b,p)	the inverse reverse cumulative exponential distribution with scale b: if exponentialtail(b,x) $= p$, then invexponentialtail(b,p) $= x$
invF(df_1,df_2,p)	the inverse cumulative F distribution: if F(df_1,df_2,f) $= p$, then invF(df_1,df_2,p) $= f$
invFtail(df_1,df_2,p)	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) $= f$
invgammap(a,p)	the inverse cumulative gamma distribution: if gammap(a,x) $= p$, then invgammap(a,p) $= x$
invgammaptail(a,p)	the inverse reverse cumulative (upper tail or survivor) gamma distribution: if gammaptail(a,x) $= p$, then invgammaptail(a,p) $= x$
invibeta(a,b,p)	the inverse cumulative beta distribution: if ibeta(a,b,x) $= p$, then invibeta(a,b,p) $= x$
invibetatail(a,b,p)	the inverse reverse cumulative (upper tail or survivor) beta distribution: if ibetatail(a,b,x) $= p$, then invibetatail(a,b,p) $= x$

`invigaussian(`m`,`a`,`p`)`	the inverse of `igaussian()`: if `igaussian(`m`,`a`,`x`)` $= p$, then `invigaussian(`m`,`a`,`p`)` $= x$
`invigaussiantail(`m`,`a`,`p`)`	the inverse of `igaussiantail()`: if `igaussiantail(`m`,`a`,`x`)` $= p$, then `invigaussiantail(`m`,`a`,`p`)` $= x$
`invlaplace(`m`,`b`,`p`)`	the inverse of `laplace()`: if `laplace(`m`,`b`,`x`)` $= p$, then `invlaplace(`m`,`b`,`p`)` $= x$
`invlaplacetail(`m`,`b`,`p`)`	the inverse of `laplacetail()`: if `laplacetail(`m`,`b`,`x`)` $= p$, then `invlaplacetail(`m`,`b`,`p`)` $= x$
`invlogistic(`p`)`	the inverse cumulative logistic distribution: if `logistic(`x`)` $= p$, then `invlogistic(`p`)` $= x$
`invlogistic(`s`,`p`)`	the inverse cumulative logistic distribution: if `logistic(`s`,`x`)` $= p$, then `invlogistic(`s`,`p`)` $= x$
`invlogistic(`m`,`s`,`p`)`	the inverse cumulative logistic distribution: if `logistic(`m`,`s`,`x`)` $= p$, then `invlogistic(`m`,`s`,`p`)` $= x$
`invlogistictail(`p`)`	the inverse reverse cumulative logistic distribution: if `logistictail(`x`)` $= p$, then `invlogistictail(`p`)` $= x$
`invlogistictail(`s`,`p`)`	the inverse cumulative logistic distribution: if `logistic(`s`,`x`)` $= p$, then `invlogistic(`s`,`p`)` $= x$
`invlogistictail(`m`,`s`,`p`)`	the inverse cumulative logistic distribution: if `logistic(`m`,`s`,`x`)` $= p$, then `invlogistic(`m`,`s`,`p`)` $= x$
`invlogit(`x`)`	the inverse of the logit function of x
`invnbinomial(`n`,`k`,`q`)`	the value of the negative binomial parameter, p, such that $q =$ `nbinomial(`n`,`k`,`p`)`
`invnbinomialtail(`n`,`k`,`q`)`	the value of the negative binomial parameter, p, such that $q =$ `nbinomialtail(`n`,`k`,`p`)`
`invnchi2(`df`,`np`,`p`)`	the inverse cumulative noncentral χ^2 distribution: if `nchi2(`df`,`np`,`x`)` $= p$, then `invnchi2(`df`,`np`,`p`)` $= x$
`invnchi2tail(`df`,`np`,`p`)`	the inverse reverse cumulative (upper tail or survivor) noncentral χ^2 distribution: if `nchi2tail(`df`,`np`,`x`)` $= p$, then `invnchi2tail(`df`,`np`,`p`)` $= x$
`invnF(`df_1`,`df_2`,`np`,`p`)`	the inverse cumulative noncentral F distribution: if `nF(`df_1`,`df_2`,`np`,`f`)` $= p$, then `invnF(`df_1`,`df_2`,`np`,`p`)` $= f$
`invnFtail(`df_1`,`df_2`,`np`,`p`)`	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`)` $= f$
`invnibeta(`a`,`b`,`np`,`p`)`	the inverse cumulative noncentral beta distribution: if `nibeta(`a`,`b`,`np`,`x`)` $= p$, then `invibeta(`a`,`b`,`np`,`p`)` $= x$
`invnormal(`p`)`	the inverse cumulative standard normal distribution: if `normal(`z`)` $= p$, then `invnormal(`p`)` $= z$
`invnt(`df`,`np`,`p`)`	the inverse cumulative noncentral Student's t distribution: if `nt(`df`,`np`,`t`)` $= p$, then `invnt(`df`,`np`,`p`)` $= t$
`invnttail(`df`,`np`,`p`)`	the inverse reverse cumulative (upper tail or survivor) noncentral Student's t distribution: if `nttail(`df`,`np`,`t`)` $= p$, then `invnttail(`df`,`np`,`p`)` $= t$
`invpoisson(`k`,`p`)`	the Poisson mean such that the cumulative Poisson distribution evaluated at k is p: if `poisson(`m`,`k`)` $= p$, then `invpoisson(`k`,`p`)` $= m$

`invpoissontail(`k`,`q`)` the Poisson mean such that the reverse cumulative Poisson distribution evaluated at k is q: if `poissontail(`m`,`k`)` $= q$, then `invpoissontail(`k`,`q`)` $= m$

`invsym(`M`)` the inverse of M if M is positive definite

`invt(`df`,`p`)` the inverse cumulative Student's t distribution: if `t(`df`,`t`)` $= p$, then `invt(`df`,`p`)` $= t$

`invttail(`df`,`p`)` the inverse reverse cumulative (upper tail or survivor) Student's t distribution: if `ttail(`df`,`t`)` $= p$, then `invttail(`df`,`p`)` $= t$

`invtukeyprob(`k`,`df`,`p`)` the inverse cumulative Tukey's Studentized range distribution with k ranges and df degrees of freedom

`invweibull(`a`,`b`,`p`)` the inverse cumulative Weibull distribution with shape a and scale b: if `weibull(`a`,`b`,`x`)` $= p$, then `invweibull(`a`,`b`,`p`)` $= x$

`invweibull(`a`,`b`,`g`,`p`)` the inverse cumulative Weibull distribution with shape a, scale b, and location g: if `weibull(`a`,`b`,`g`,`x`)` $= p$, then `invweibull(`a`,`b`,`g`,`p`)` $= x$

`invweibullph(`a`,`b`,`p`)` the inverse cumulative Weibull (proportional hazards) distribution with shape a and scale b: if `weibullph(`a`,`b`,`x`)` $= p$, then `invweibullph(`a`,`b`,`p`)` $= x$

`invweibullph(`a`,`b`,`g`,`p`)` 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`)` $= x$

`invweibullphtail(`a`,`b`,`p`)` 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`)`$=x$

`invweibullphtail(`a`,`b`,`g`,`p`)` 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`)` $= x$

`invweibulltail(`a`,`b`,`p`)` the inverse reverse cumulative Weibull distribution with shape a and scale b: if `weibulltail(`a`,`b`,`x`)` $= p$, then `invweibulltail(`a`,`b`,`p`)` $= x$

`invweibulltail(`a`,`b`,`g`,`p`)` the inverse reverse cumulative Weibull distribution with shape a, scale b, and location g: if `weibulltail(`a`,`b`,`g`,`x`)` $= p$, then `invweibulltail(`a`,`b`,`g`,`p`)` $= x$

`irecode(`x`,`x_1`,`$\ldots$`,`x_n`)` *missing* if x is missing or $x_1, \ldots, x_n$ is not weakly increasing; `0` if $x \le x_1$; `1` if $x_1 < x \le x_2$; `2` if $x_2 < x \le x_3$; $\ldots$; n if $x > x_n$

`isleapsecond(`e_{tC}`)` `1` if e_{tC} is a leap second; otherwise, `0`

`isleapyear(`Y`)` `1` if Y is a leap year; otherwise, `0`

`issymmetric(`M`)` `1` if the matrix is symmetric; otherwise, `0`

`J(`r`,`c`,`z`)` the $r \times c$ matrix containing elements z

`laplace(`m`,`b`,`x`)` the cumulative Laplace distribution with mean m and scale parameter b

`laplaceden(`m`,`b`,`x`)` the probability density of the Laplace distribution with mean m and scale parameter b

`laplacetail(`m`,`b`,`x`)` the reverse cumulative (upper tail or survivor) Laplace distribution with mean m and scale parameter b

`lastdayofmonth(`e_d`)`	the e_d date of the last day of the month of e_d
`ln(`x`)`	the natural logarithm, $\ln(x)$
`ln1m(`x`)`	the natural logarithm of $1-x$ with higher precision than `ln(`$1-x$`)` for small values of $\lvert x\rvert$
`ln1p(`x`)`	the natural logarithm of $1+x$ with higher precision than `ln(`$1+x$`)` for small values of $\lvert x\rvert$
`lncauchyden(`a`,`b`,`x`)`	the natural logarithm of the density of the Cauchy distribution with location parameter a and scale parameter b
`lnfactorial(`n`)`	the natural log of n factorial = $\ln(n!)$
`lngamma(`x`)`	$\ln\{\Gamma(x)\}$
`lnigammaden(`a`,`b`,`x`)`	the natural logarithm of the inverse gamma density, where a is the shape parameter and b is the scale parameter
`lnigaussianden(`m`,`a`,`x`)`	the natural logarithm of the inverse Gaussian density with mean m and shape parameter a
`lniwishartden(`df`,`V`,`X`)`	the natural logarithm of the density of the inverse Wishart distribution; missing if $df \le n-1$
`lnlaplaceden(`m`,`b`,`x`)`	the natural logarithm of the density of the Laplace distribution with mean m and scale parameter b
`lnmvnormalden(`M`,`V`,`X`)`	the natural logarithm of the multivariate normal density
`lnnormal(`z`)`	the natural logarithm of the cumulative standard normal distribution
`lnnormalden(`z`)`	the natural logarithm of the standard normal density, $N(0,1)$
`lnnormalden(`x`,`σ`)`	the natural logarithm of the normal density with mean 0 and standard deviation σ
`lnnormalden(`x`,`μ`,`σ`)`	the natural logarithm of the normal density with mean μ and standard deviation σ, $N(\mu,\sigma^2)$
`lnwishartden(`df`,`V`,`X`)`	the natural logarithm of the density of the Wishart distribution; missing if $df \le n-1$
`log(`x`)`	a synonym for `ln(`x`)`
`log10(`x`)`	the base-10 logarithm of x
`log1m(`x`)`	a synonym for `ln1m(`x`)`
`log1p(`x`)`	a synonym for `ln1p(`x`)`
`logistic(`x`)`	the cumulative logistic distribution with mean 0 and standard deviation $\pi/\sqrt{3}$
`logistic(`s`,`x`)`	the cumulative logistic distribution with mean 0, scale s, and standard deviation $s\pi/\sqrt{3}$
`logistic(`m`,`s`,`x`)`	the cumulative logistic distribution with mean m, scale s, and standard deviation $s\pi/\sqrt{3}$
`logisticden(`x`)`	the density of the logistic distribution with mean 0 and standard deviation $\pi/\sqrt{3}$
`logisticden(`s`,`x`)`	the density of the logistic distribution with mean 0, scale s, and standard deviation $s\pi/\sqrt{3}$
`logisticden(`m`,`s`,`x`)`	the density of the logistic distribution with mean m, scale s, and standard deviation $s\pi/\sqrt{3}$
`logistictail(`x`)`	the reverse cumulative logistic distribution with mean 0 and standard deviation $\pi/\sqrt{3}$

`logistictail(`s`,`x`)`	the reverse cumulative logistic distribution with mean 0, scale s, and standard deviation $s\pi/\sqrt{3}$
`logistictail(`m`,`s`,`x`)`	the reverse cumulative logistic distribution with mean m, scale s, and standard deviation $s\pi/\sqrt{3}$
`logit(`x`)`	the log of the odds ratio of x, `logit(`x`)` $= \ln\{x/(1-x)\}$
`matmissing(`M`)`	`1` if any elements of the matrix are missing; otherwise, `0`
`matrix(`*exp*`)`	restricts name interpretation to scalars and matrices; see `scalar()`
`matuniform(`r`,`c`)`	the $r \times c$ matrices containing uniformly distributed pseudorandom numbers on the interval $(0,1)$
`max(`x_1`,`x_2`,...,`x_n`)`	the maximum value of $x_1, x_2, \ldots, x_n$
`maxbyte()`	the largest value that can be stored in storage type `byte`
`maxdouble()`	the largest value that can be stored in storage type `double`
`maxfloat()`	the largest value that can be stored in storage type `float`
`maxint()`	the largest value that can be stored in storage type `int`
`maxlong()`	the largest value that can be stored in storage type `long`
`mdy(`M`,`D`,`Y`)`	the e_d date (days since 01jan1960) corresponding to M, D, Y
`mdyhms(`M`,`D`,`Y`,`h`,`m`,`s`)`	the e_{tc} datetime (ms. since 01jan1960 00:00:00.000) corresponding to M, D, Y, h, m, s
`mi(`x_1`,`x_2`,...,`x_n`)`	a synonym for `missing(`x_1`,`x_2`,...,`x_n`)`
`min(`x_1`,`x_2`,...,`x_n`)`	the minimum value of $x_1, x_2, \ldots, x_n$
`minbyte()`	the smallest value that can be stored in storage type `byte`
`mindouble()`	the smallest value that can be stored in storage type `double`
`minfloat()`	the smallest value that can be stored in storage type `float`
`minint()`	the smallest value that can be stored in storage type `int`
`minlong()`	the smallest value that can be stored in storage type `long`
`minutes(`ms`)`	$ms/60{,}000$
`missing(`x_1`,`x_2`,...,`x_n`)`	`1` if any x_i evaluates to *missing*; otherwise, `0`
`mm(`e_{tc}`)`	the minute corresponding to datetime e_{tc} (ms. since 01jan1960 00:00:00.000)
`mmC(`e_{tC}`)`	the minute corresponding to datetime e_{tC} (ms. with leap seconds since 01jan1960 00:00:00.000)
`mod(`x`,`y`)`	the modulus of x with respect to y
`mofd(`e_d`)`	the e_m monthly date (months since 1960m1) containing date e_d
`month(`e_d`)`	the numeric month corresponding to date e_d
`monthly(`s_1`,`s_2`[,`Y`])`	the e_m monthly date (months since 1960m1) corresponding to s_1 based on s_2 and Y; Y specifies *topyear*; see `date()`
`mreldif(`X`,`Y`)`	the relative difference of X and Y, where the relative difference is defined as $\max_{i,j}\{\lvert x_{ij}-y_{ij}\rvert/(\lvert y_{ij}\rvert+1)\}$
`msofhours(`h`)`	$h \times 3{,}600{,}000$
`msofminutes(`m`)`	$m \times 60{,}000$
`msofseconds(`s`)`	$s \times 1{,}000$
`nbetaden(`a`,`b`,`np`,`x`)`	the probability density function of the noncentral beta distribution; `0` if $x < 0$ or $x > 1$

nbinomial(n,k,p)	the cumulative probability of the negative binomial distribution
nbinomialp(n,k,p)	the negative binomial probability
nbinomialtail(n,k,p)	the reverse cumulative probability of the negative binomial distribution
nchi2(df,np,x)	the cumulative noncentral χ^2 distribution; 0 if $x < 0$
nchi2den(df,np,x)	the probability density of the noncentral χ^2 distribution; 0 if $x < 0$
nchi2tail(df,np,x)	the reverse cumulative (upper tail or survivor) noncentral χ^2 distribution; 1 if $x < 0$
nextbirthday($e_{d\,\mathrm{DOB}}$,e_d[,s_{nl}])	the e_d date of the first birthday after e_d for date of birth $e_{d\,\mathrm{DOB}}$ with s_{nl} the nonleap-year birthday for 29feb birthdates
nextleapyear(Y)	the first leap year after year Y
nF(df_1,df_2,np,f)	the cumulative 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,np,f)	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 $f < 0$
nFtail(df_1,df_2,np,f)	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 $f < 0$
nibeta(a,b,np,x)	the cumulative noncentral beta distribution; 0 if $x < 0$; or 1 if $x > 1$
normal(z)	the cumulative standard normal distribution
normalden(z)	the standard normal density, $N(0, 1)$
normalden(x,σ)	the normal density with mean 0 and standard deviation σ
normalden(x,μ,σ)	the normal density with mean μ and standard deviation σ, $N(\mu, \sigma^2)$
now()	the current e_{tc} datetime
npnchi2(df,x,p)	the noncentrality parameter, np, for noncentral χ^2: if nchi2(df,np,x) $= p$, then npnchi2(df,x,p) $= np$
npnF(df_1,df_2,f,p)	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) $= np$
npnt(df,t,p)	the noncentrality parameter, np, for the noncentral Student's t distribution: if nt(df,np,t) $= p$, then npnt(df,t,p) $= np$
nt(df,np,t)	the cumulative noncentral Student's t distribution with df degrees of freedom and noncentrality parameter np
ntden(df,np,t)	the probability density function of the noncentral Student's t distribution with df degrees of freedom and noncentrality parameter np
nttail(df,np,t)	the reverse cumulative (upper tail or survivor) noncentral Student's t distribution with df degrees of freedom and noncentrality parameter np
nullmat(*matname*)	use with the row-join (,) and column-join (\\) operators
plural(n,s)	the plural of s if $n \neq \pm 1$
plural(n,s_1,s_2)	the plural of s_1, as modified by or replaced with s_2, if $n \neq \pm 1$
poisson(m,k)	the probability of observing floor(k) or fewer outcomes that are distributed as Poisson with mean m

