Title

functions — Functions

Description Acknowledgments References Also see

Description

This entry 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.

Functions are listed under the following headings:

Mathematical functions

`abs(`x`)`
Domain: $-$8e+307 to 8e+307
Range: 0 to 8e+307
Description: returns the absolute value of x.

`acos(`x`)`
Domain: -1 to 1
Range: 0 to π
Description: returns the radian value of the arccosine of x.

`acosh(`x`)`
Domain: 1 to 8.9e+307
Range: 0 to 709.77
Description: returns the inverse hyperbolic cosine of x, `acosh(`x`)` $= \ln(x + \sqrt{x^2 - 1})$.

`asin(`x`)`
Domain: -1 to 1
Range: $-\pi/2$ to $\pi/2$
Description: returns the radian value of the arcsine of x.

`asinh(`x`)`
Domain: $-$8.9e+307 to 8.9e+307
Range: $-$709.77 to 709.77
Description: returns the inverse hyperbolic sine of x, `asinh(`x`)` $= \ln(x + \sqrt{x^2 + 1})$.

`atan(`x`)`
Domain: $-8\text{e}+307$ to 8e+307
Range: $-\pi/2$ to $\pi/2$
Description: returns the radian value of the arctangent of x.

`atan2(`y`, `x`)`
Domain y: $-8\text{e}+307$ to 8e+307
Domain x: $-8\text{e}+307$ to 8e+307
Range: $-\pi$ to π
Description: returns 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`)`
Domain: -1 to 1
Range: $-8\text{e}+307$ to 8e+307
Description: returns the inverse hyperbolic tangent of x, `atanh(`x`)` $= \frac{1}{2}\{\ln(1+x) - \ln(1-x)\}$.

`ceil(`x`)`
Domain: $-8\text{e}+307$ to 8e+307
Range: integers in $-8\text{e}+307$ to 8e+307
Description: returns the unique integer n such that $n-1 < x \leq n$.
returns x (not "`.`") if x is missing, meaning that `ceil(.a)` = `.a`.

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

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

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

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

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

`digamma(`x`)`
Domain: $-1\text{e}+15$ to 8e+307
Range: $-8\text{e}+307$ to 8e+307 and *missing*
Description: returns 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, $\psi(x)$.

`exp(`x`)`
Domain: −8e+307 to 709
Range: 0 to 8e+307
Description: returns the exponential function e^x. This function is the inverse of `ln(`x`)`.

`floor(`x`)`
Domain: −8e+307 to 8e+307
Range: integers in −8e+307 to 8e+307
Description: returns the unique integer n such that $n \leq x < n+1$.
returns x (not "`.`") if x is missing, meaning that `floor(.a)` = `.a`.

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

`int(`x`)`
Domain: −8e+307 to 8e+307
Range: integers in −8e+307 to 8e+307
Description: returns the integer obtained by truncating x toward 0; thus,
`int(5.2)` = 5
`int(-5.8)` = −5
returns x (not "`.`") if x is missing, meaning that `int(.a)` = `.a`.

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

`invcloglog(`x`)`
Domain: −8e+307 to 8e+307
Range: 0 to 1 and *missing*
Description: returns the inverse of the complementary log-log function of x,
`invcloglog(`x`)` $= 1 - \exp\{-\exp(x)\}$.

`invlogit(`x`)`
Domain: −8e+307 to 8e+307
Range: 0 to 1 and *missing*
Description: returns the inverse of the logit function of x,
`invlogit(`x`)` $= \exp(x)/\{1+\exp(x)\}$.

`ln(`x`)`
Domain: 1e–323 to 8e+307
Range: −744 to 709
Description: returns the natural logarithm, $\ln(x)$. This function is the inverse of `exp(`x`)`.

The logarithm of x in base b can be calculated via $\log_b(x) = \log_a(x)/\log_a(b)$. Hence,
$\log_5(x)$ = `ln(`x`)/ln(5)` = `log(`x`)/log(5)` = `log10(`x`)/log10(5)`
$\log_2(x)$ = `ln(`x`)/ln(2)` = `log(`x`)/log(2)` = `log10(`x`)/log10(2)`

You can calculate $\log_b(x)$ by using the formula that best suits your needs.

`lnfactorial(`n`)`
Domain: integers 0 to 1e+305
Range: 0 to 8e+307
Description: returns the natural log of 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.

`lngamma(`x`)`
Domain: −2,147,483,648 to 1e+305 (excluding negative integers)
Range: −8e+307 to 8e+307
Description: returns 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.