`poissonp(`m`,`k`)`	the probability of observing `floor(`k`)` outcomes that are distributed as Poisson with mean m
`poissontail(`m`,`k`)`	the probability of observing `floor(`k`)` or more outcomes that are distributed as Poisson with mean m
`previousbirthday(`$e_{d\,\mathrm{DOB}}$`,`e_d [`,`s_{nl}]`)`	the e_d date of the birthday immediately before e_d for date of birth $e_{d\,\mathrm{DOB}}$ with s_{nl} the nonleap-year birthday for 29feb birthdates
`previousleapyear(`Y`)`	the leap year immediately before year Y
`qofd(`e_d`)`	the e_q quarterly date (quarters since 1960q1) containing date e_d
`quarter(`e_d`)`	the numeric quarter of the year corresponding to date e_d
`quarterly(`s_1`,`s_2 [`,`Y]`)`	the e_q quarterly date (quarters since 1960q1) corresponding to s_1 based on s_2 and Y; Y specifies *topyear*; see `date()`
`r(`*name*`)`	the value of the stored result `r(`*name*`)`; see [U] **18.8 Accessing results calculated by other programs**
`rbeta(`a`,`b`)`	beta(a,b) random variates, where a and b are the beta distribution shape parameters
`rbinomial(`n`,`p`)`	binomial(n,p) random variates, where n is the number of trials and p is the success probability
`rcauchy(`a`,`b`)`	Cauchy(a,b) random variates, where a is the location parameter and b is the scale parameter
`rchi2(`*df*`)`	χ^2, with *df* degrees of freedom, random variates
`recode(`x`,`x_1`,`...`,`x_n`)`	*missing* if $x_1, x_2, \ldots, x_n$ is not weakly increasing; x if x is missing; x_1 if $x \le x_1$; x_2 if $x \le x_2$, ...; otherwise, x_n if $x > x_1$, x_2, ..., x_{n-1}. $x_i \geq .$ is interpreted as $x_i = +\infty$
`real(`s`)`	s converted to numeric or *missing*
`regexm(`s`,`re`)`	performs a match of a regular expression and evaluates to `1` if regular expression re is satisfied by the ASCII string s; otherwise, `0`
`regexr(`s_1`,`re`,`s_2`)`	replaces the first substring within ASCII string s_1 that matches re with ASCII string s_2 and returns the resulting string
`regexs(`n`)`	subexpression n from a previous `regexm()` match, where $0 \le n < 10$
`reldif(`x`,`y`)`	the "relative" difference $\lvert x-y\rvert/(\lvert y\rvert+1)$; `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*
`replay()`	`1` if the first nonblank character of local macro `` `0' `` is a comma, or if `` `0' `` is empty
`return(`*name*`)`	the value of the to-be-stored result `r(`*name*`)`; see [P] **return**
`rexponential(`b`)`	exponential random variates with scale b
`rgamma(`a`,`b`)`	gamma(a,b) random variates, where a is the gamma shape parameter and b is the scale parameter
`rhypergeometric(`N`,`K`,`n`)`	hypergeometric random variates
`rigaussian(`m`,`a`)`	inverse Gaussian random variates with mean m and shape parameter a
`rlaplace(`m`,`b`)`	Laplace(m,b) random variates with mean m and scale parameter b
`rlogistic()`	logistic variates with mean 0 and standard deviation $\pi/\sqrt{3}$

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

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

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

`ustrregexra(`s_1`,`re`,`s_2[`,`noc]`)` replaces all substrings within the Unicode string s_1 that match re with s_2 and returns the resulting string

`ustrregexrf(`s_1`,`re`,`s_2[`,`noc]`)` replaces the first substring within the Unicode string s_1 that matches re with s_2 and returns the resulting string

`ustrregexs(`n`)` subexpression n from a previous `ustrregexm()` match

`ustrreverse(`s`)` reverses the Unicode string s

`ustrright(`s`,`n`)` the last n Unicode characters of the Unicode string s

`ustrrpos(`s_1`,`s_2[`,`n]`)` the position in s_1 at which s_2 is last found; otherwise, `0`

`ustrrtrim(`s`)` remove trailing Unicode whitespace characters and blanks from the Unicode string s

`ustrsortkey(`s[`,`loc]`)` generates a null-terminated byte array that can be used by the `sort` command to produce the same order as `ustrcompare()`

`ustrsortkeyex(`s`,`loc`,`st`,`$case$`,`$cslv$`,`$norm$`,`num`,`alt`,`fr`)`
generates a null-terminated byte array that can be used by the `sort` command to produce the same order as `ustrcompare()`

`ustrtitle(`s[`,`loc]`)` a string with the first characters of Unicode words titlecased and other characters lowercased

`ustrto(`s`,`enc`,`$mode$`)` converts the Unicode string s in UTF-8 encoding to a string in encoding enc

`ustrtohex(`s[`,`n]`)` escaped hex digit string of s up to 200 Unicode characters

`ustrtoname(`s[`,`p]`)` string s translated into a Stata name

`ustrtrim(`s`)` removes leading and trailing Unicode whitespace characters and blanks from the Unicode string s

`ustrunescape(`s`)` the Unicode string corresponding to the escaped sequences of s

`ustrupper(`s[`,`loc]`)` uppercase all characters in string s under the given locale loc

`ustrword(`s`,`n[`,`loc]`)` the nth Unicode word in the Unicode string s

`ustrwordcount(`s[`,`loc]`)` the number of nonempty Unicode words in the Unicode string s

`usubinstr(`s_1`,`s_2`,`s_3`,`n`)` replaces the first n occurrences of the Unicode string s_2 with the Unicode string s_3 in s_1

`usubstr(`s`,`n_1`,`n_2`)` the Unicode substring of s, starting at n_1, for a length of n_2

`vec(`M`)` a column vector formed by listing the elements of M, starting with the first column and proceeding column by column

`vecdiag(`M`)` the row vector containing the diagonal of matrix M

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

`weekly(`s_1`,`s_2[`,`Y]`)` the e_w weekly date (weeks since 1960w1) corresponding to s_1 based on s_2 and Y; Y specifies *topyear*; see `date()`

`weibull(`a`,`b`,`x`)` the cumulative Weibull distribution with shape a and scale b

`weibull(`a`,`b`,`g`,`x`)` the cumulative Weibull distribution with shape a, scale b, and location g

`weibullden(`a`,`b`,`x`)` the probability density function of the Weibull distribution with shape a and scale b

`weibullden(`a`,`b`,`g`,`x`)` the probability density function of the Weibull distribution with shape a, scale b, and location g

`weibullph(`a`,`b`,`x`)`	the cumulative Weibull (proportional hazards) distribution with shape a and scale b
`weibullph(`a`,`b`,`g`,`x`)`	the cumulative Weibull (proportional hazards) distribution with shape a, scale b, and location g
`weibullphden(`a`,`b`,`x`)`	the probability density function of the Weibull (proportional hazards) distribution with shape a and scale b
`weibullphden(`a`,`b`,`g`,`x`)`	the probability density function of the Weibull (proportional hazards) distribution with shape a, scale b, and location g
`weibullphtail(`a`,`b`,`x`)`	the reverse cumulative Weibull (proportional hazards) distribution with shape a and scale b
`weibullphtail(`a`,`b`,`g`,`x`)`	the reverse cumulative Weibull (proportional hazards) distribution with shape a, scale b, and location g
`weibulltail(`a`,`b`,`x`)`	the reverse cumulative Weibull distribution with shape a and scale b
`weibulltail(`a`,`b`,`g`,`x`)`	the reverse cumulative Weibull distribution with shape a, scale b, and location g
`wofd(`e_d`)`	the e_w weekly date (weeks since 1960w1) containing date e_d
`word(`s`,`n`)`	the nth word in s; *missing* (`""`) if n is missing
`wordbreaklocale(`*loc*`,`*type*`)`	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*
`wordcount(`s`)`	the number of words in s
`year(`e_d`)`	the numeric year corresponding to date e_d
`yearly(`s_1`,`s_2 [`,`Y]`)`	the e_y yearly date (year) corresponding to s_1 based on s_2 and Y; Y specifies *topyear*; see `date()`
`yh(`Y`,`H`)`	the e_h half-yearly date (half-years since 1960h1) corresponding to year Y, half-year H
`ym(`Y`,`M`)`	the e_m monthly date (months since 1960m1) corresponding to year Y, month M
`yofd(`e_d`)`	the e_y yearly date (year) containing date e_d
`yq(`Y`,`Q`)`	the e_q quarterly date (quarters since 1960q1) corresponding to year Y, quarter Q
`yw(`Y`,`W`)`	the e_w weekly date (weeks since 1960w1) corresponding to year Y, week W

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**

Title

Date and time functions

Contents Functions Remarks and examples Methods and formulas
References Also see

Contents

`age(`$e_{d\,\mathrm{DOB}}$`,`e_d [`,`s_{nl}]`)`	the age in integer years on e_d for date of birth $e_{d\,\mathrm{DOB}}$ with s_{nl} the nonleap-year birthday for 29feb birthdates
`age_frac(`$e_{d\,\mathrm{DOB}}$`,`e_d [`,`s_{nl}]`)`	the age in years, including the fractional part, on e_d for date of birth $e_{d\,\mathrm{DOB}}$ with s_{nl} the nonleap-year birthday for 29feb birthdates
`birthday(`$e_{d\,\mathrm{DOB}}$`,`Y [`,`s_{nl}]`)`	the e_d date of the birthday in year Y for date of birth $e_{d\,\mathrm{DOB}}$ with s_{nl} the nonleap-year birthday for 29feb birthdates
`bofd("`*cal*`",`e_d`)`	the e_b business date corresponding to e_d
`Cdhms(`e_d`,`h`,`m`,`s`)`	the e_{tC} datetime (ms. with leap seconds since 01jan1960 00:00:00.000) corresponding to e_d, h, m, s
`Chms(`h`,`m`,`s`)`	the e_{tC} datetime (ms. with leap seconds since 01jan1960 00:00:00.000) corresponding to h, m, s on 01jan1960
`Clock(`s_1`,`s_2 [`,`Y]`)`	the e_{tC} datetime (ms. with leap seconds since 01jan1960 00:00:00.000) corresponding to s_1 based on s_2 and Y
`clock(`s_1`,`s_2 [`,`Y]`)`	the e_{tc} datetime (ms. since 01jan1960 00:00:00.000) corresponding to s_1 based on s_2 and Y
`Clockdiff(`e_{tC1}`, `e_{tC2}`,`s_u`)`	the e_{tC} datetime difference, rounded down to an integer, from e_{tC1} to e_{tC2} in s_u units of days, hours, minutes, seconds, or milliseconds
`clockdiff(`e_{tc1}`,`e_{tc2}`,`s_u`)`	the e_{tc} datetime difference, rounded down to an integer, from e_{tc1} to e_{tc2} in s_u units of days, hours, minutes, seconds, or milliseconds
`Clockdiff_frac(`e_{tC1}`,`e_{tC2}`,`s_u`)`	the e_{tC} datetime difference, including the fractional part, from e_{tC1} to e_{tC2} in s_u units of days, hours, minutes, seconds, or milliseconds
`clockdiff_frac(`e_{tc1}`,`e_{tc2}`,`s_u`)`	the e_{tc} datetime difference, including the fractional part, from e_{tc1} to e_{tc2} in s_u units of days, hours, minutes, seconds, or milliseconds
`Clockpart(`e_{tC}`,`s_u`)`	the integer year, month, day, hour, minute, second, or millisecond of e_{tC} with s_u specifying which time part
`clockpart(`e_{tc}`,`s_u`)`	the integer year, month, day, hour, minute, second, or millisecond of e_{tc} with s_u specifying which time part
`Cmdyhms(`M`,`D`,`Y`,`h`,`m`,`s`)`	the e_{tC} datetime (ms. with leap seconds since 01jan1960 00:00:00.000) corresponding to M, D, Y, h, m, s
`Cofc(`e_{tc}`)`	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)
`cofC(`e_{tC}`)`	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)

`Cofd(`e_d`)` — the e_{tC} datetime (ms. with leap seconds since 01jan1960 00:00:00.000) of date e_d at time 00:00:00.000

`cofd(`e_d`)` — the e_{tc} datetime (ms. since 01jan1960 00:00:00.000) of date e_d at time 00:00:00.000

`daily(`s_1`,`s_2[`,`Y]`)` — a synonym for `date(`s_1`,`s_2[`,`Y]`)`

`date(`s_1`,`s_2[`,`Y]`)` — the e_d date (days since 01jan1960) corresponding to s_1 based on s_2 and Y

`datediff(`e_{d1}`,`e_{d2}`,`s_u[`,`s_{nl}]`)` — the difference, rounded down to an integer, from e_{d1} to e_{d2} in s_u units of days, months, or years with s_{nl} the nonleap-year anniversary for e_{d1} on 29feb

`datediff_frac(`e_{d1}`,`e_{d2}`,`s_u[`,`s_{nl}]`)` — the difference, including the fractional part, from e_{d1} to e_{d2} in s_u units of days, months, or years with s_{nl} the nonleap-year anniversary for e_{d1} on 29feb

`datepart(`e_d`,`s_u`)` — the integer year, month, or day of e_d with s_u specifying year, month, or day

`day(`e_d`)` — the numeric day of the month corresponding to e_d

`daysinmonth(`e_d`)` — the number of days in the month of e_d

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

`dofb(`e_b`,"`*cal*`")` — the e_d datetime corresponding to e_b

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

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

`dofh(`e_h`)` — the e_d date (days since 01jan1960) of the start of half-year e_h

`dofm(`e_m`)` — the e_d date (days since 01jan1960) of the start of month e_m

`dofq(`e_q`)` — the e_d date (days since 01jan1960) of the start of quarter e_q

`dofw(`e_w`)` — the e_d date (days since 01jan1960) of the start of week e_w

`dofy(`e_y`)` — the e_d date (days since 01jan1960) of 01jan in year e_y

`dow(`e_d`)` — the numeric day of the week corresponding to date e_d; $0 =$ Sunday, $1 =$ Monday, ..., $6 =$ Saturday

`doy(`e_d`)` — the numeric day of the year corresponding to date e_d

`firstdayofmonth(`e_d`)` — the e_d date of the first day of the month of e_d

`halfyear(`e_d`)` — the numeric half of the year corresponding to date e_d

`halfyearly(`s_1`,`s_2[`,`Y]`)` — 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()`

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

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

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

`hofd(`e_d`)` — the e_h half-yearly date (half years since 1960h1) containing date e_d

`hours(`ms`)` — $ms/3{,}600{,}000$

isleapsecond(e_{tC})	1 if e_{tC} is a leap second; otherwise, 0
isleapyear(Y)	1 if Y is a leap year; otherwise, 0
lastdayofmonth(e_d)	the e_d date of the last day of the month of e_d
mdy(M,D,Y)	the e_d date (days since 01jan1960) corresponding to M, D, Y
mdyhms(M,D,Y,h,m,s)	the e_{tc} datetime (ms. since 01jan1960 00:00:00.000) corresponding to M, D, Y, h, m, s
minutes(ms)	$ms/60{,}000$
mm(e_{tc})	the minute corresponding to datetime e_{tc} (ms. since 01jan1960 00:00:00.000)
mmC(e_{tC})	the minute corresponding to datetime e_{tC} (ms. with leap seconds since 01jan1960 00:00:00.000)
mofd(e_d)	the e_m monthly date (months since 1960m1) containing date e_d
month(e_d)	the numeric month corresponding to date e_d
monthly(s_1,s_2[,Y])	the e_m monthly date (months since 1960m1) corresponding to s_1 based on s_2 and Y; Y specifies *topyear*; see date()
msofhours(h)	$h \times 3{,}600{,}000$
msofminutes(m)	$m \times 60{,}000$
msofseconds(s)	$s \times 1{,}000$
nextbirthday($e_{d\,\mathrm{DOB}}$,e_d[,s_{nl}])	the e_d date of the first birthday after e_d for date of birth $e_{d\,\mathrm{DOB}}$ with s_{nl} the nonleap-year birthday for 29feb birthdates
nextleapyear(Y)	the first leap year after year Y
now()	the current e_{tc} datetime
previousbirthday($e_{d\,\mathrm{DOB}}$,e_d[,s_{nl}])	the e_d date of the birthday immediately before e_d for date of birth $e_{d\,\mathrm{DOB}}$ with s_{nl} the nonleap-year birthday for 29feb birthdates
previousleapyear(Y)	the leap year immediately before year Y
qofd(e_d)	the e_q quarterly date (quarters since 1960q1) containing date e_d
quarter(e_d)	the numeric quarter of the year corresponding to date e_d
quarterly(s_1,s_2[,Y])	the e_q quarterly date (quarters since 1960q1) corresponding to s_1 based on s_2 and Y; Y specifies *topyear*; see date()
seconds(ms)	$ms/1{,}000$
ss(e_{tc})	the second corresponding to datetime e_{tc} (ms. since 01jan1960 00:00:00.000)
ssC(e_{tC})	the second corresponding to datetime e_{tC} (ms. with leap seconds since 01jan1960 00:00:00.000)
tC(l)	convenience function to make typing dates and times in expressions easier
tc(l)	convenience function to make typing dates and times in expressions easier
td(l)	convenience function to make typing dates in expressions easier
th(l)	convenience function to make typing half-yearly dates in expressions easier
tm(l)	convenience function to make typing monthly dates in expressions easier

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

Functions

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

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

`age(`$e_{d\,\mathrm{DOB}}$`,`e_d [`,`s_{nl}]`)`

Description: the age in integer years on e_d for date of birth $e_{d\,\mathrm{DOB}}$ with s_{nl} the nonleap-year birthday for 29feb birthdates

s_{nl} specifies when someone born on 29feb becomes another year older in nonleap years. s_{nl} = `"01mar"` (the default) means the birthday is taken to be 01mar. s_{nl} = `"28feb"` means the birthday is taken to be 28feb. See *Methods and formulas*.

When $e_d < e_{d\,\mathrm{DOB}}$, the result is *missing*.

Domain $e_{d\,\mathrm{DOB}}$: e_d dates 01jan0101 to 31dec9998 (integers −678,985 to 2,936,184)
Domain e_d: e_d dates 01jan0101 to 31dec9998 (integers −678,985 to 2,936,184)
Domain s_{nl}: strings `"28feb"`, `"feb28"`, `"01mar"`, `"1mar"`, `"mar01"`, and `"mar1"` (case insensitive)
Range: integers 0 to 9897 or *missing*

`age_frac(`$e_{d\,\mathrm{DOB}}$`,`e_d [`,`s_{nl}]`)`

Description: the age in years, including the fractional part, on e_d for date of birth $e_{d\,\mathrm{DOB}}$ with s_{nl} the nonleap-year birthday for 29feb birthdates

s_{nl} specifies when someone born on 29feb becomes another year older in nonleap years. s_{nl} = `"01mar"` (the default) means the birthday is taken to be 01mar. s_{nl} = `"28feb"` means the birthday is taken to be 28feb. See *Methods and formulas*.

When $e_d < e_{d\,\mathrm{DOB}}$, the result is *missing*.

Domain $e_{d\,\mathrm{DOB}}$: e_d dates 01jan0101 to 31dec9998 (integers −678,985 to 2,936,184)
Domain e_d: e_d dates 01jan0101 to 31dec9998 (integers −678,985 to 2,936,184)
Domain s_{nl}: strings `"28feb"`, `"feb28"`, `"01mar"`, `"1mar"`, `"mar01"`, and `"mar1"` (case insensitive)
Range: reals 0 to 9897.997... or *missing*

`birthday(`$e_{d\,\mathrm{DOB}}$`,`Y [`,`s_{nl}]`)`

Description: the e_d date of the birthday in year Y for date of birth $e_{d\,\mathrm{DOB}}$ with s_{nl} the nonleap-year birthday for 29feb birthdates

s_{nl} specifies when someone born on 29feb becomes another year older in nonleap years. s_{nl} = `"01mar"` (the default) means the birthday is taken to be 01mar. s_{nl} = `"28feb"` means the birthday is taken to be 28feb. See *Methods and formulas*.

Domain $e_{d\,\mathrm{DOB}}$: e_d dates 01jan0100 to 31dec9999 (integers −679,350 to 2,936,549)
Domain Y: integers 0100 to 9999 (but probably 1800 to 2100)
Domain s_{nl}: strings `"28feb"`, `"feb28"`, `"01mar"`, `"1mar"`, `"mar01"`, and `"mar1"` (case insensitive)
Range: e_d dates 01jan0100 to 31dec9999 (integers −679,350 to 2,936,549) or *missing*

`bofd("`*cal*`",`e_d`)`
- Description: the e_b business date corresponding to e_d
- Domain *cal*: business calendar names and formats
- Domain e_d: e_d 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: e_d 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: 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+number of leap seconds) 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: 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+number of leap seconds) 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 t_C value rather than an unadjusted t_c 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: 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+number of leap seconds) 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 = `"MD19Y hm"` would translate s_1 = `"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: 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) or *missing*

`Clockdiff(`e_{tC1}`,`e_{tC2}`,`s_u`)`

Description: the e_{tC} datetime difference, rounded down to an integer, from e_{tC1} to e_{tC2} in s_u units of days, hours, minutes, seconds, or milliseconds

Note that `Clockdiff(`e_{tC1}`,`e_{tC2}`,`s_u`)` = −`Clockdiff(`e_{tC2}`,`e_{tC1}`,`s_u`)`.

Domain e_{tC1}: 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+number of leap seconds)
Domain e_{tC2}: 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+number of leap seconds)
Domain s_u: strings `"day"` or `"d"` for day; `"hour"` or `"h"` for hour; `"minute"`, `"min"`, or `"m"` for minute; `"second"`, `"sec"`, or `"s"` for second; and `"millisecond"` or `"ms"` for millisecond (case insensitive)
Range: integers −312,413,759,999,999 − number of leap seconds to 312,413,759,999,999 + number of leap seconds or *missing*

`clockdiff(`e_{tc1}`,`e_{tc2}`,`s_u`)`

Description: the e_{tc} datetime difference, rounded down to an integer, from e_{tc1} to e_{tc2} in s_u units of days, hours, minutes, seconds, or milliseconds

Note that `clockdiff(`e_{tc1}`,`e_{tc2}`,`s_u`)` = −`clockdiff(`e_{tc2}`,`e_{tc1}`,`s_u`)`.

Domain e_{tc1}: 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)
Domain e_{tc2}: 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)
Domain s_u: strings `"day"` or `"d"` for day; `"hour"` or `"h"` for hour; `"minute"`, `"min"`, or `"m"` for minute; `"second"`, `"sec"`, or `"s"` for second; and `"millisecond"` or `"ms"` for millisecond (case insensitive)
Range: integers −312,413,759,999,999 to 312,413,759,999,999 or *missing*

`Clockdiff_frac(`e_{tC1}`,`e_{tC2}`,`s_u`)`

Description: the e_{tC} datetime difference, including the fractional part, from e_{tC1} to e_{tC2} in s_u units of days, hours, minutes, seconds, or milliseconds

Note that `Clockdiff_frac(`e_{tC1}`,`e_{tC2}`,`s_u`)` = −`Clockdiff_frac(`e_{tC2}`,`e_{tC1}`,`s_u`)`.

Domain e_{tC1}: 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+number of leap seconds)

Domain e_{tC2}: 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+number of leap seconds)

Domain s_u: strings `"day"` or `"d"` for day; `"hour"` or `"h"` for hour; `"minute"`, `"min"`, or `"m"` for minute; `"second"`, `"sec"`, or `"s"` for second; and `"millisecond"` or `"ms"` for millisecond (case insensitive)

Range: reals −312,413,759,999,999 − number of leap seconds to 312,413,759,999,999 + number of leap seconds or *missing*

`clockdiff_frac(`e_{tc1}`,`e_{tc2}`,`s_u`)`

Description: the e_{tc} datetime difference, including the fractional part, from e_{tc1} to e_{tc2} in s_u units of days, hours, minutes, seconds, or milliseconds

Note that `clockdiff_frac(`e_{tc1}`,`e_{tc2}`,`s_u`)` = −`clockdiff_frac(`e_{tc2}`,`e_{tc1}`,`s_u`)`.

Domain e_{tc1}: 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)

Domain e_{tc2}: 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)

Domain s_u: strings `"day"` or `"d"` for day; `"hour"` or `"h"` for hour; `"minute"`, `"min"`, or `"m"` for minute; `"second"`, `"sec"`, or `"s"` for second; and `"millisecond"` or `"ms"` for millisecond (case insensitive)

Range: reals −312,413,759,999,999 to 312,413,759,999,999 or *missing*

`Clockpart(`e_{tC}`,`s_u`)`

Description: the integer year, month, day, hour, minute, second, or millisecond of e_{tC} with s_u specifying which time part

Domain e_{tC}: 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+number of leap seconds)

Domain s_u: strings `"year"` or `"y"` for year; `"month"` or `"mon"` for month; `"day"` or `"d"` for day; `"hour"` or `"h"` for hour; `"minute"` or `"min"` for minute; `"second"`, `"sec"`, or `"s"` for second; and `"millisecond"` or `"ms"` for millisecond (case insensitive)

Range: integers 0 to 9999 or *missing*

`clockpart(`e_{tc}`,`s_u`)`

Description: the integer year, month, day, hour, minute, second, or millisecond of e_{tc} with s_u specifying which time part

Domain e_{tc}: 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)

Domain s_u: strings `"year"` or `"y"` for year; `"month"` or `"mon"` for month; `"day"` or `"d"` for day; `"hour"` or `"h"` for hour; `"minute"` or `"min"` for minute; `"second"`, `"sec"`, or `"s"` for second; and `"millisecond"` or `"ms"` for millisecond (case insensitive)

Range: integers 0 to 9999 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: 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+number of leap seconds) 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}: 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: 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+number of leap seconds)

`cofC(`e_{tC}`)`

Description: 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)
Domain e_{tC}: 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+number of leap seconds)
Range 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)

`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: e_d dates 01jan0100 to 31dec9999 (integers −679,350 to 2,936,549)
Range: 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+number of leap seconds)

`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: e_d dates 01jan0100 to 31dec9999 (integers −679,350 to 2,936,549)
Range: 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)

`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 = `"MD19Y"` would translate s_1 = `"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 conversion** for more information.

Domain s_1: strings
Domain s_2: strings
Domain Y: integers 1000 to 9998 (but probably 2001 to 2099)
Range: e_d dates 01jan0100 to 31dec9999 (integers −679,350 to 2,936,549) or *missing*

`datediff(`e_{d1}`,`e_{d2}`,`s_u[`,`s_{nl}]`)`

Description: the difference, rounded down to an integer, from e_{d1} to e_{d2} in s_u units of days, months, or years with s_{nl} the nonleap-year anniversary for e_{d1} on 29feb

s_{nl} specifies the anniversary when e_{d1} is on 29feb. s_{nl} = `"01mar"` (the default) means the anniversary is taken to be 01mar. s_{nl} = `"28feb"` means the anniversary is taken to be 28feb. See *Methods and formulas*.

Note that `datediff(`e_{d1}`,`e_{d2}`,`s_u`,`s_{nl}`)` = `−datediff(`e_{d2}`,`e_{d1}`,`s_u`,`s_{nl}`)`.

Domain e_{d1}: e_d dates 01jan0101 to 31dec9998 (integers −678,985 to 2,936,184)
Domain e_{d2}: e_d dates 01jan0101 to 31dec9998 (integers −678,985 to 2,936,184)
Domain s_u: strings `"day"` or `"d"` for day; `"month"`, `"mon"`, or `"m"` for month; and `"year"` or `"y"` for year (case insensitive)
Domain s_{nl}: strings `"28feb"`, `"feb28"`, `"01mar"`, `"1mar"`, `"mar01"`, and `"mar1"` (case insensitive)
Range: integers −3,615,169 to 3,615,169 or *missing*

`datediff_frac(`e_{d1}`,`e_{d2}`,`s_u[`,`s_{nl}]`)`

Description: the difference, including the fractional part, from e_{d1} to e_{d2} in s_u units of days, months, or years with s_{nl} the nonleap-year anniversary for e_{d1} on 29feb

s_{nl} specifies the anniversary when e_{d1} is on 29feb. s_{nl} = `"01mar"` (the default) means the anniversary is taken to be 01mar. s_{nl} = `"28feb"` means the anniversary is taken to be 28feb. See *Methods and formulas*.

Note that `datediff_frac(`e_{d1}`,`e_{d2}`,`s_u`,`s_{nl}`)` = `-datediff_frac(`e_{d2}`,`e_{d1}`,`s_u`,`s_{nl}`)`.