`log(`x`)`
Domain: 1e–323 to 8e+307
Range: −744 to 709
Description: returns the natural logarithm, ln(x), which is a synonym for `ln(`x`)`. Also see `ln(`x`)` for more information.

`log10(`x`)`
Domain: 1e–323 to 8e+307
Range: −323 to 308
Description: returns the base-10 logarithm of x.

`logit(`x`)`
Domain: 0 to 1 (exclusive)
Range: −8e+307 to 8e+307 and *missing*
Description: returns the log of the odds ratio of x,
`logit(`x`)` $= \ln\{x/(1-x)\}$.

`max(`x_1`,`x_2`,...,`x_n`)`
Domain x_1: −8e+307 to 8e+307 and *missing*
Domain x_2: −8e+307 to 8e+307 and *missing*
...
Domain x_n: −8e+307 to 8e+307 and *missing*
Range: −8e+307 to 8e+307 and *missing*
Description: returns 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(.,.,.)` = `.`

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

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

`mod(`x`,`y`)`

Domain x: $-$8e+307 to 8e+307
Domain y: 0 to 8e+307
Range: 0 to 8e+307
Description: returns the modulus of x with respect to y.
`mod(`x, y`)` = $x - y$ `floor(`x/y`)`
`mod(`x`,0)` = .

`reldif(`x`,`y`)`

Domain x: $-$8e+307 to 8e+307 and *missing*
Domain y: $-$8e+307 to 8e+307 and *missing*
Range: $-$8e+307 to 8e+307 and *missing*
Description: returns the "relative" difference $|x - y|/(|y| + 1)$.
returns `0` if both arguments are the same type of extended missing value.
returns *missing* if only one argument is missing or if the two arguments are two different types of *missing*.

`round(`x`,`y`)` or `round(`x`)`

Domain x: $-$8e+307 to 8e+307
Domain y: $-$8e+307 to 8e+307
Range: $-$8e+307 to 8e+307
Description: returns x rounded in units of y or x rounded to the nearest integer if the argument y is omitted.
returns x (not "`.`") if x is missing, meaning that `round(.a)` = `.a` and `round(.a,`y`)` = `.a` if y is not missing; 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`)`.

`sign(`x`)`

Domain: $-$8e+307 to 8e+307 and *missing*
Range: -1, 0, 1 and *missing*
Description: returns the sign of x: -1 if $x < 0$, 0 if $x = 0$, 1 if $x > 0$, and *missing* if x is missing.

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

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

`sqrt(`x`)`
Domain: 0 to 8e+307
Range: 0 to 1e+154
Description: returns the square root of x.

`sum(`x`)`
Domain: all real numbers and *missing*
Range: $-8e+307$ to 8e+307 (excluding *missing*)
Description: returns 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.

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

`tanh(`x`)`
Domain: $-8e+307$ to 8e+307
Range: -1 to 1 and *missing*
Description: returns the hyperbolic tangent of x,
`tanh(`x`)` $= \{\exp(x) - \exp(-x)\}/\{\exp(x) + \exp(-x)\}$.

`trigamma(`x`)`
Domain: $-1e+15$ to 8e+307
Range: 0 to 8e+307 and *missing*
Description: returns the second derivative of `lngamma(`x`)` $= d^2 \ln\Gamma(x)/dx^2$. The `trigamma()` function is the derivative of `digammma(`x`)`.

`trunc(`x`)` is a synonym for `int(`x`)`.

❑ 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`)`

❑

Probability distributions and density functions

The probability distributions and density functions are organized under the following headings:

Beta and noncentral beta distributions

`ibeta(`a`,`b`,`x`)`

Domain a: 1e–10 to 1e+17
Domain b: 1e–10 to 1e+17
Domain x: −8e+307 to 8e+307
Interesting domain is $0 \leq x \leq 1$
Range: 0 to 1
Description: returns the cumulative beta distribution with shape parameters a and b defined by

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

returns `0` if $x < 0$.
returns `1` if $x > 1$.

`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.

`betaden(`a`,`b`,`x`)`

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
Description: returns the probability density of the beta distribution,

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

where a and b are the shape parameters.
returns `0` if $x < 0$ or $x > 1$.

`ibetatail(`a`,`b`,`x`)`

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
Description: returns the reverse cumulative (upper tail or survivor) beta distribution with shape parameters a and b defined by

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

returns `1` if $x < 0$.
returns `0` if $x > 1$.

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

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

Domain a: 1e–10 to 1e+17
Domain b: 1e–10 to 1e+17
Domain p: 0 to 1
Range: 0 to 1
Description: returns the inverse cumulative beta distribution: if `ibeta(`a`,`b`,`x`)` $= p$, then `invibeta(`a`,`b`,`p`)` $= x$.