Domain e_{d1}: e_d dates 01jan0101 to 31dec9998 (integers −678,985 to 2,936,184)
Domain e_{d2}: e_d dates 01jan0101 to 31dec9998 (integers −678,985 to 2,936,184)
Domain s_u: strings `"day"` or `"d"` for day; `"month"`, `"mon"`, or `"m"` for month; and `"year"` or `"y"` for year (case insensitive)
Domain s_{nl}: strings `"28feb"`, `"feb28"`, `"01mar"`, `"1mar"`, `"mar01"`, and `"mar1"` (case insensitive)
Range: reals −3,615,169 to 3,615,169 or *missing*

`datepart(`e_d`,`s_u`)`

Description: the integer year, month, or day of e_d with s_u specifying year, month, or day
Domain e_d: e_d dates 01jan0100 to 31dec9999 (integers −679,350 to 2,936,549)
Domain s_u: strings `"day"` or `"d"` for day; `"month"`, `"mon"`, or `"m"` for month; and `"year"` or `"y"` for year (case insensitive)
Range: integers 1 to 9999 or *missing*

`day(`e_d`)`

Description: the numeric day of the month corresponding to e_d
Domain e_d: e_d dates 01jan0100 to 31dec9999 (integers −679,350 to 2,936,549)
Range: integers 1 to 31 or *missing*

`daysinmonth(`e_d`)`

Description: the number of days in the month of e_d
Domain e_d: e_d dates 01jan0100 to 31dec9999 (integers −679,350 to 2,936,549)
Range: integers 28 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 s
Domain e_d: e_d 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: 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) or *missing*

`dofb(`e_b`,"`*cal*`")`

Description: the e_d datetime corresponding to e_b
Domain e_b: e_b 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}: 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+number of leap seconds)
Range: e_d 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}: 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: e_d 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: e_h dates 0100h1 to 9999h2 (integers −3,720 to 16,079)
Range: e_d 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: e_m dates 0100m1 to 9999m12 (integers −22,320 to 96,479)
Range: e_d 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: e_q dates 0100q1 to 9999q4 (integers −7,440 to 32,159)
Range: e_d 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: e_w dates 0100w1 to 9999w52 (integers −96,720 to 418,079)
Range: e_d 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: e_y dates 0100 to 9999 (integers 0100 to 9999)
Range: e_d 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: e_d 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: e_d dates 01jan0100 to 31dec9999 (integers −679,350 to 2,936,549)
Range: integers 1 to 366 or *missing*

`firstdayofmonth(`e_d`)`

Description: the e_d date of the first day of the month of e_d
Domain e_d: e_d dates 01jan0100 to 31dec9999 (integers −679,350 to 2,936,549)
Range: e_d dates 01jan0100 to 01dec9999 (integers −679,350 to 2,936,519) or *missing*

`halfyear(`e_d`)`

Description: the numeric half of the year corresponding to date e_d
Domain e_d: e_d 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: e_h 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}: 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 or *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}: 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+number of leap seconds)
Range: integers 0 through 23 or *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: e_d dates 01jan0100 to 31dec9999 (integers −679,350 to 2,936,549)
Range: e_h 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*

`isleapsecond(`e_{tC}`)`

Description: 1 if e_{tC} is a leap second; otherwise, 0
Domain e_{tC}: 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+number of leap seconds)
Range: 0, 1, or *missing*

`isleapyear(`Y`)`

Description: 1 if Y is a leap year; otherwise, 0
Domain Y: integers 0100 to 9999 (but probably 1800 to 2100)
Range: 0, 1, or *missing*

`lastdayofmonth(`e_d`)`

Description: the e_d date of the last day of the month of e_d
Domain e_d: e_d dates 01jan0100 to 31dec9999 (integers −679,350 to 2,936,549)
Range: e_d dates 31jan0100 to 31dec9999 (integers −679,320 to 2,936,549) 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: e_d 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: 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) 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}: 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 or *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}: 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+number of leap seconds)
Range: integers 0 through 59 or *missing*

`mofd(`e_d`)`

Description: the e_m monthly date (months since 1960m1) containing date e_d
Domain e_d: e_d dates 01jan0100 to 31dec9999 (integers −679,350 to 2,936,549)
Range: e_m 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: e_d 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: e_m 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

`nextbirthday(`$e_{d\,\mathrm{DOB}}$`,`e_d[`,`s_{nl}]`)`

Description: the e_d date of the first birthday after e_d for date of birth $e_{d\,\mathrm{DOB}}$ with s_{nl} the nonleap-year birthday for 29feb birthdates

s_{nl} specifies when someone born on 29feb becomes another year older in nonleap years. s_{nl} = `"01mar"` (the default) means the birthday is taken to be 01mar. s_{nl} = `"28feb"` means the birthday is taken to be 28feb. See *Methods and formulas*.

Domain $e_{d\,\mathrm{DOB}}$: e_d dates 01jan0100 to 31dec9999 (integers −679,350 to 2,936,549)
Domain e_d: e_d dates 01jan0100 to 31dec9999 (integers −679,350 to 2,936,549)
Domain s_{nl}: strings `"28feb"`, `"feb28"`, `"01mar"`, `"1mar"`, `"mar01"`, and `"mar1"` (case insensitive)
Range: e_d dates 01jan0101 to 31dec9999 (integers −678,985 to 2,936,549) or *missing*

`nextleapyear(`Y`)`

Description: the first leap year after year Y
Domain Y: integers 0100 to 9999 (but probably 1800 to 2100)
Range: integers 1584 to 9996 or *missing*

`now()`

Description: the current e_{tc} datetime

Result matches `c(current_date)` with `c(current_time)` added. Precisely, `now()` returns `clock(c(current_date) + c(current_time), "DMYhms")`. See [P] creturn.

Range: 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)

`previousbirthday(`$e_{d\,\mathrm{DOB}}$`,`e_d [`,`s_{nl}]`)`

Description: the e_d date of the birthday immediately before e_d for date of birth $e_{d\,\mathrm{DOB}}$ with s_{nl} the nonleap-year birthday for 29feb birthdates

s_{nl} specifies when someone born on 29feb becomes another year older in nonleap years. s_{nl} = `"01mar"` (the default) means the birthday is taken to be 01mar. s_{nl} = `"28feb"` means the birthday is taken to be 28feb. See *Methods and formulas*.

Domain $e_{d\,\mathrm{DOB}}$: e_d dates 01jan0100 to 31dec9999 (integers −679,350 to 2,936,549)

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

Domain s_{nl}: strings `"28feb"`, `"feb28"`, `"01mar"`, `"1mar"`, `"mar01"`, and `"mar1"` (case insensitive)

Range: e_d dates 01jan0100 to 31dec9998 (integers −679,350 to 2,936,184) or *missing*

`previousleapyear(`Y`)`

Description: the leap year immediately before year Y

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

Range: integers 1584 to 9996 or *missing*

`qofd(`e_d`)`

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

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

Range: e_q 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: e_d 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: e_q 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}: 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 or *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}: 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+number of leap seconds)

Range: real 0.000 through 60.999 or *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: 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+number of leap seconds)

`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: 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)

`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: e_d 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: e_h 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: e_m dates 0100m1 to 9999m12 (integers $-22{,}320$ to 96,479) `today()`

Description: today's e_d date

Range: e_d dates 01jan0100 to 31dec9999 (integers −679,350 to 2,936,549)

`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: e_q 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: e_w 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: e_d dates 01jan0100 to 31dec9999 (integers −679,350 to 2,936,549)

Range: integers 1 to 52 or *missing*

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

Description: the e_w weekly date (weeks since 1960w1) corresponding to s_1 based on s_2 and Y; Y specifies *topyear*; see `date()`

Domain s_1: strings

Domain s_2: strings `"WY"` and `"YW"`; `Y` may be prefixed with *##*

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

Range: e_w 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: e_d dates 01jan0100 to 31dec9999 (integers −679,350 to 2,936,549)

Range: e_w 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: e_d dates 01jan0100 to 31dec9999 (integers −679,350 to 2,936,549)

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

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

Description: the e_y yearly date (year) corresponding to s_1 based on s_2 and Y; Y specifies *topyear*; see `date()`

Domain s_1: strings

Domain s_2: string `"Y"`; `Y` may be prefixed with *##*

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

Range: e_y 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: e_h 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: e_m 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: e_d dates 01jan0100 to 31dec9999 (integers $-679{,}350$ to 2,936,549)
- Range: e_y 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: e_q 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: e_w dates 1000w1 to 9999w52 (integers $-49{,}920$ to 418,079)

Remarks and examples

Stata's date and time functions are described with examples in [U] **25 Working with dates and times**, [D] **Datetime**, [D] **Datetime durations**, and [D] **Datetime relative dates**.

Video example

How to create a date variable from a date stored as a string

Methods and formulas

The functions `age()` and `age_frac()` are based on `datediff()` and `datediff_frac()`, respectively,

$$\texttt{age(}e_{d\,\text{DOB}}\texttt{,}e_d\texttt{,}s_{nl}\texttt{)} = \texttt{datediff(}e_{d\,\text{DOB}}\texttt{,}e_d\texttt{,"year",}s_{nl}\texttt{)}$$

and

$$\texttt{age\_frac(}e_{d\,\text{DOB}}\texttt{,}e_d\texttt{,}s_{nl}\texttt{)} = \texttt{datediff\_frac(}e_{d\,\text{DOB}}\texttt{,}e_d\texttt{,"year",}s_{nl}\texttt{)}$$

when $e_d \geq e_{d\,\text{DOB}}$. When $e_d < e_{d\,\text{DOB}}$, `age()` and `age_frac()` return *missing* (`.`).

`datediff(`e_{d1}`,`e_{d2}`,"year",`s_{nl}`)` returns an integer that is the number of years between e_{d1} and e_{d2}. Assume $e_{d2} \geq e_{d1}$. If the month and day of e_{d2} are the same or after the month and day of e_{d1}, it returns `year(`e_{d2}`)` − `year(`e_{d1}`)`. If the month and day of e_{d2} are before the month and day of e_{d1}, it returns `year(`e_{d2}`)` − `year(`e_{d1}`)` − 1.

If $e_{d2} < e_{d1}$, the result is calculated using

$$\texttt{datediff(}e_{d1}\texttt{,}e_{d2}\texttt{,"year",}s_{nl}\texttt{)} = -\texttt{datediff(}e_{d2}\texttt{,}e_{d1}\texttt{,"year",}s_{nl}\texttt{)}$$

This formula also holds for units of `"month"` and `"day"` and for `datediff_frac()`.

`datediff(`e_{d1}`,`e_{d2}`,"year",`s_{nl}`)` has an optional fourth argument, s_{nl}, that applies only to a starting date e_{d1} on 29feb when the ending date e_{d2} is not in a leap year. There are two possible values for s_{nl}: either `"01mar"` (with equivalents `"1mar"`, `"mar01"`, `"mar1"`) or `"28feb"` (`"feb28"`). When `"01mar"` is specified and e_{d1} is on 29feb, `datediff()` increases by one in nonleap years when e_{d2} goes to 01mar. When `"28feb"` is specified and e_{d1} is on 29feb, it increases by one in nonleap years when e_{d2} goes to 28feb.

In other words, s_{nl} sets the anniversary date (or birthday) in nonleap years for starting dates (or dates of birth) on 29feb. When the fourth argument is omitted, it is as if `"01mar"` was specified.

Regardless of the value of s_{nl}, when e_{d1} is on 29feb, `datediff(...,"year",...)` increases by one in leap years when e_{d2} goes to 29feb.

`datediff_frac(`e_{d1}`,`e_{d2}`,"year",`s_{nl}`)` is defined similarly. `datediff_frac(...,"year",...)` is exactly an integer and equal to `datediff(...,"year",...)` for days e_{d2} on which `datediff()` increases by one from the day previous to e_{d2}.

The fractional part of `datediff_frac(`e_{d1}`,`e_{d2}`,"year",`s_{nl}`)` is calculated by first counting the number of days, d_1, from the closest date prior to e_{d2} that has an exact integer value of `datediff_frac(...,"year",...)` to e_{d2}. Then number of the days, d_2, from e_{d2} to the closest following date that has an exact integer value of `datediff_frac()` is determined. The fractional part is $d_1/(d_1 + d_2)$, and $d_1 + d_2$ is either 365 or 366.

For examples, see example 1 and example 3 in [D] **Datetime durations**.

`datediff(`e_{d1}`,`e_{d2}`,"month",`s_{nl}`)` and `datediff_frac(`e_{d1}`,`e_{d2}`,"month",`s_{nl}`)` follow the corresponding definitions with `"year"`. `datediff(...,"month",...)` increases to an integer multiple of 12 when `datediff(...,"year",...)` increases by one from the day previous to e_{d2}. `datediff_frac(...,"month",...)` is exactly 12 times `datediff_frac(...,"year",...)` when `datediff_frac(...,"year",...)` is an integer.

`datediff(`e_{d1}`,`e_{d2}`,"month",`s_{nl}`)` increases by one from the day previous to e_{d2} when `day(`e_{d2}`)` = `day(`e_{d1}`)`. If there is no `day(`e_{d1}`)` in the month, then it increases by one on the first day of the next month. For example, if e_{d1} is on 30aug, then `datediff(...,"month",...)` increases by one when e_{d2} goes to 30sep. If e_{d1} is on 31aug, then `datediff(...,"month",...)` increases by one when e_{d2} goes to 01oct.

The optional fourth argument, s_{nl}, again sets the date, either `"01mar"` or `"28feb"`, when `datediff(...,"month",...)` increases by one when e_{d1} is on 29feb.

`datediff_frac(...,"month",...)` is defined like `datediff_frac(..., "year",...)`. Days on which `datediff_frac(...,"month",...)` is an exact integer are determined, and the fractional part for other days is determined by interpolating between these days. The denominator of the fractional part is 28, 29, 30, or 31.

See example 2 of `datediff()` and `datediff_frac()` for months in [D] **Datetime durations**.

`datediff(`e_{d1}`,`e_{d2}`,"day",`s_{nl}`)` and `datediff_frac(`e_{d1}`,`e_{d2}`,"day",`s_{nl}`)` have no such complications. Both are equal to $e_{d2} - e_{d1}$ and are always integers. The optional fourth argument has no bearing on the calculation and is ignored.

`clockdiff(`e_{tc1}`,`e_{tc2}`,`s_u`)` and `clockdiff_frac(`e_{tc1}`,`e_{tc2}`,`s_u`)` take the difference $e_{tc2} - e_{tc1}$, which is in milliseconds, and converts the difference to the units specified by s_u, days ($24 \times 60 \times 60 \times 1000$ milliseconds), hours ($60 \times 60 \times 1000$ milliseconds), minutes (60×1000 milliseconds), or seconds (1000 milliseconds). `clockdiff()` rounds the result down to an integer, whereas `clockdiff_frac()` retains the fractional part of the difference.

`Clockdiff(`e_{tC1}`,`e_{tC2}`,`s_u`)` and `Clockdiff_frac(`e_{tC1}`,`e_{tC2}`,`s_u`)` are similar to `clockdiff()` and `clockdiff_frac()` except they are used with datetime/C values (times with leap seconds) rather than datetime/c values (times without leap seconds). In almost all cases, `Clockdiff()` and `Clockdiff_frac()` give the same results as `clockdiff()` and `clockdiff_frac()` with the datetime/C values converted to datetime/c values. They only differ when either or both of times e_{tC1} and e_{tC2} are close to a leap second and the units are days, hours, or minutes. By "close", we mean within a day, hour, or minute of the leap second, respectively, for the chosen unit, and less than or equal to the leap second.

Stata system file `leapseconds.maint` lists the dates on which leap seconds occurred. To view the file, type

```
. viewsource leapseconds.maint
```

For times close to leap seconds or times that are leap seconds, `Clockdiff()` and `Clockdiff_frac()` base their calculations on there being a minute consisting of 61 seconds, an hour of $60 \times 60 + 1 = 3{,}601$ seconds, and a day of $24 \times 60 \times 60 + 1 = 86{,}401$ seconds before the leap second (and including the leap second).

For example, `31dec2016 23:59:60` is a leap second, so the time difference between `31dec2016 23:59:00` and `01jan2017 00:00:00` is a minute that consists of 61 seconds. The time difference between e_{tC1} = `31dec2016 23:59:00` and e_{tC2} = `31dec2016 23:59:59` is 59 seconds. So `Clockdiff_frac(`e_{tC1}`,`e_{tC2}`,"minute")` $= 59/61 = 0.9672$ minute.

For times further away from the leap second, say, e_{tC1} = `31dec2016 23:58:00` and e_{tC2} = `01jan2017 00:02:01`, having a leap second between these times has no effect on the result. In this case, `Clockdiff_frac(`e_{tC1}`,`e_{tC2}`,"minute")` $= 4 + 1/60 = 4.0167$ minutes. `01jan2017 00:02:00` is considered the "anniversary" minute of `31dec2016 23:58:00`, so the difference between these times is exactly 4 minutes. Increasing the ending time by a second gives the result $4 + 1/60$ minutes. This is, of course, the same result produced by `clockdiff_frac(...,"minute")` with the datetime/C values converted to datetime/c.

For units of days or hours, the logic of the calculation is similar. For units of seconds or milliseconds, the results are straightforward. The arguments e_{tC1} and e_{tC2} are numbers of milliseconds, so

$$\texttt{Clockdiff\_frac(}e_{tC1}\texttt{,}e_{tC2}\texttt{,"millisecond")} = e_{tC2} - e_{tC1}$$

and

$$\texttt{Clockdiff\_frac(}e_{tC1}\texttt{,}e_{tC2}\texttt{,"second")} = (e_{tC2} - e_{tC1})/1000$$

References

Cox, N. J. 2018. Stata tip 130: 106610 and all that: Date variables that need to be fixed. *Stata Journal* 18: 755–757.

——. 2019. Speaking Stata: The last day of the month. *Stata Journal* 19: 719–728.

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] **Datetime durations** — Obtaining and working with durations

[D] **Datetime relative dates** — Obtaining dates and date information from other dates

[D] **egen** — Extensions to generate

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

[M-5] **date( )** — Date and time manipulation

[U] 13.3 Functions

[U] 25 Working with dates and times

Title

Mathematical functions

Contents Functions Video example References Also see

Contents

round(x,y) or round(x)	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
sign(x)	the sign of x: -1 if $x < 0$, 0 if $x = 0$, 1 if $x > 0$, or *missing* if x is missing
sqrt(x)	the square root of x
sum(x)	the running sum of x, treating missing values as zero
trigamma(x)	the second derivative of lngamma(x) = $d^2 \ln\Gamma(x)/dx^2$
trunc(x)	a synonym for int(x)

Functions

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

ceil(x)
  Description: the unique integer n such that $n - 1 < x \le n$; 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 8e+307
  Range: integers in $-8\text{e}+307$ to 8e+307

cloglog(x)
  Description: the complementary log-log of x

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

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

`floor(`x`)`
Description: the unique integer n such that $n \le x < n+1$; 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 8e+307
Range: integers in $-8\text{e}+307$ to 8e+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 $x \ge 0$. However, use of the `round()` function is preferred. Also see `round(`x`)`, `ceil(`x`)`, and `floor(`x`)`.
Domain: $-8\text{e}+307$ to 8e+307
Range: integers in $-8\text{e}+307$ to 8e+307

`invcloglog(`x`)`
Description: the inverse of the complementary log-log function of x

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

Domain: $-8\text{e}+307$ to 8e+307
Range: 0 to 1 or *missing*

`invlogit(`x`)`
Description: the inverse of the logit function of x

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

Domain: $-8\text{e}+307$ to 8e+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,
$\log_5(x) = \texttt{ln}(x)/\texttt{ln(5)} = \texttt{log}(x)/\texttt{log(5)} = \texttt{log10}(x)/\texttt{log10(5)}$
$\log_2(x) = \texttt{ln}(x)/\texttt{ln(2)} = \texttt{log}(x)/\texttt{log(2)} = \texttt{log10}(x)/\texttt{log10(2)}$

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: 1e–323 to 8e+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 $1-$ `c(epsdouble)`
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: $-1+$ `c(epsdouble)` to 8e+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 1e+305
Range: 0 to 8e+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 1e+305 (excluding negative integers)
Range: $-8\text{e}{+}307$ to 8e+307

`log(`x`)`
Description: a synonym for `ln(`x`)`

`log10(`x`)`
Description: the base-10 logarithm of x
Domain: 1e–323 to 8e+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, `logit(`x`)` $= \ln\{x/(1-x)\}$
Domain: 0 to 1 (exclusive)
Range: $-8\text{e}{+}307$ to 8e+307 or *missing*

`max(`x_1`,`x_2`,...,`x_n`)`
Description: the maximum value of $x_1, x_2, \ldots, x_n$

Unless all arguments are *missing*, missing values are ignored.
`max(2,10,.,7)` = `10`
`max(.,.,.)` = `.`

Domain x_1: −8e+307 to 8e+307 or *missing*
Domain x_2: −8e+307 to 8e+307 or *missing*
...
Domain x_n: −8e+307 to 8e+307 or *missing*
Range: −8e+307 to 8e+307 or *missing*

`min(`x_1`,`x_2`,...,`x_n`)`
Description: the minimum value of $x_1, x_2, \ldots, x_n$

Unless all arguments are *missing*, missing values are ignored.
`min(2,10,.,7)` = `2`
`min(.,.,.)` = `.`

Domain x_1: −8e+307 to 8e+307 or *missing*
Domain x_2: −8e+307 to 8e+307 or *missing*
...
Domain x_n: −8e+307 to 8e+307 or *missing*
Range: −8e+307 to 8e+307 or *missing*

`mod(`x`,`y`)`
Description: the modulus of x with respect to y

`mod(`x,y`)` = $x - y$ `floor(`x/y`)`
`mod(`x`,0)` = `.`

Domain x: −8e+307 to 8e+307
Domain y: 0 to 8e+307
Range: 0 to 8e+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: −8e+307 to 8e+307 or *missing*
Domain y: −8e+307 to 8e+307 or *missing*
Range: −8e+307 to 8e+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 $y \neq 1$. 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 jth observation on y contains the sum of the first through jth 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 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**

Title

Matrix functions

Contents Functions Reference Also see

Contents

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

`vec(`M`)` a column vector formed by listing the elements of M, starting with the first column and proceeding column by column

`vecdiag(`M`)` 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 $R =$ `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 $-$8e+307 to 8e+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 ith row/column swept

The row and column names of the resultant matrix are obtained from M, except that the nth row and column names are interchanged. If $B =$ `sweep(`A`,`k`)`, then

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

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 8e+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`[floor(`i`),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 8e+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 $\max_{i,j}\left\{|x_{ij}-y_{ij}|/(|y_{ij}|+1)\right\}$
Domain X: matrices
Domain Y: matrices with same number of rows and columns as X
Range: scalars −8e+307 to 8e+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(max_matdim)` or *missing*

`rownfreeparms(`M`)`
Description: the number of free parameters in rows of M
Domain: matrices
Range: integer scalars 0 to `c(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(max_matdim)` or *missing*

`rowsof(`M`)`
Description: the number of rows of M
Domain: matrices
Range: integer scalars 1 to `c(max_matdim)`

`trace(`M`)`
Description: the trace of matrix M
Domain: $n \times n$ (square) matrices
Range: scalars −8e+307 to 8e+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

Mazýa, 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

Title

Programming functions

Contents   Functions   References   Also see

Contents

`autocode(`x`,`n`,`x_0`,`x_1`)`	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 or the upper bound of the first or last interval if $x < x_0$ or $x > x_1$, respectively
`byteorder()`	`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
`c(`*name*`)`	the value of the system or constant result `c(`*name*`)` (see [P] **creturn**)
`_caller()`	`version` of the program or session that invoked the currently running program; see [P] **version**
`chop(`x`, `ϵ`)`	`round(`x`)` if `abs(`x $-$ `round(`x`))` $< \epsilon$; otherwise, x; or x if x is missing
`clip(`x`,`a`,`b`)`	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
`cond(`x`,`a`,`b[`,`c]`)`	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*
`e(`*name*`)`	the value of stored result `e(`*name*`)`; see [U] **18.8 Accessing results calculated by other programs**
`e(sample)`	`1` if the observation is in the estimation sample and `0` otherwise
`epsdouble()`	the machine precision of a double-precision number
`epsfloat()`	the machine precision of a floating-point number
`fileexists(`f`)`	`1` if the file specified by f exists; otherwise, `0`
`fileread(`f`)`	the contents of the file specified by f
`filereaderror(`s`)`	`0` or positive integer, said value having the interpretation of a return code
`filewrite(`f`,`s[`,`r]`)`	writes the string specified by s to the file specified by f and returns the number of bytes in the resulting file
`float(`x`)`	the value of x rounded to `float` precision
`fmtwidth(`*fmtstr*`)`	the output length of the *%fmt* contained in *fmtstr*; *missing* if *fmtstr* does not contain a valid *%fmt*
`frval()`	returns values of variables stored in other frames
`_frval()`	programmer's version of `frval()`
`has_eprop(`*name*`)`	`1` if *name* appears as a word in `e(properties)`; otherwise, `0`
`inlist(`z`,`a`,`b`,...)`	`1` if z is a member of the remaining arguments; otherwise, `0`
`inrange(`z`,`a`,`b`)`	`1` if it is known that $a \leq z \leq b$; otherwise, `0`
`irecode(`x`,`x_1`,...,`x_n`)`	*missing* if x is missing or $x_1, \ldots, x_n$ is not weakly increasing; `0` if $x \leq x_1$; `1` if $x_1 < x \leq x_2$; `2` if $x_2 < x \leq x_3$; $\ldots$; n if $x > x_n$

`matrix(`*exp*`)`	restricts name interpretation to scalars and matrices; see `scalar()`
`maxbyte()`	the largest value that can be stored in storage type `byte`
`maxdouble()`	the largest value that can be stored in storage type `double`
`maxfloat()`	the largest value that can be stored in storage type `float`
`maxint()`	the largest value that can be stored in storage type `int`
`maxlong()`	the largest value that can be stored in storage type `long`
`mi(`x_1`,`x_2`,...,`x_n`)`	a synonym for `missing(`x_1`,`x_2`,...,`x_n`)`
`minbyte()`	the smallest value that can be stored in storage type `byte`
`mindouble()`	the smallest value that can be stored in storage type `double`
`minfloat()`	the smallest value that can be stored in storage type `float`
`minint()`	the smallest value that can be stored in storage type `int`
`minlong()`	the smallest value that can be stored in storage type `long`
`missing(`x_1`,`x_2`,...,`x_n`)`	`1` if any x_i evaluates to *missing*; otherwise, `0`
`r(`*name*`)`	the value of the stored result `r(`*name*`)`; see [U] **18.8 Accessing results calculated by other programs**
`recode(`x`,`x_1`,...,`x_n`)`	*missing* if $x_1, x_2, \ldots, x_n$ is not weakly increasing; x if x is missing; x_1 if $x \leq x_1$; x_2 if $x \leq x_2$, ...; otherwise, x_n if $x > x_1$, x_2, ..., x_{n-1}. $x_i \geq .$ is interpreted as $x_i = +\infty$
`replay()`	`1` if the first nonblank character of local macro `` `0' `` is a comma, or if `` `0' `` is empty
`return(`*name*`)`	the value of the to-be-stored result `r(`*name*`)`; see [P] **return**
`s(`*name*`)`	the value of stored result `s(`*name*`)`; see [U] **18.8 Accessing results calculated by other programs**
`scalar(`*exp*`)`	restricts name interpretation to scalars and matrices
`smallestdouble()`	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 or the upper bound of the first or last interval if $x < x_0$ or $x > x_1$, respectively

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

$$
\begin{array}{l}
\text{if}\,(n \ge \texttt{.}\,|\,x_0 \ge \texttt{.}\,|\,x_1 \ge \texttt{.}\,|\,n \le 0\,|\,x_0 \ge x_1) \\
\quad \text{then return } \textit{missing} \\
\quad \text{if } x \ge \texttt{.}\text{, then return } x \\
\text{otherwise} \\
\quad \text{for } i = 1 \text{ to } n-1 \\
\qquad \textit{xmap} = x_0 + i * (x_1 - x_0)/n \\
\qquad \text{if } x \le \textit{xmap} \text{ then return } \textit{xmap} \\
\quad \text{end} \\
\quad \text{otherwise} \\
\qquad \text{return } x_1
\end{array}
$$

Domain x: $-8\text{e}+307$ to $8\text{e}+307$
Domain n: integers 1 to 10,000
Domain x_0: $-8\text{e}+307$ to $8\text{e}+307$
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 17.1, so 17.1 is the upper end of this range. If Stata 17.2 were the current version, 17.2 would be the upper end of this range, and likewise, if Stata 18 were the current version, 18 would be the upper end of this range. This is a function for use by programmers.

Range: 1 to 17.0

`chop(`x`,` ϵ`)`

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

Domain x: −8e+307 to 8e+307

Domain ϵ: −8e+307 to 8e+307

Range: −8e+307 to 8e+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: −8e+307 to 8e+307

Domain a: −8e+307 to 8e+307

Domain b: −8e+307 to 8e+307

Range: −8e+307 to 8e+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: −8e+307 to 8e+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 $d <$ `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 $d <$ `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: $-1e+38$ to $1e+38$

Range: $-1e+38$ to $1e+38$

`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*`,`*unm*`)`

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

`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 *unm* 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 *unm*: any numeric value if *var* is numeric; any string value when *var* is string

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

`_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 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 *frm*: any existing framename
Domain *var*: any existing variable name in *frm*; varname abbreviation is allowed
Domain *i*: any numeric values including missing values even though the nonmissing values should be integers in the range 1 to *frm*'s `_N`; 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 *var* in *frm* plus missing value; numeric missing value (`.`) when *var* is numeric, and string missing value (`""`) when *var* 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 *frm*: any existing framename
Domain *var*: any existing variable name in *frm*; varname abbreviation is allowed
Domain *i*: any numeric values including missing values even though the nonmissing values should be integers in the range 1 to *frm*'s `_N`; 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 *v*: any numeric value when *var* is numeric; any string value when *var* is string (can be a constant or vary observation by observation)
Range: range of *var* in *frm* plus *v*

`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`,`x_1`,`x_2`,`x_3`,...,`x_n`)`

Description: *missing* if x is missing or $x_1, \ldots, x_n$ 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: $-$8e+307 to 8e+307

Domain x_i: $-$8e+307 to 8e+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`

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

Range: one integer number

`maxlong()`

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

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

Range: one integer number

`mi(`x_1`,`x_2`,...,`x_n`)`

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

`minbyte()`

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

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

Range: one integer number

`mindouble()`

Description: the smallest 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

`minfloat()`

Description: the smallest 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

`minint()`

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

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

Range: one integer number

`minlong()`

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

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

Range: one integer number

`missing(`x_1`,`x_2`,...,`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`,...,`x_n`)`

Description: *missing* if $x_1, x_2, \ldots, x_n$ is not weakly increasing; x if x is missing; x_1 if $x \leq x_1$; x_2 if $x \leq x_2$, ...; otherwise, x_n if $x > x_1$, x_2, ..., 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, ..., 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 $0 < d <$ `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**

Title

Random-number functions

Contents | Functions | Remarks and examples | Methods and formulas
Acknowledgments | References | Also see

Contents

`rbeta(`a`,`b`)`	beta(a,b) random variates, where a and b are the beta distribution shape parameters
`rbinomial(`n`,`p`)`	binomial(n,p) random variates, where n is the number of trials and p is the success probability
`rcauchy(`a`,`b`)`	Cauchy(a,b) random variates, where a is the location parameter and b is the scale parameter
`rchi2(`*df*`)`	χ^2, with *df* degrees of freedom, random variates
`rexponential(`b`)`	exponential random variates with scale b
`rgamma(`a`,`b`)`	gamma(a,b) random variates, where a is the gamma shape parameter and b is the scale parameter
`rhypergeometric(`N`,`K`,`n`)`	hypergeometric random variates
`rigaussian(`m`,`a`)`	inverse Gaussian random variates with mean m and shape parameter a
`rlaplace(`m`,`b`)`	Laplace(m,b) random variates with mean m and scale parameter b
`rlogistic()`	logistic variates with mean 0 and standard deviation $\pi/\sqrt{3}$
`rlogistic(`s`)`	logistic variates with mean 0, scale s, and standard deviation $s\pi/\sqrt{3}$
`rlogistic(`m`,`s`)`	logistic variates with mean m, scale s, and standard deviation $s\pi/\sqrt{3}$
`rnbinomial(`n`,`p`)`	negative binomial random variates
`rnormal()`	standard normal (Gaussian) random variates, that is, variates from a normal distribution with a mean of 0 and a standard deviation of 1
`rnormal(`m`)`	normal(m,1) (Gaussian) random variates, where m is the mean and the standard deviation is 1
`rnormal(`m`,`s`)`	normal(m,s) (Gaussian) random variates, where m is the mean and s is the standard deviation
`rpoisson(`m`)`	Poisson(m) random variates, where m is the distribution mean
`rt(`*df*`)`	Student's t random variates, where *df* is the degrees of freedom
`runiform()`	uniformly distributed random variates over the interval $(0,1)$
`runiform(`a`,`b`)`	uniformly distributed random variates over the interval (a,b)
`runiformint(`a`,`b`)`	uniformly distributed random integer variates on the interval $[a,b]$

rweibull(a,b)	Weibull variates with shape a and scale b
rweibull(a,b,g)	Weibull variates with shape a, scale b, and location g
rweibullph(a,b)	Weibull (proportional hazards) variates with shape a and scale b
rweibullph(a,b,g)	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 $(0,1)$

runiform() can be seeded with the set seed command; see [R] **set seed**.