`invibetatail(`a`,`b`,`p`)`

Domain a: 1e–10 to 1e+17
Domain b: 1e–10 to 1e+17
Domain p: 0 to 1
Range: 0 to 1
Description: returns the inverse reverse cumulative (upper tail or survivor) beta distribution: if `ibetatail(`a`,`b`,`x`)` $= p$, then `invibetatail(`a`,`b`,`p`)` $= x$.

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

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
Description: returns the cumulative noncentral beta distribution

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

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()`.

returns `0` if $x < 0$.
returns `1` if $x > 1$.

`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).

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

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
Description: returns the probability density function of the noncentral beta distribution,

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

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

returns `0` if $x < 0$ or $x > 1$.

`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).

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

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
Description: returns the inverse cumulative noncentral beta distribution:

if `nibeta(`a`,`b`,`np`,`x`)` $= p$, then `invibeta(`a`,`b`,`np`,`p`)` $= x$.

Binomial distribution

`binomial(`n`,`k`,`θ`)`

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

`binomialp(`n`,`k`,`p`)`

Domain n: 1 to 1e+6
Domain k: 0 to n
Domain p: 0 to 1
Range: 0 to 1
Description: returns 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`)`

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

`invbinomial(`n`,`k`,`p`)`

Domain n: 1 to 1e+17
Domain k: 0 to $n-1$
Domain p: 0 to 1 (exclusive)
Range: 0 to 1
Description: returns the inverse of the cumulative binomial; that is, it returns 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`)`

Domain n: 1 to 1e+17
Domain k: 1 to n
Domain p: 0 to 1 (exclusive)
Range: 0 to 1
Description: returns the inverse of the right cumulative binomial; that is, it returns 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.

Chi-squared and noncentral chi-squared distributions

`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
- Description: returns the cumulative χ^2 distribution with *df* degrees of freedom.
  `chi2(`*df*`,`x`)` $=$ `gammap(`*df*`/2,`x`/2)`.
  returns `0` if $x < 0$.

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

`chi2tail(`*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
- Description: returns the reverse cumulative (upper tail or survivor) χ^2 distribution with *df* degrees of freedom. `chi2tail(`*df*`,`x`)` $= 1 -$ `chi2(`*df*`,`x`)`.
  returns `1` if $x < 0$.

`invchi2(`*df*`,`p`)`
- Domain *df*: 2e–10 to 2e+17 (may be nonintegral)
- Domain p: 0 to 1
- Range: 0 to 8e+307
- Description: returns the inverse of `chi2()`: if `chi2(`*df*`,`x`)` $= p$, then `invchi2(`*df*`,`p`)` $= x$.

`invchi2tail(`*df*`,`p`)`
- Domain *df*: 2e–10 to 2e+17 (may be nonintegral)
- Domain p: 0 to 1
- Range: 0 to 8e+307
- Description: returns the inverse of `chi2tail()`: if `chi2tail(`*df*`,`x`)` $= p$, then `invchi2tail(`*df*`,`p`)` $= x$.

`nchi2(`df`,`np`,`x`)`

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
Description: returns the cumulative noncentral χ^2 distribution,

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

where df denotes the degrees of freedom, np is the noncentrality parameter, and x is the value of χ^2.
returns `0` if $x < 0$.

`nchi2(`df`,0,`x`)` = `chi2(`df`,`x`)`, but `chi2()` is the preferred function to use for the central χ^2 distribution.

`nchi2den(`df`,`np`,`x`)`

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
Description: returns the probability density of the noncentral χ^2 distribution, where df denotes the degrees of freedom, np is the noncentrality parameter, and x is the value of the χ^2.
returns `0` if $x < 0$.

`nchi2den(`df`,0,`x`)` = `chi2den(`df`,`x`)`, but `chi2den()` is the preferred function to use for the central χ^2 distribution.

`nchi2tail(`df`,`np`,`x`)`

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

`invnchi2(`df`,`np`,`p`)`

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
Description: returns the inverse cumulative noncentral χ^2 distribution:
if `nchi2(`df`,`np`,`x`)` = p, then `invnchi2(`df`,`np`,`p`)` = x;
df must be an integer.

`invnchi2tail(`*df*`,`*np*`,`*p*`)`

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

`npnchi2(`*df*`,`*x*`,`*p*`)`

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

Dunnett's multiple range distribution

`dunnettprob(`*k*`,`*df*`,`*x*`)`

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
Description: returns the cumulative multiple range distribution that is used in Dunnett's multiple-comparison method with k ranges and *df* degrees of freedom.
returns `0` if $x < 0$.