Range: 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: χ^2, 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(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(mindouble)` to `c(maxdouble)`

`rlogistic()`

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

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

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

`rlogistic(`s`)`

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

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

Domain s: 0 to `c(maxdouble)`

Range: `c(mindouble)` to `c(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 $x =$ `invlogistic(`m`,`s`,`u`)`, where u is a random uniform(0,1) variate.

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

Domain s: 0 to `c(maxdouble)`

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

`rnbinomial(`n`,`p`)`

Description: negative binomial random variates

If n is integer valued, `rnbinomial()` returns the number of failures before the nth 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 $2^{53} - 1$

`rnormal()`
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)`

`rnormal(`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)`

`rnormal(`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 $x =$ `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 = `invweibulltail(`a`,`b`,`g`,`u`)`, where u is a random uniform(0,1) variate.

Domain a: 0.01 to 1e+6
Domain b: 1e–323 to 8e+307
Domain g: −8e+307 to 8e+307
Range: g + `c(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 = `invweibullphtail(`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

`rweibullph(`a`,`b`,`g`)`

Description: Weibull (proportional hazards) variates with shape a, scale b, and location g

The variates x are generated by x = `invweibullphtail(`a`,`b`,`g`,`u`)`, where u is a random uniform(0,1) variate.

Domain a: 0.01 to 1e+6
Domain b: 1e–323 to 8e+307
Domain g: −8e+307 to 8e+307
Range: g + `c(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) 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). 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

$$x_n = 69069\, x_{n-1} + 1234567 \quad \bmod 2^{32} \tag{1}$$

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

$$z_n = 65184\bigl(z_{n-1} \bmod 2^{16}\bigr) + \mathrm{int}\bigl(z_{n-1}/2^{16}\bigr) \tag{3}$$

$$w_n = 63663\bigl(w_{n-1} \bmod 2^{16}\bigr) + \mathrm{int}\bigl(w_{n-1}/2^{16}\bigr) \tag{4}$$

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), int(x) is the integer part of x.

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

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

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

$$\begin{aligned} x_0 &= 123456789 \\ y_0 &= 521288629 \\ z_0 &= 362436069 \\ w_0 &= 2262615 \end{aligned}$$

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. https://doi.org/10.1007/BF02293108.

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. https://doi.org/10.2307/2346828.

———. 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. https://doi.org/10.2307/2344350.

Best, D. J. 1983. A note on gamma variate generators with shape parameters less than unity. *Computing* 30: 185–188. https://doi.org/10.1007/BF02280789.

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 F_2-linear random number generators. *INFORMS Journal on Computing* 20: 385–390. https://doi.org/10.1287/ijoc.1070.0251.

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. https://doi.org/10.1080/00949658508810839.

——. 1988. Binomial random variate generation. *Communications of the Association for Computing Machinery* 31: 216–222. https://doi.org/10.1145/42372.42381.

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. https://doi.org/10.1080/00949659008811246.

Kemp, C. D. 1986. A modal method for generating binomial variates. *Communications in Statistics—Theory and Methods* 15: 805–813. https://doi.org/10.1080/03610928608829152.

Kemp, C. D., and A. W. Kemp. 1991. Poisson random variate generation. *Applied Statistics* 40: 143–158. https://doi.org/10.2307/2347913.

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. https://doi.org/10.1145/355744.355750.

——. 1980. New methods for generating Student's t and gamma variables. *Computing* 25: 369–377. https://doi.org/10.1007/BF02285231.

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. https://doi.org/10.1145/363872.363883.

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. https://doi.org/10.1145/272991.272995.

Michael, J. R., W. R. Schucany, and R. W. Haas. 1976. Generating random variates using transformations with multiple roots. *American Statistician* 30: 88–90. https://doi.org/10.2307/2683801.

Schmeiser, B. W., and A. J. G. Babu. 1980. Beta variate generation via exponential majorizing functions. *Operations Research* 28: 917–926. https://doi.org/10.1287/opre.28.4.917.

Schmeiser, B. W., and R. Lal. 1980. Squeeze methods for generating gamma variates. *Journal of the American Statistical Association* 75: 679–682. https://doi.org/10.2307/2287668.

Walker, A. J. 1977. An efficient method for generating discrete random variables with general distributions. *ACM Transactions on Mathematical Software* 3: 253–256. https://doi.org/10.1145/355744.355749.

Wichura, M. J. 1988. Algorithm AS241: The percentage points of the normal distribution. *Applied Statistics* 37: 477–484. https://doi.org/10.2307/2347330.

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

Title

Selecting time-span functions

Contents Functions Also see

Contents

Functions

`tin(`d_1`,`d_2`)`

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*

`twithin(`d_1`,`d_2`)`

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

See `tin()` above; `twithin()` 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, $1 \Rightarrow$ *true*

Also see

[FN] **Functions by category**

[D] **egen** — Extensions to generate

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

[U] **13.3 Functions**

Title

Statistical functions

Contents Functions References Also see

Contents

`F(`df_1`,`df_2`,`f`)` — 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 $f < 0$

`Fden(`df_1`,`df_2`,`f`)` — the probability density function of the F distribution with df_1 numerator and df_2 denominator degrees of freedom; `0` if $f < 0$

`Ftail(`df_1`,`df_2`,`f`)` — the reverse cumulative (upper tail or survivor) F distribution with df_1 numerator and df_2 denominator degrees of freedom; `1` if $f < 0$

`gammaden(`a`,`b`,`g`,`x`)` — the probability density function of the gamma distribution; `0` if $x < g$

`gammap(`a`,`x`)` — the cumulative gamma distribution with shape parameter a; `0` if $x < 0$

`gammaptail(`a`,`x`)` — the reverse cumulative (upper tail or survivor) gamma distribution with shape parameter a; `1` if $x < 0$

`hypergeometric(`N`,`K`,`n`,`k`)` — the cumulative probability of the hypergeometric distribution

`hypergeometricp(`N`,`K`,`n`,`k`)` — 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

`ibeta(`a`,`b`,`x`)` — the cumulative beta distribution with shape parameters a and b; `0` if $x < 0$; or `1` if $x > 1$

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

`igaussian(`m`,`a`,`x`)` — the cumulative inverse Gaussian distribution with mean m and shape parameter a; `0` if $x \leq 0$

`igaussianden(`m`,`a`,`x`)` — the probability density of the inverse Gaussian distribution with mean m and shape parameter a; `0` if $x \leq 0$

`igaussiantail(`m`,`a`,`x`)` — the reverse cumulative (upper tail or survivor) inverse Gaussian distribution with mean m and shape parameter a; `1` if $x \leq 0$

`invbinomial(`n`,`k`,`p`)` — the inverse of the cumulative binomial; that is, θ ($\theta =$ probability of success on one trial) such that the probability of observing `floor(`k`)` or fewer successes in `floor(`n`)` trials is p

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

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

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

`invchi2(`df`,`p`)` — the inverse of `chi2()`: if `chi2(`df`,`x`)` $= p$, then `invchi2(`df`,`p`)` $= x$

`invchi2tail(`df`,`p`)` — the inverse of `chi2tail()`: if `chi2tail(`df`,`x`)` $= p$, then `invchi2tail(`df`,`p`)` $= x$

`invdunnettprob(`k`,`df`,`p`)` — the inverse cumulative multiple range distribution that is used in Dunnett's multiple-comparison method with k ranges and df degrees of freedom

`invexponential(`b`,`p`)` — the inverse cumulative exponential distribution with scale b: if `exponential(`b`,`x`)` $= p$, then `invexponential(`b`,`p`)` $= x$

`invexponentialtail(`b`,`p`)` the inverse reverse cumulative exponential distribution with scale b: if `exponentialtail(`b`,`x`)` $= p$, then `invexponentialtail(`b`,`p`)` $= x$

`invF(`df_1`,`df_2`,`p`)` the inverse cumulative F distribution: if `F(`df_1`,`df_2`,`f`)` $= p$, then `invF(`df_1`,`df_2`,`p`)` $= f$

`invFtail(`df_1`,`df_2`,`p`)` 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`)` $= f$

`invgammap(`a`,`p`)` the inverse cumulative gamma distribution: if `gammap(`a`,`x`)` $= p$, then `invgammap(`a`,`p`)` $= x$

`invgammaptail(`a`,`p`)` the inverse reverse cumulative (upper tail or survivor) gamma distribution: if `gammaptail(`a`,`x`)` $= p$, then `invgammaptail(`a`,`p`)` $= x$

`invibeta(`a`,`b`,`p`)` the inverse cumulative beta distribution: if `ibeta(`a`,`b`,`x`)` $= p$, then `invibeta(`a`,`b`,`p`)` $= x$

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

`invigaussian(`m`,`a`,`p`)` the inverse of `igaussian()`: if `igaussian(`m`,`a`,`x`)` $= p$, then `invigaussian(`m`,`a`,`p`)` $= x$

`invigaussiantail(`m`,`a`,`p`)` the inverse of `igaussiantail()`: if `igaussiantail(`m`,`a`,`x`)` $= p$, then `invigaussiantail(`m`,`a`,`p`)` $= x$

`invlaplace(`m`,`b`,`p`)` the inverse of `laplace()`: if `laplace(`m`,`b`,`x`)` $= p$, then `invlaplace(`m`,`b`,`p`)` $= x$

`invlaplacetail(`m`,`b`,`p`)` the inverse of `laplacetail()`: if `laplacetail(`m`,`b`,`x`)` $= p$, then `invlaplacetail(`m`,`b`,`p`)` $= x$

`invlogistic(`p`)` the inverse cumulative logistic distribution: if `logistic(`x`)` $= p$, then `invlogistic(`p`)` $= x$

`invlogistic(`s`,`p`)` the inverse cumulative logistic distribution: if `logistic(`s`,`x`)` $= p$, then `invlogistic(`s`,`p`)` $= x$

`invlogistic(`m`,`s`,`p`)` the inverse cumulative logistic distribution: if `logistic(`m`,`s`,`x`)` $= p$, then `invlogistic(`m`,`s`,`p`)` $= x$

`invlogistictail(`p`)` the inverse reverse cumulative logistic distribution: if `logistictail(`x`)` $= p$, then `invlogistictail(`p`)` $= x$

`invlogistictail(`s`,`p`)` the inverse reverse cumulative logistic distribution: if `logistictail(`s`,`x`)` $= p$, then `invlogistictail(`s`,`p`)` $= x$

`invlogistictail(`m`,`s`,`p`)` the inverse reverse cumulative logistic distribution: if `logistictail(`m`,`s`,`x`)` $= p$, then `invlogistictail(`m`,`s`,`p`)` $= x$

`invnbinomial(`n`,`k`,`q`)` the value of the negative binomial parameter, p, such that $q =$ `nbinomial(`n`,`k`,`p`)`

`invnbinomialtail(`n`,`k`,`q`)` the value of the negative binomial parameter, p, such that $q =$ `nbinomialtail(`n`,`k`,`p`)`

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

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

`invnF(`df_1`,`df_2`,`np`,`p`)` the inverse cumulative noncentral F distribution: if `nF(`df_1`,`df_2`,`np`,`f`)` $= p$, then `invnF(`df_1`,`df_2`,`np`,`p`)` $= f$

`invnFtail(`df_1`,`df_2`,`np`,`p`)` 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`)` $= f$

`invnibeta(`a`,`b`,`np`,`p`)` the inverse cumulative noncentral beta distribution: if `nibeta(`a`,`b`,`np`,`x`)` $= p$, then `invibeta(`a`,`b`,`np`,`p`)` $= x$

`invnormal(`p`)` the inverse cumulative standard normal distribution: if `normal(`z`)` $= p$, then `invnormal(`p`)` $= z$

`invnt(`df`,`np`,`p`)` the inverse cumulative noncentral Student's t distribution: if `nt(`df`,`np`,`t`)` $= p$, then `invnt(`df`,`np`,`p`)` $= t$

`invnttail(`df`,`np`,`p`)` the inverse reverse cumulative (upper tail or survivor) noncentral Student's t distribution: if `nttail(`df`,`np`,`t`)` $= p$, then `invnttail(`df`,`np`,`p`)` $= t$

`invpoisson(`k`,`p`)` the Poisson mean such that the cumulative Poisson distribution evaluated at k is p: if `poisson(`m`,`k`)` $= p$, then `invpoisson(`k`,`p`)` $= m$

`invpoissontail(`k`,`q`)` the Poisson mean such that the reverse cumulative Poisson distribution evaluated at k is q: if `poissontail(`m`,`k`)` $= q$, then `invpoissontail(`k`,`q`)` $= m$

`invt(`df`,`p`)` the inverse cumulative Student's t distribution: if `t(`df`,`t`)` $= p$, then `invt(`df`,`p`)` $= t$

`invttail(`df`,`p`)` the inverse reverse cumulative (upper tail or survivor) Student's t distribution: if `ttail(`df`,`t`)` $= p$, then `invttail(`df`,`p`)` $= t$

`invtukeyprob(`k`,`df`,`p`)` the inverse cumulative Tukey's Studentized range distribution with k ranges and df degrees of freedom

`invweibull(`a`,`b`,`p`)` the inverse cumulative Weibull distribution with shape a and scale b: if `weibull(`a`,`b`,`x`)` $= p$, then `invweibull(`a`,`b`,`p`)` $= x$

`invweibull(`a`,`b`,`g`,`p`)` the inverse cumulative Weibull distribution with shape a, scale b, and location g: if `weibull(`a`,`b`,`g`,`x`)` $= p$, then `invweibull(`a`,`b`,`g`,`p`)` $= x$

`invweibullph(`a`,`b`,`p`)` the inverse cumulative Weibull (proportional hazards) distribution with shape a and scale b: if `weibullph(`a`,`b`,`x`)` $= p$, then `invweibullph(`a`,`b`,`p`)` $= x$

`invweibullph(`a`,`b`,`g`,`p`)` 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`)` $= x$

`invweibullphtail(`a`,`b`,`p`)` 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`)` $= x$

`invweibullphtail(`a`,`b`,`g`,`p`)` 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`)` $= x$

`invweibulltail(`a`,`b`,`p`)` the inverse reverse cumulative Weibull distribution with shape a and scale b: if `weibulltail(`a`,`b`,`x`)` $= p$, then `invweibulltail(`a`,`b`,`p`)` $= x$

`invweibulltail(`a`,`b`,`g`,`p`)` the inverse reverse cumulative Weibull distribution with shape a, scale b, and location g: if `weibulltail(`a`,`b`,`g`,`x`)` $= p$, then `invweibulltail(`a`,`b`,`g`,`p`)` $= x$

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

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

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

`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

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

Domain a: 1e–323 to 8e+307
Domain b: 1e–323 to 8e+307
Domain x: $-$8e+307 to 8e+307; interesting domain is $0 \le x \le 1$
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 `(gamma(`a`)*gamma(`b`)/gamma(`a`+`b`))*ibeta(`a`,`b`,`x`)` or, better when a or b might be large, `exp(lngamma(`a`)+lngamma(`b`)-lngamma(`a`+`b`))*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 $0 \le x \le 1$
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

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

`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 $0 \le x \le 1$
Range: 0 to 1

`invibeta(`a`,`b`,`p`)`

Description: the inverse cumulative beta distribution: if `ibeta(`a`,`b`,`x`)` $= p$, then `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 `ibetatail(`a`,`b`,`x`)` $= p$, then `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.

`nbetaden(`a`,`b`,0,`x`)` = `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 $0 \le x \le 1$

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()`.

`nibeta(`a`,`b`,0,`x`)` = `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 $0 \le x \le 1$

Range: 0 to 1

invnibeta(a,b,np,p)
Description: the inverse cumulative noncentral beta distribution: if nibeta(a,b,np,x) $= p$, then invibeta(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 ($\theta =$ 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 1e+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: $-$1e+300 to 1e+300
Domain b: 1e–100 to 1e+300
Domain x: $-$8e+307 to 8e+307
Range: 0 to 8e+307

`cauchy(`a`,`b`,`x`)`
Description: the cumulative Cauchy distribution with location parameter a and scale parameter b
Domain a: $-$1e+300 to 1e+300
Domain b: 1e–100 to 1e+300
Domain x: $-$8e+307 to 8e+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: $-$1e+300 to 1e+300
Domain b: 1e–100 to 1e+300
Domain x: $-$8e+307 to 8e+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: $-$1e+300 to 1e+300
Domain b: 1e–100 to 1e+300
Domain p: 0 to 1 (exclusive)
Range: $-$8e+307 to 8e+307

`invcauchytail(`a`,`b`,`p`)`
- Description: the inverse of `cauchytail()`: if `cauchytail(`a`,`b`,`x`)` $= p$, then `invcauchytail(`a`,`b`,`p`)` $= x$
- Domain a: $-$1e+300 to 1e+300
- Domain b: 1e–100 to 1e+300
- Domain p: 0 to 1 (exclusive)
- Range: $-$8e+307 to 8e+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: $-$1e+300 to 1e+300
- Domain b: 1e–100 to 1e+300
- Domain x: $-$8e+307 to 8e+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.

χ^2 and noncentral χ^2 distributions

`chi2den(`df`,`x`)`
- Description: the probability density of the χ^2 distribution with df degrees of freedom; `0` if $x < 0$ `chi2den(`df`,`x`)` $=$ `gammaden(`$df/2$`,`2`,`0`,`x`)`
- Domain df: 2e–10 to 2e+17 (may be nonintegral)
- Domain x: $-$8e+307 to 8e+307
- Range: 0 to 8e+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: 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

`chi2tail(`df`,`x`)`

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

`chi2tail(`df`,`x`)` $= 1 -$ `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 `chi2(`df`,`x`)` $= p$, then `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 `chi2tail(`df`,`x`)` $= p$, then `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.

`nchi2den(`df`,0,`x`)` $=$ `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.

`nchi2(`df`,0,`x`)` $=$ `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 `exponential(`b`,`x`)` $= p$, then `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 `exponentialtail(`b`,`x`)` $= p$, then `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

$$\texttt{Fden}(df_1,df_2,f) = \frac{\Gamma(\frac{df_1+df_2}{2})}{\Gamma(\frac{df_1}{2})\Gamma(\frac{df_2}{2})}\left(\frac{df_1}{df_2}\right)^{\frac{df_1}{2}} \cdot f^{\frac{df_1}{2}-1}\left(1+\frac{df_1}{df_2}f\right)^{-\frac{1}{2}(df_1+df_2)}$$

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(`df_1`,`df_2`,`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(`df_1`,`df_2`,`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(`df_1`,`df_2`,`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(`df_1`,`df_2`,`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 10,000

Domain 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 1,000

Domain 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+307

Domain 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+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

`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

$$\texttt{gammaptail(}a\texttt{,}x\texttt{)} = 1 - \texttt{gammap(}a\texttt{,}x\texttt{)} = \int_x^\infty \texttt{gammaden(}a\texttt{,}t\texttt{)}\, dt$$

`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 `gammaptail(`a`,`x`)` $=p$, then `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 $P(a,x) =$ `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 $P(a,x) =$ `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 $P(a,x) =$ `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 $P(a,x) =$ `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 $P(a,x) =$ `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: 1e–300 to 8e+307
Range: $-$8e+307 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

`igaussiantail(`m`,`a`,`x`)`

Description: the reverse cumulative (upper tail or survivor) inverse Gaussian distribution with mean m and shape parameter a; `1` if $x \leq 0$

`igaussiantail(`m`,`a`,`x`)` $= 1 -$ `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

`invigaussian(`m`,`a`,`p`)`
Description: the inverse of `igaussian()`: if `igaussian(`m`,`a`,`x`)` $= p$, then `invigaussian(`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

`invigaussiantail(`m`,`a`,`p`)`
Description: the inverse of `igaussiantail()`: if `igaussiantail(`m`,`a`,`x`)` $= p$, then `invigaussiantail(`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

`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
`laplacetail(`m`,`b`,`x`)` $= 1 -$ `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: $-$8e+307 to 8e+307
Domain b: 1e–307 to 8e+307
Domain p: 0 to 1 (exclusive)
Range: $-$8e+307 to 8e+307

`invlaplacetail(`m`,`b`,`p`)`
Description: the inverse of `laplacetail()`: if `laplacetail(`m`,`b`,`x`)` $= p$, then `invlaplacetail(`m`,`b`,`p`)` $= x$
Domain m: $-$8e+307 to 8e+307
Domain b: 1e–307 to 8e+307
Domain p: 0 to 1 (exclusive)
Range: $-$8e+307 to 8e+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: $-$8e+307 to 8e+307
Domain b: 1e–307 to 8e+307
Domain x: $-$8e+307 to 8e+307
Range: $-$8e+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: $-$8e+307 to 8e+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: 1e–323 to 8e+307
Domain x: $-$8e+307 to 8e+307
Range: 0 to 8e+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 8e+307
Domain s: 1e–323 to 8e+307
Domain x: $-8\text{e}+307$ to 8e+307
Range: 0 to 8e+307

`logistic(`x`)`

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

`logistic(`x`)` = `logistic(1,`x`)` = `logistic(0,1,`x`)`, where x is the value of a logistic random variable.

Domain x: $-8\text{e}+307$ to 8e+307
Range: 0 to 1

`logistic(`s`,`x`)`

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

`logistic(`s`, `x`)` = `logistic(0,`s`,`x`)`, where s is the scale and x is the value of a logistic random variable.

Domain s: 1e–323 to 8e+307
Domain x: $-8\text{e}+307$ to 8e+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 8e+307
Domain s: 1e–323 to 8e+307
Domain x: $-8\text{e}+307$ to 8e+307
Range: 0 to 1

`logistictail(`x`)`

Description: the reverse cumulative logistic distribution with mean 0 and standard deviation $\pi/\sqrt{3}$

`logistictail(`x`)` = `logistictail(1,`x`)` = `logistictail(0,1,`x`)`, where x is the value of a logistic random variable.

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

Range: 0 to 1

`logistictail(`s`,`x`)`

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

`logistictail(`s`,`x`)` = `logistictail(0,`s`,`x`)`, where s is the scale and x is the value of a logistic random variable.

Domain s: 1e–323 to 8e+307

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

Domain s: 1e–323 to 8e+307

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

Range: 0 to 1

`invlogistic(`p`)`

Description: the inverse cumulative logistic distribution: if `logistic(`x`)` = p, then `invlogistic(`p`)` = x

Domain p: 0 to 1

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

`invlogistic(`s`,`p`)`

Description: the inverse cumulative logistic distribution: if `logistic(`s`,`x`)` = p, then `invlogistic(`s`,`p`)` = x

Domain s: 1e–323 to 8e+307

Domain p: 0 to 1

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

`invlogistic(`m`,`s`,`p`)`
Description: the inverse cumulative logistic distribution: if `logistic(`m`,`s`,`x`)` $= p$, then `invlogistic(`m`,`s`,`p`)` $= x$

Domain m: $-8\text{e}+307$ to 8e+307
Domain s: 1e–323 to 8e+307
Domain p: 0 to 1
Range: $-8\text{e}+307$ to 8e+307

`invlogistictail(`p`)`
Description: the inverse reverse cumulative logistic distribution: if `logistictail(`x`)` $= p$, then `invlogistictail(`p`)` $= x$

Domain p: 0 to 1
Range: $-8\text{e}+307$ to 8e+307

`invlogistictail(`s`,`p`)`
Description: the inverse reverse cumulative logistic distribution: if `logistictail(`s`,`x`)` $= p$, then `invlogistictail(`s`,`p`)` $= x$

Domain s: 1e–323 to 8e+307
Domain p: 0 to 1
Range: $-8\text{e}+307$ to 8e+307

`invlogistictail(`m`,`s`,`p`)`
Description: the inverse reverse cumulative logistic distribution: if `logistictail(`m`,`s`,`x`)` $= p$, then `invlogistictail(`m`,`s`,`p`)` $= x$

Domain m: $-8\text{e}+307$ to 8e+307
Domain s: 1e–323 to 8e+307
Domain p: 0 to 1
Range: $-8\text{e}+307$ to 8e+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 nth success when the probability of a success on one trial is p.

Domain n: 1e–10 to 1e+6 (can be nonintegral)
Domain k: 0 to 1e+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 nth success, when the probability of a success on one trial is p.

The negative binomial distribution function is evaluated using `ibeta()`.
Domain n: 1e–10 to 1e+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 nth success, when the probability of a success on one trial is p.

The reverse negative binomial distribution function is evaluated using `ibetatail()`.
Domain n: 1e–10 to 1e+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 $q =$ `nbinomial(`n`,`k`,`p`)`

`invnbinomial()` is evaluated using `invibeta()`.
Domain n: 1e–10 to 1e+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
$q =$ `nbinomialtail(`n`,`k`,`p`)`

`invnbinomialtail()` is evaluated using `invibetatail()`.
Domain n: 1e–10 to 1e+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: $-$8e+307 to 8e+307
Range: 0 to 0.39894 ...

`normalden(`x`,`σ`)`

Description: the normal density with mean 0 and standard deviation σ

`normalden(`x`,1)` $=$ `normalden(`x`)` and
`normalden(`x`,`σ`)` $=$ `normalden(`x/σ`)`$/\sigma$.

Domain x: $-8\text{e+}307$ to 8e+307
Domain σ: 1e–308 to 8e+307
Range: 0 to 8e+307

`normalden(`x`,`μ`,`σ`)`

Description: the normal density with mean μ and standard deviation σ, $N(\mu, \sigma^2)$

`normalden(`x`,0,`s`)` $=$ `normalden(`x`,`s`)` and
`normalden(`x`,`μ`,`σ`)` $=$ `normalden(`$(x-\mu)/\sigma$`)`$/\sigma$. In general,

$$\texttt{normalden(}z\texttt{,}\mu\texttt{,}\sigma\texttt{)} = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left\{\frac{(z-\mu)}{\sigma}\right\}^2}$$

Domain x: $-8\text{e+}307$ to 8e+307
Domain μ: $-8\text{e+}307$ to 8e+307
Domain σ: 1e–308 to 8e+307
Range: 0 to 8e+307

`normal(`z`)`

Description: the cumulative standard normal distribution

`normal(`z`)` $= \int_{-\infty}^{z} \frac{1}{\sqrt{2\pi}} e^{-x^2/2} dx$

Domain: $-8\text{e+}307$ to 8e+307
Range: 0 to 1

`invnormal(`p`)`

Description: the inverse cumulative standard normal distribution: if `normal(`z`)` $= p$, then `invnormal(`p`)` $= z$

Domain: 1e–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 1e+154
Range: $-5\text{e+}307$ to -0.91893853 $=$ `lnnormalden(0)`

`lnnormalden(`x`,`σ`)`

Description: the natural logarithm of the normal density with mean 0 and standard deviation σ

`lnnormalden(`x`, 1)` $=$ `lnnormalden(`x`)` and
`lnnormalden(`x`,`σ`)` $=$ `lnnormalden(`x/σ`)` $- \ln(\sigma)$.

Domain x: $-8\text{e+}307$ to 8e+307
Domain σ: 1e–323 to 8e+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, $N(\mu, \sigma^2)$

`lnnormalden(`x`,0,`s`)` $=$ `lnnormalden(`x`,`s`)` and `lnnormalden(`x`,`μ`,`σ`)` $=$ `lnnormalden((`$x-\mu$`)/`σ`)` $-\ \ln(\sigma)$. In general,

$$\texttt{lnnormalden(}z\texttt{,}\mu\texttt{,}\sigma\texttt{)} = \ln\left[\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{1}{2}\left\{\frac{(z-\mu)}{\sigma}\right\}^2}\right]$$

Domain x: $-8\text{e}+307$ to 8e+307
Domain μ: $-8\text{e}+307$ to 8e+307
Domain σ: 1e–323 to 8e+307
Range: 1e–323 to 8e+307

`lnnormal(`z`)`

Description: the natural logarithm of the cumulative standard normal distribution

$$\texttt{lnnormal(}z\texttt{)} = \ln\left(\int_{-\infty}^{z}\frac{1}{\sqrt{2\pi}}e^{-x^2/2}dx\right)$$

Domain: $-1\text{e}+99$ to 8e+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 $(-\infty, h\,] \times (-\infty, k\,]$:

$$\Phi(h,k,\rho) = \frac{1}{2\pi\sqrt{1-\rho^2}}\int_{-\infty}^{h}\int_{-\infty}^{k}\exp\left\{-\frac{1}{2(1-\rho^2)}\left(x_1^2 - 2\rho x_1 x_2 + x_2^2\right)\right\}dx_1\,dx_2$$

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

`poissontail(`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}

`invpoissontail(`k`,`q`)`

Description: the Poisson mean such that the reverse cumulative Poisson distribution evaluated at k is q: if `poissontail(`m`,`k`)` $= q$, then `invpoissontail(`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

$$\texttt{tden}(df,t) = \frac{\Gamma\{(df+1)/2\}}{\sqrt{\pi df}\,\Gamma(df/2)} \cdot \left(1+t^2/df\right)^{-(df+1)/2}$$

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 $T > t$

$$\texttt{ttail}(df,t) = \int_t^{\infty} \frac{\Gamma\{(df+1)/2\}}{\sqrt{\pi df}\,\Gamma(df/2)} \cdot \left(1+x^2/df\right)^{-(df+1)/2}\,dx$$

Domain df: 2e–10 to 2e+17 (may be nonintegral)
Domain t: $-$8e+307 to 8e+307
Range: 0 to 1

`invt(`df`,`p`)`

Description: the inverse cumulative Student's t distribution: if `t(`df`,`t`)` $= p$, then `invt(`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 `ttail(`df`,`t`)` $= p$, then `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

`invnt(`df`,`np`,`p`)`

Description: the inverse cumulative noncentral Student's t distribution: if `nt(`df`,`np`,`t`)` $= p$, then `invnt(`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 `nttail(`df`,`np`,`t`)` $= p$, then `invnttail(`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: $-8\text{e}+10$ to 8e+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: 1e–100 to 1e+10 (may be nonintegral)
Domain np: $-1{,}000$ to 1,000
Domain t: $-8\text{e}+307$ to 8e+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

`nt(`df`,0,`t`)` $=$ `t(`df`,`t`)`.
Domain df: 1e–100 to 1e+10 (may be nonintegral)
Domain np: $-1{,}000$ to 1,000
Domain t: $-8\text{e}+307$ to 8e+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: 1e–100 to 1e+10 (may be nonintegral)
Domain np: $-1{,}000$ to 1,000
Domain t: $-8\text{e}+307$ to 8e+307
Range: 0 to 1

`npnt(`df`,`t`,`p`)`

Description: the noncentrality parameter, np, for the noncentral Student's t distribution: if `nt(`df`,`np`,`t`)` $= p$, then `npnt(`df`,`t`,`p`)` $= np$
Domain df: 1e–100 to 1e+8 (may be nonintegral)
Domain t: $-8\text{e}+307$ to 8e+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 $b^2\left(\Gamma\{(a+2)/a\} - [\Gamma\{(a+1)/a\}]^2\right)$ 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

`weibulltail(`a`,`b`,`x`)` $=$ `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 `weibull(`a`,`b`,`x`)` $= p$, then `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 `weibull(`a`,`b`,`g`,`x`)` $= p$, then
`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(epsdouble)` to 8e+307

`invweibulltail(`a`,`b`,`p`)`

Description: the inverse reverse cumulative Weibull distribution with shape a and scale b: if `weibulltail(`a`,`b`,`x`)` $= p$, then
`invweibulltail(`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

`invweibulltail(`a`,`b`,`g`,`p`)`

Description: the inverse reverse cumulative Weibull distribution with shape a, scale b, and location g: if `weibulltail(`a`,`b`,`g`,`x`)` $= p$, then
`invweibulltail(`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(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

`weibullphden(`a`,`b`,`x`)` = `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+307

Domain b: 1e–323 to 8e+307

Domain 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

$$ba(x-g)^{a-1}\exp\left\{-b(x-g)^a\right\}$$

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 (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 8e+307

`weibullph(`a`,`b`,`x`)`

Description: the cumulative Weibull (proportional hazards) distribution with shape a and scale b

`weibullph(`a`,`b`,`x`)` = `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

$$1 - \exp\left\{-b(x-g)^a\right\}$$

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 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

$$b^{-\frac{2}{a}}\left(\Gamma\{(a+2)/a\} - [\Gamma\{(a+1)/a\}]^2\right)$$

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

`weibullphtail(`a`,`b`,`x`)` = `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

$$\exp\left\{-b(x-g)^a\right\}$$

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(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: 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(epsdouble)` to 8e+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 1e+100 (may be nonintegral)

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

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

Range: $-$8e+307 to 8e+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 1e+100 (may be nonintegral)

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

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

Range: $-$8e+307 to 8e+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. https://doi.org/10.2307/2281208.