`dunnettprob()` is computed using an algorithm described in Miller (1981).

`invdunnettprob(`*k*`,`*df*`,`*p*`)`

Domain k: 2 to 1e+6
Domain *df*: 2 to 1e+6
Domain p: 0 to 1 (right exclusive)
Range: 0 to 8e+307
Description: returns 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).

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.

F and noncentral F distributions

`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
Description: returns the cumulative F distribution with df_1 numerator and df_2 denominator degrees of freedom: $\texttt{F}(df_1,df_2,f) = \int_0^f \texttt{Fden}(df_1,df_2,t)\,dt$.
returns `0` if $f < 0$.

`Fden(`df_1`,`df_2`,`f`)`

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
Description: returns the probability density function of the F distribution with df_1 numerator and df_2 denominator degrees of freedom:

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

returns `0` if $f < 0$.

`Ftail(`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
Description: returns the reverse cumulative (upper tail or survivor) F distribution with df_1 numerator and df_2 denominator degrees of freedom.
$\texttt{Ftail}(df_1,df_2,f) = 1 - \texttt{F}(df_1,df_2,f)$.
returns `1` if $f < 0$.

`invF(`df_1`,`df_2`,`p`)`
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
Description: returns 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`)`
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
Description: returns the inverse reverse cumulative (upper tail or survivor) F distribution: if `Ftail(`df_1`,`df_2`,`f`)` $= p$, yy then `invFtail(`df_1`,`df_2`,`p`)` $= f$.

`nF(`df_1`,`df_2`,`np`,`f`)`
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
Description: returns the cumulative noncentral F distribution with df_1 numerator and df_2 denominator degrees of freedom and noncentrality parameter np.
`nF(`df_1`,`df_2`,0,`f`)` $=$ `F(`df_1`,`df_2`,`f`)`.
returns 0 if $f < 0$.

`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`,`$df_1 \times f/((df_1 \times f) + df_2)$`)`.

`nFden(`df_1`,`df_2`,`np`,`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 f: $-$8e+307 to 8e+307
Interesting domain is $f \geq 0$
Range: 0 to 8e+307
Description: returns the probability density function of the noncentral F distribution with df_1 numerator and df_2 denominator degrees of freedom and noncentrality parameter np.
returns `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.

`nFtail(`df_1`,`df_2`,`np`,`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 f: $-$8e+307 to 8e+307
Interesting domain is $f \geq 0$
Range: 0 to 1
Description: returns 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.
returns `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.

`invnFtail(`df_1`,`df_2`,`np`,`p`)`
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
Description: returns the inverse reverse cumulative (upper tail or survivor) noncentral F distribution: if `nFtail(`df_1`,`df_2`,`np`,`x`)` $= p$, then `invnFtail(`df_1`,`df_2`,`np`,`p`)` $= x$.

`npnF(`df_1`,`df_2`,`f`,`p`)`

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
Description: returns 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`)` $= np$.

Gamma distribution

`gammap(`a`,`x`)`

Domain a: 1e–10 to 1e+17
Domain x: $-$8e+307 to 8e+307
Interesting domain is $x \geq 0$
Range: 0 to 1
Description: returns the cumulative gamma distribution with shape parameter a defined by

$$\frac{1}{\Gamma(a)} \int_0^x e^{-t} t^{a-1}\, dt$$

returns `0` if $x < 0$.

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`,`$(x-g)/b$`)`.

`gammaden(`a`,`b`,`g`,`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 8e+307
Description: returns the probability density function of the gamma distribution 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.

returns `0` if $x < g$.

`gammaptail(`a`,`x`)`

Domain a: 1e–10 to 1e+17

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

Interesting domain is $x \geq 0$

Range: 0 to 1

Description: returns the reverse cumulative (upper tail or survivor) gamma distribution with shape parameter a defined by

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

returns `1` if $x < 0$.

`gammaptail()` is also known as the complement to the incomplete gamma function (ratio).

`invgammap(`a`,`p`)`

Domain a: 1e–10 to 1e+17

Domain p: 0 to 1

Range: 0 to 8e+307

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

`invgammaptail(`a`,`p`)`

Domain a: 1e–10 to 1e+17

Domain p: 0 to 1

Range: 0 to 8e+307

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

`dgammapda(`a`,`x`)`

Domain a: 1e–7 to 1e+17

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

Interesting domain is $x \geq 0$

Range: $-$16 to 0

Description: returns $\frac{\partial P(a,x)}{\partial a}$, where $P(a,x) =$ `gammap(`a`,`x`)`.
returns `0` if $x < 0$.