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. https://doi.org/10.2307/2348014.

Posten, H. O. 1993. An effective algorithm for the noncentral beta distribution function. *American Statistician* 47: 129–131. https://doi.org/10.1080/00031305.1993.10475957.

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. https://doi.org/10.1002/bimj.200810459.

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**

Title

String functions

Contents Functions References Also see

Contents

`abbrev(`s`,`n`)`	name s, abbreviated to a length of n
`char(`n`)`	the character corresponding to ASCII or extended ASCII code n; `""` if n is not in the domain
`collatorlocale(`*loc*`,`*type*`)`	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`
`collatorversion(`*loc*`)`	the version string of a collator based on locale *loc*
`indexnot(`s_1`,`s_2`)`	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
`plural(`n`,`s`)`	the plural of s if $n \neq \pm 1$
`plural(`n`,`s_1`,`s_2`)`	the plural of s_1, as modified by or replaced with s_2, if $n \neq \pm 1$
`real(`s`)`	s converted to numeric or *missing*
`regexm(`s`,`re`)`	performs a match of a regular expression and evaluates to `1` if regular expression re is satisfied by the ASCII string s; otherwise, `0`
`regexr(`s_1`,`re`,`s_2`)`	replaces the first substring within ASCII string s_1 that matches re with ASCII string s_2 and returns the resulting string
`regexs(`n`)`	subexpression n from a previous `regexm()` match, where $0 \leq n < 10$
`soundex(`s`)`	the soundex code for a string, s
`soundex_nara(`s`)`	the U.S. Census soundex code for a string, s
`strcat(`s_1`,`s_2`)`	there is no `strcat()` function; instead the addition operator is used to concatenate strings
`strdup(`s_1`,`n`)`	there is no `strdup()` function; instead the multiplication operator is used to create multiple copies of strings
`string(`n`)`	a synonym for `strofreal(`n`)`
`string(`n`,`s`)`	a synonym for `strofreal(`n`,`s`)`
`stritrim(`s`)`	s with multiple, consecutive internal blanks (ASCII space character `char(32)`) collapsed to one blank
`strlen(`s`)`	the number of characters in ASCII s or length in bytes
`strlower(`s`)`	lowercase ASCII characters in string s
`strltrim(`s`)`	s without leading blanks (ASCII space character `char(32)`)
`strmatch(`s_1`,`s_2`)`	`1` if s_1 matches the pattern s_2; otherwise, `0`
`strofreal(`n`)`	n converted to a string
`strofreal(`n`,`s`)`	n converted to a string using the specified display format
`strpos(`s_1`,`s_2`)`	the position in s_1 at which s_2 is first found, `0` if s_2 does not occur, and `1` if s_2 is empty

`strproper(`s`)`	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
`strreverse(`s`)`	reverses the ASCII string s
`strrpos(`s_1`,`s_2`)`	the position in s_1 at which s_2 is last found, 0 if s_2 does not occur, and 1 if s_2 is empty
`strrtrim(`s`)`	s without trailing blanks (ASCII space character `char(32)`)
`strtoname(`s[`,`p]`)`	s translated into a Stata 13 compatible name
`strtrim(`s`)`	s without leading and trailing blanks (ASCII space character `char(32)`); equivalent to `strltrim(strrtrim(`s`))`
`strupper(`s`)`	uppercase ASCII characters in string s
`subinstr(`s_1`,`s_2`,`s_3`,`n`)`	s_1, where the first n occurrences in s_1 of s_2 have been replaced with s_3
`subinword(`s_1`,`s_2`,`s_3`,`n`)`	s_1, where the first n occurrences in s_1 of s_2 as a word have been replaced with s_3
`substr(`s`,`n_1`,`n_2`)`	the substring of s, starting at n_1, for a length of n_2
`tobytes(`s[`,`n]`)`	escaped decimal or hex digit strings of up to 200 bytes of s
`uchar(`n`)`	the Unicode character corresponding to Unicode code point n or an empty string if n is beyond the Unicode code-point range
`udstrlen(`s`)`	the number of display columns needed to display the Unicode string s in the Stata Results window
`udsubstr(`s`,`n_1`,`n_2`)`	the Unicode substring of s, starting at character n_1, for n_2 display columns
`uisdigit(`s`)`	1 if the first Unicode character in s is a Unicode decimal digit; otherwise, 0
`uisletter(`s`)`	1 if the first Unicode character in s is a Unicode letter; otherwise, 0
`ustrcompare(`s_1`,`s_2[`,`loc]`)`	compares two Unicode strings
`ustrcompareex(`s_1`,`s_2`,`loc`,`st`,`$case$`,`$cslv$`,`$norm$`,`num`,`alt`,`fr`)`	compares two Unicode strings
`ustrfix(`s[`,`rep]`)`	replaces each invalid UTF-8 sequence with a Unicode character
`ustrfrom(`s`,`enc`,`$mode$`)`	converts the string s in encoding enc to a UTF-8 encoded Unicode string
`ustrinvalidcnt(`s`)`	the number of invalid UTF-8 sequences in s
`ustrleft(`s`,`n`)`	the first n Unicode characters of the Unicode string s
`ustrlen(`s`)`	the number of characters in the Unicode string s
`ustrlower(`s[`,`loc]`)`	lowercase all characters of Unicode string s under the given locale loc
`ustrltrim(`s`)`	removes the leading Unicode whitespace characters and blanks from the Unicode string s
`ustrnormalize(`s`,`$norm$`)`	normalizes Unicode string s to one of the five normalization forms specified by $norm$
`ustrpos(`s_1`,`s_2[`,`n]`)`	the position in s_1 at which s_2 is first found; otherwise, 0
`ustrregexm(`s`,`re[`,`noc]`)`	performs a match of a regular expression and evaluates to 1 if regular expression re is satisfied by the Unicode string s; otherwise, 0
`ustrregexra(`s_1`,`re`,`s_2[`,`noc]`)`	replaces all substrings within the Unicode string s_1 that match re with s_2 and returns the resulting string

`ustrregexrf(`s_1`,`*re*`,`s_2`[,`*noc*`])`	replaces the first substring within the Unicode string s_1 that matches *re* with s_2 and returns the resulting string
`ustrregexs(`*n*`)`	subexpression *n* from a previous `ustrregexm()` match
`ustrreverse(`*s*`)`	reverses the Unicode string *s*
`ustrright(`*s*`,`*n*`)`	the last *n* Unicode characters of the Unicode string *s*
`ustrrpos(`s_1`,`s_2`[,`*n*`])`	the position in s_1 at which s_2 is last found; otherwise, `0`
`ustrrtrim(`*s*`)`	remove trailing Unicode whitespace characters and blanks from the Unicode string *s*
`ustrsortkey(`*s*`[,`*loc*`])`	generates a null-terminated byte array that can be used by the `sort` command to produce the same order as `ustrcompare()`
`ustrsortkeyex(`*s*`,`*loc*`,`*st*`,`*case*`,`*cslv*`,`*norm*`,`*num*`,`*alt*`,`*fr*`)`	generates a null-terminated byte array that can be used by the `sort` command to produce the same order as `ustrcompare()`
`ustrtitle(`*s*`[,`*loc*`])`	a string with the first characters of Unicode words titlecased and other characters lowercased
`ustrto(`*s*`,`*enc*`,`*mode*`)`	converts the Unicode string *s* in UTF-8 encoding to a string in encoding *enc*
`ustrtohex(`*s*`[,`*n*`])`	escaped hex digit string of *s* up to 200 Unicode characters
`ustrtoname(`*s*`[,`*p*`])`	string *s* translated into a Stata name
`ustrtrim(`*s*`)`	removes leading and trailing Unicode whitespace characters and blanks from the Unicode string *s*
`ustrunescape(`*s*`)`	the Unicode string corresponding to the escaped sequences of *s*
`ustrupper(`*s*`[,`*loc*`])`	uppercase all characters in string *s* under the given locale *loc*
`ustrword(`*s*`,`*n*`[,`*loc*`])`	the *n*th Unicode word in the Unicode string *s*
`ustrwordcount(`*s*`[,`*loc*`])`	the number of nonempty Unicode words in the Unicode string *s*
`usubinstr(`s_1`,`s_2`,`s_3`,`*n*`)`	replaces the first *n* occurrences of the Unicode string s_2 with the Unicode string s_3 in s_1
`usubstr(`*s*`,`n_1`,`n_2`)`	the Unicode substring of *s*, starting at n_1, for a length of n_2
`word(`*s*`,`*n*`)`	the *n*th word in *s*; *missing* (`""`) if *n* is missing
`wordbreaklocale(`*loc*`,`*type*`)`	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*
`wordcount(`*s*`)`	the number of words in *s*

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 $n \neq \pm 1$

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*

`regexm(`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`).

`regexm()` 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 `ustrregexm()`.

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 `ustrregexrf()` or `ustrregexra()`.

Domain s_1: ASCII strings
Domain re: regular expressions
Domain s_2: ASCII strings
Range: ASCII strings

`regexs(`n`)`

Description: subexpression n from a previous `regexm()` 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

`ustrregexm(`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.

`ustrregexm("12345", "([0-9]){5}")` = `1`
`ustrregexm("de TRÈS près", "rès")` = `1`
`ustrregexm("de TRÈS près", "Rès")` = `0`
`ustrregexm("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 `ustrregexm()` 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 $\geq$ 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"
"Здравствуйте " * 2 = "Здравствуйте Здравствуйте "
```

Domain s_1: strings

Domain n: nonnegative integers 0, 1, 2, ...

Range: strings

`string(`n`)`
Description: a synonym for `strofreal(`n`)`

`string(`n`,`s`)`
Description: a synonym for `strofreal(`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: strings
Range: 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 strings
Range: 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 "I" is "i" 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("ὈΔΥΣΣΕΎΣ") = "ὀδυσσεύς"
```

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 `regexm()`, `regexr()`, and `regexs()`.

`strmatch("café", "caf?")` = 1

Domain s_1: strings
Domain s_2: strings
Range: integers 0 or 1

`strofreal(`n`)`

Description: n converted to a string

Also see `real()`.

`strofreal(4)+"F"` = `"4F"`
`strofreal(1234567)` = `"1234567"`
`strofreal(12345678)` = `"1.23e+07"`
`strofreal(.)` = `"."`

Domain n: −8e+307 to 8e+307 or *missing*
Range: strings

`strofreal(`n`,`s`)`

Description: n converted to a string using the specified display format

Also see `real()`.

```
strofreal(4,"%9.2f") = "4.00"
strofreal(123456789,"%11.0g") = "123456789"
strofreal(123456789,"%13.0gc") = "123,456,789"
strofreal(0,"%td") = "01jan1960"
strofreal(225,"%tq") = "2016q2"
strofreal(225,"not a format") = ""
```

Domain n: $-$8e+307 to 8e+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, `0` if s_2 does not occur, and `1` if s_2 is empty

`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
strpos("this","") = 1
```

Domain s_1: strings (to be searched)

Domain s_2: strings (to search for)

Range: integers ≥ 0

`ustrpos(`s_1`,`$s_2$$\left[\right.$`,`$n$$\left.\right]$`)`

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 nth 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, `0` if s_2 does not occur, and `1` if s_2 is empty

`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`
`strrpos("this is","")` = `1`

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 nth 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

`ustrrtrim(`*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.

```
ustrrtrim("this ") = "this"
ustrltrim("this"+char(10)) = "this"
ustrrtrim("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("0médiane") = "_0médiane"
ustrtoname("0médiane", 1) = "_0médiane"
ustrtoname("0médiane", 0) = "0médiane"
```

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 “I” 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

`subinstr(`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

`subinstr()` 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 `usubinstr()`.

If n is *missing*, all occurrences are replaced.

Also see `regexm()`, `regexr()`, and `regexs()`.

`subinstr("this is the day","is","X",1)` = `"thX is the day"`
`subinstr("this is the hour","is","X",2)` = `"thX X the hour"`
`subinstr("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

`usubinstr(`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.

`usubinstr("de très près","ès","es",1)` = `"de tres près"`
`usubinstr("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()`, `regexr()`, 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("Café","café","fr", -1, 1, -1, -1, -1, -1, -1) = -1
ustrcompareex("Café","café","fr", -1, 2, -1, -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("café","Cafe","fr", 1, -1, 1, -1, -1, -1, -1) = -1
ustrcompareex("café","Cafe","fr", 1, 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("coté", "côte","fr_CA",-1,-1,-1,-1,-1,-1,0) = -1
ustrcompareex("coté", "côte","fr_CA",-1,-1,-1,-1,-1,-1,1) = 1
ustrcompareex("coté", "côte","fr_CA",-1,-1,-1,-1,-1,-1,-1) = 1
ustrcompareex("coté", "côte","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")` = `"ï"`

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 `\Uhhhhhhhh` for code points greater than `\uffff`. The function starts at the nth 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("нулю") = "\u043d\u0443\u043b\u044e"
ustrtohex("нулю", 2) = "\u0443\u043b\u044e"
ustrtohex("i"+char(200)+char(0)+"s") =
        "\u0069\ufffd\u0000\u0073"
```

Domain s: Unicode strings
Domain 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 `\Uhhhhhhhh`; 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 `\Uhhhhhhhh` begins with a capital letter "U".

Also see `ustrtohex()`.

```
ustrunescape("\u043d\u0443\u043b\u044e") = "нулю"
```

Domain s: strings of escaped hex values
Range: Unicode strings

`word(s,n)`

Description: the nth 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: strings
Domain n: integers
Range: strings

`ustrword(`s`,`n`[,`loc`])`

Description: the nth 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. lsemantica: 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

Title

Trigonometric functions

Contents Functions Reference Also see

Contents

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

$$\texttt{acosh}(x) = \ln(x + \sqrt{x^2 - 1})$$

Domain: 1 to 8.9e+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

$$\texttt{asinh}(x) = \ln(x + \sqrt{x^2 + 1})$$

Domain: $-8.9\text{e+}307$ to 8.9e+307
Range: -709.77 to 709.77

`atan(`x`)`
Description: the radian value of the arctangent of x
Domain: $-$8e+307 to 8e+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 answer
Domain y: $-$8e+307 to 8e+307
Domain x: $-$8e+307 to 8e+307
Range: $-\pi$ to π

`atanh(`x`)`
Description: the inverse hyperbolic tangent of x
$$\texttt{atanh}(x) = \frac{1}{2}\{\ln(1+x) - \ln(1-x)\}$$
Domain: -1 to 1
Range: $-$8e+307 to 8e+307

`cos(`x`)`
Description: the cosine of x, where x is in radians
Domain: $-$1e+18 to 1e+18
Range: -1 to 1

`cosh(`x`)`
Description: the hyperbolic cosine of x
$$\texttt{cosh}(x) = \{\exp(x) + \exp(-x)\}/2$$
Domain: -709 to 709
Range: 1 to 4.11e+307

`sin(`x`)`
Description: the sine of x, where x is in radians
Domain: $-$1e+18 to 1e+18
Range: -1 to 1

`sinh(`x`)`
Description: the hyperbolic sine of x
$$\texttt{sinh}(x) = \{\exp(x) - \exp(-x)\}/2$$
Domain: -709 to 709
Range: $-$4.11e+307 to 4.11e+307

`tan(`x`)`
Description: the tangent of x, where x is in radians
Domain: $-$1e+18 to 1e+18
Range: $-$1e+17 to 1e+17 or *missing*

`tanh(`x`)`
Description: the hyperbolic tangent of x
$$\texttt{tanh}(x) = \{\exp(x) - \exp(-x)\}/\{\exp(x) + \exp(-x)\}$$
Domain: $-$8e+307 to 8e+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 *Stata Index*.