`dgammapdada(`a`,`x`)`

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

Description: returns $\frac{\partial^2 P(a,x)}{\partial a^2}$, where $P(a,x) =$ `gammap(`a`,`x`)`.
returns `0` if $x < 0$.

`dgammapdadx(`a`,`x`)`

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
Description: returns $\frac{\partial^2 P(a,x)}{\partial a \partial x}$, where $P(a,x) =$ `gammap(`a`,`x`)`.
returns `0` if $x < 0$.

`dgammapdx(`a`,`x`)`

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
Description: returns $\frac{\partial P(a,x)}{\partial x}$, where $P(a,x) =$ `gammap(`a`,`x`)`.
returns `0` if $x < 0$.

`dgammapdxdx(`a`,`x`)`

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
Description: returns $\frac{\partial^2 P(a,x)}{\partial x^2}$, where $P(a,x) =$ `gammap(`a`,`x`)`.
returns `0` if $x < 0$.

Hypergeometric distribution

`hypergeometric(`N`,`K`,`n`,`k`)`

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
Description: returns 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.

`hypergeometricp(`N`,`K`,`n`,`k`)`

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)
Description: returns the hypergeometric probability of k successes (where success is obtaining an element with the attribute of interest) out of a sample of size n, from a population of size N containing K elements that have the attribute of interest.

Negative binomial distribution

`nbinomial(`n`,`k`,`p`)`

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
Description: returns 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 the `ibeta()` function.

`nbinomialp(`n`,`k`,`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
Description: returns 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.

`nbinomialtail(`n`,`k`,`p`)`

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
Description: returns 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 the `ibetatail()` function.

`invnbinomial(`n`,`k`,`q`)`

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
Description: returns the value of the negative binomial parameter, p, such that $q =$ `nbinomial(`n`,`k`,`p`)`.

`invnbinomial()` is evaluated using `invibeta()`.

`invnbinomialtail(`n`,`k`,`q`)`

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)
Description: returns the value of the negative binomial parameter, p, such that $q =$ `nbinomialtail(`n`,`k`,`p`)`.

`invnbinomialtail()` is evaluated using `invibetatail()`.

Normal (Gaussian), log of the normal, and binormal distributions

`binormal(`h`,`k`,`ρ`)`

Domain h: $-$8e+307 to 8e+307
Domain k: $-$8e+307 to 8e+307
Domain ρ: -1 to 1
Range: 0 to 1
Description: returns the joint cumulative distribution $\Phi(h,k,\rho)$ of bivariate normal with correlation ρ; cumulative over $(-\infty,h] \times (-\infty,k]$:

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

`normal(`z`)`

Domain: $-$8e+307 to 8e+307
Range: 0 to 1
Description: returns the cumulative standard normal distribution.
`normal(`z`)` $= \int_{-\infty}^{z} \frac{1}{\sqrt{2\pi}} e^{-x^2/2} dx$

`normalden(`z`)`

Domain: $-$8e+307 to 8e+307
Range: 0 to 0.39894 ...
Description: returns the standard normal density, $N(0,1)$.

`normalden(`x`,`σ`)`

Domain x: $-$8e+307 to 8e+307
Domain σ: 1e–308 to 8e+307
Range: 0 to 8e+307
Description: returns the normal density with mean 0 and standard deviation σ:
`normalden(`x`,1)` $=$ `normalden(`x`)` and
`normalden(`x`,`σ`)` $=$ `normalden(`x/σ`)`$/\sigma$.

`normalden(`x`,`μ`,`σ`)`
Domain x: $-$8e+307 to 8e+307
Domain μ: $-$8e+307 to 8e+307
Domain σ: 1e–308 to 8e+307
Range: 0 to 8e+307
Description: returns 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((`$x-\mu$`)/`σ`)/`σ. In general,

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

`invnormal(`p`)`
Domain: 1e–323 to $1-2^{-53}$
Range: $-$38.449394 to 8.2095362
Description: returns the inverse cumulative standard normal distribution:
if `normal(`z`)` $= p$, then `invnormal(`p`)` $= z$.

`lnnormal(`z`)`
Domain: $-$1e+99 to 8e+307
Range: $-$5e+197 to 0
Description: returns the natural logarithm of the cumulative standard normal distribution:

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

`lnnormalden(`z`)`
Domain: $-$1e+154 to 1e+154
Range: $-$5e+307 to $-$0.91893853 = `lnnormalden(0)`
Description: returns the natural logarithm of the standard normal density, $N(0,1)$.

`lnnormalden(`x`,`σ`)`
Domain x: $-$8e+307 to 8e+307
Domain σ: 1e–323 to 8e+307
Range: $-$5e+307 to 742.82799
Description: returns 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(σ).

`lnnormalden(`x`,`μ`,`σ`)`
Domain x: $-$8e+307 to 8e+307
Domain μ: $-$8e+307 to 8e+307
Domain σ: 1e–323 to 8e+307
Range: 1e–323 to 8e+307
Description: returns 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(σ). In general,

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

Poisson distribution

poisson(m,k)
Domain m: 1e–10 to $2^{53} - 1$
Domain k: 0 to $2^{53} - 1$
Range: 0 to 1
Description: returns the probability of observing floor(k) or fewer outcomes that are distributed as Poisson with mean m.

The Poisson distribution function is evaluated using the gammaptail() function.

poissonp(m,k)
Domain m: 1e–10 to 1e+8
Domain k: 0 to 1e+9
Range: 0 to 1
Description: returns the probability of observing floor(k) outcomes that are distributed as Poisson with mean m.

The Poisson probability function is evaluated using the gammaden() function.

poissontail(m,k)
Domain m: 1e–10 to $2^{53} - 1$
Domain k: 0 to $2^{53} - 1$
Range: 0 to 1
Description: returns 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 the gammap() function.

invpoisson(k,p)
Domain k: 0 to $2^{53} - 1$
Domain p: 0 to 1 (exclusive)
Range: 1.110e–16 to 2^{53}
Description: returns 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 the invgammaptail() function.

invpoissontail(k,q)
Domain k: 0 to $2^{53} - 1$
Domain q: 0 to 1 (exclusive)
Range: 0 to 2^{53} (left exclusive)
Description: returns 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 the invgammap() function.

Student's t and noncentral Student's t distributions

t(df,t)

Domain df: 2e+10 to 2e+17 (may be nonintegral)
Domain t: −8e+307 to 8e+307
Range: 0 to 1
Description: returns the cumulative Student's t distribution with df degrees of freedom.

tden(df,t)

Domain df: 1e–323 to 8e+307(may be nonintegral)
Domain t: −8e+307 to 8e+307
Range: 0 to 0.39894 . . .
Description: returns the probability density function of Student's t distribution:

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

ttail(df,t)

Domain df: 2e–10 to 2e+17 (may be nonintegral)
Domain t: −8e+307 to 8e+307
Range: 0 to 1
Description: returns the reverse cumulative (upper tail or survivor) Student's t distribution; it returns the probability $T > t$:

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

invt(df,p)

Domain df: 2e–10 to 2e+17 (may be nonintegral)
Domain p: 0 to 1
Range: −8e+307 to 8e+307
Description: returns the inverse cumulative Student's t distribution: if t(df,t) $= p$, then invt(df,p) $= t$.

invttail(df,p)

Domain df: 2e–10 to 2e+17 (may be nonintegral)
Domain p: 0 to 1
Range: −8e+307 to 8e+307
Description: returns the inverse reverse cumulative (upper tail or survivor) Student's t distribution: if ttail(df,t) $= p$, then invttail(df,p) $= t$.

nt(df,np,t)

Domain df: 1e–100 to 1e+10 (may be nonintegral)
Domain np: −1,000 to 1,000
Domain t: −8e+307 to 8e+307
Range: 0 to 1
Description: returns the cumulative noncentral Student's t distribution with df degrees of freedom and noncentrality parameter np. nt(df,0,t) $=$ t(df,t).

`ntden(`*df*`,`*np*`,`*t*`)`
Domain *df*: 1e–100 to 1e+10 (may be nonintegral)
Domain *np*: −1,000 to 1,000
Domain *t*: −8e+307 to 8e+307
Range: 0 to 0.39894 ...
Description: returns the probability density function of the noncentral Student's t distribution with *df* degrees of freedom and noncentrality parameter *np*.

`nttail(`*df*`,`*np*`,`*t*`)`
Domain *df*: 1e–100 to 1e+10 (may be nonintegral)
Domain *np*: −1,000 to 1,000
Domain *t*: −8e+307 to 8e+307
Range: 0 to 1
Description: returns the reverse cumulative (upper tail or survivor) noncentral Student's t distribution with *df* degrees of freedom and noncentrality parameter *np*.

`invnttail(`*df*`,`*np*`,`*p*`)`
Domain *df*: 1 to 1e+6 (may be nonintegral)
Domain *np*: −1,000 to 1,000
Domain *p*: 0 to 1
Range: −8e+10 to 8e+10
Description: returns the inverse reverse cumulative (upper tail or survivor) noncentral Student's t distribution: if `nttail(`*df*`,`*np*`,`*t*`)` $= p$, then `invnttail(`*df*`,`*np*`,`*p*`)` $= t$.

`npnt(`*df*`,`*t*`,`*p*`)`
Domain *df*: 1e–100 to 1e+8 (may be nonintegral)
Domain *t*: −8e+307 to 8e+307
Domain *p*: 0 to 1
Range: −1,000 to 1,000
Description: returns the noncentrality parameter, *np*, for the noncentral Student's t distribution: if `nt(`*df*`,`*np*`,`*t*`)` $= p$, then `npnt(`*df*`,`*t*`,`*p*`)` $= np$.

Tukey's Studentized range distribution

`tukeyprob(`*k*`,`*df*`,`*x*`)`
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
Description: returns the 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.
returns `0` if $x < 0$.

`tukeyprob()` is computed using an algorithm described in Miller (1981).

`invtukeyprob(`k`,`df`,`p`)`
- Domain k: 2 to 1e+6
- Domain df: 2 to 1e+6
- Domain p: 0 to 1
- Range: 0 to 8e+307
- Description: returns 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).

Random-number functions

`runiform()`
- Range: 0 to nearly 1 (0 to $1 - 2^{-32}$)
- Description: returns uniform random variates.

  `runiform()` returns uniformly distributed random variates on the interval $[0, 1)$. `runiform()` takes no arguments, but the parentheses must be typed. `runiform()` can be seeded with the `set seed` command; see the technical note at the end of this subsection. (See *Matrix functions* for the related `matuniform()` matrix function.)

  To generate random variates over the interval $[a, b)$, use a`+(`b`-`a`)*runiform()`.

  To generate random integers over $[a, b]$, use a`+int((`b`-`a`+1)*runiform())`.

`rbeta(`a`,`b`)`
- Domain a: 0.05 to 1e+5
- Domain b: 0.15 to 1e+5
- Range: 0 to 1 (exclusive)
- Description: returns beta(a,b) random variates, where a and b are the beta distribution shape parameters.

  Besides 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).

`rbinomial(`n`,`p`)`

Domain n: 1 to 1e+11
Domain p: 1e–8 to 1−1e–8
Range: 0 to n
Description: returns binomial(n,p) random variates, where n is the number of trials and p is the success probability.

Besides 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).

`rchi2(`df`)`

Domain df: 2e–4 to 2e+8
Range: 0 to `c(maxdouble)`
Description: returns chi-squared, with df degrees of freedom, random variates.

`rgamma(`a`,`b`)`

Domain a: 1e–4 to 1e+8
Domain b: `c(smallestdouble)` to `c(maxdouble)`
Range: 0 to `c(maxdouble)`
Description: returns 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).

`rhypergeometric(`N`,`K`,`n`)`

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`)`
Description: returns 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 the standard methodology for generating random variates from a given distribution, `rhypergeometric()` uses the specialized algorithms of Kachitvichyanukul (1982) and Kachitvichyanukul and Schmeiser (1985).

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

Domain n: 1e–4 to 1e+5
Domain p: 1e–4 to 1−1e–4
Range: 0 to $2^{53}-1$
Description: returns 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.

`rnormal()`

Range: `c(mindouble)` to `c(maxdouble)`
Description: returns 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*`)`

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

`rnormal(`*m*`,`*s*`)`

Domain m: `c(mindouble)` to `c(maxdouble)`
Domain s: 0 to `c(maxdouble)`
Range: `c(mindouble)` to `c(maxdouble)`
Description: returns 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).

`rpoisson(`*m*`)`

Domain m: 1e–6 to 1e+11
Range: 0 to $2^{53} - 1$
Description: returns 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), as well as the method of Kachitvichyanukul (1982).

`rt(`*df*`)`

Domain df: 1 to $2^{53} - 1$
Range: `c(mindouble)` to `c(maxdouble)`
Description: returns 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).

❑ Technical note

The uniform pseudorandom-number function, `runiform()`, is based on George Marsaglia's (G. Marsaglia, 1994, pers. comm.) 32-bit pseudorandom-number generator KISS (keep it simple stupid). The KISS generator is composed of two 32-bit pseudorandom-number generators and two 16-bit generators (combined to make one 32-bit generator). The four generators are defined by the recursions

$$x_n = 69069\, x_{n-1} + 1234567 \quad \bmod 2^{32} \tag{1}$$
$$y_n = y_{n-1}(I + L^{13})(I + R^{17})(I + L^5) \tag{2}$$
$$z_n = 65184\big(z_{n-1} \bmod 2^{16}\big) + \mathrm{int}\big(z_{n-1}/2^{16}\big) \tag{3}$$
$$w_n = 63663\big(w_{n-1} \bmod 2^{16}\big) + \mathrm{int}\big(w_{n-1}/2^{16}\big) \tag{4}$$

In recursion (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 recursions (3) and (4), int(x) is the integer part of x.

The KISS generator produces the 32-bit random number

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

`runiform()` takes the output from the KISS generator and divides it by 2^{32} to produce a real number on the interval $[0, 1)$.

All the nonuniform random-number generators rely on uniform random numbers that are also generated using this KISS algorithm.

The recursions (1)–(4) have, respectively, the periods

$$2^{32} \tag{1}$$
$$2^{32} - 1 \tag{2}$$
$$(65184 \cdot 2^{16} - 2)/2 \approx 2^{31} \tag{3}$$
$$(63663 \cdot 2^{16} - 2)/2 \approx 2^{31} \tag{4}$$

Thus the overall period for the KISS generator 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 KISS by using the seeds

$$x_0 = 123456789 \tag{1}$$
$$y_0 = 521288629 \tag{2}$$
$$z_0 = 362436069 \tag{3}$$
$$w_0 = 2262615 \tag{4}$$

Successive calls to `runiform()` then produce the sequence

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

Hence, `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 `runiform()` give the sequence

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

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 `runiform()` produces by default; also see [R] **set seed**.

❑

❑ Technical note

You may "capture" the current seed (x, y, z, w) by coding

```
. local curseed = "‘c(seed)’"
```

and, later in your code, reestablish that seed by coding

```
. set seed ‘curseed’
```

When the seed is set this way, the first 100 random numbers are not discarded.

`c(seed)` contains a 30-plus long character string similar to

X075bcd151f123bb5159a55e50022865746ad

The string contains an encoding of the four numbers (x, y, z, w) along with checksums and redundancy to ensure that, at `set seed` time, it is valid.

❑

String functions

Stata includes the following *string functions*. In the display below, s indicates a string subexpression (a string literal, a string variable, or another string expression), n indicates a numeric subexpression (a number, a numeric variable, or another numeric expression), and re indicates a regular expression based on Henry Spencer's NFA algorithms and this is nearly identical to the POSIX.2 standard.

`abbrev(`s`,`n`)`
Domain s: strings
Domain n: 5 to 32
Range: strings
Description: returns name s, abbreviated to n 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`.

`char(`n`)`
Domain: integers 0 to 255
Range: ASCII characters
Description: returns the character corresponding to ASCII code n.
returns `""` if n is not in the domain.

`indexnot(`s_1`,`s_2`)`
Domain s_1: strings (to be searched)
Domain s_2: strings of individual characters (to search for)
Range: integers ≥ 0
Description: returns the position in s_1 of the first character of s_1 not found in s_2, or `0` if all characters of s_1 are found in s_2.

`itrim(`s`)`
Domain: strings
Range: strings with no multiple, consecutive internal blanks
Description: returns s with multiple, consecutive internal blanks collapsed to one blank.

`itrim("hello     there")` = `"hello there"`

`length(`s`)`
Domain: strings
Range: integers ≥ 0
Description: returns the length of s. `length("ab")` = 2

`lower(`s`)`
Domain: strings
Range: strings with lowercased characters
Description: returns the lowercased variant of s. `lower("THIS")` = `"this"`

`ltrim(`s`)`
Domain: strings
Range: strings without leading blanks
Description: returns s without leading blanks. `ltrim(" this")` = `"this"`

`plural(`n`,`s`)` or `plural(`n`,`s_1`,`s_2`)`
Domain n: real numbers
Domain s: strings
Domain s_1: strings
Domain s_2: strings
Range: strings
Description: returns the plural of s, or s_1 in the 3-argument case, if $n \neq \pm 1$.
The plural is formed by adding "s" to s if you called `plural(`n`,`s`)`. If you called `plural(`n`,`s_1`,`s_2`)` and 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.
returns s, or s_1 in the 3-argument case, if $n = \pm 1$.

```
plural(1, "horse") = "horse"
plural(2, "horse") = "horses"
plural(2, "glass", "+es") = "glasses"
plural(1, "mouse", "mice") = "mouse"
plural(2, "mouse", "mice") = "mice"
plural(2, "abcdefg", "-efg") = "abcd"
```

`proper(`s`)`
Domain: strings
Range: strings
Description: returns a string with the first letter capitalized, and capitalizes any other letters immediately following characters that are not letters; all other letters converted to lowercase.
`proper("mR. joHn a. sMitH")` = `"Mr. John A. Smith"`
`proper("jack o'reilly")` = `"Jack O'Reilly"`
`proper("2-cent's worth")` = `"2-Cent'S Worth"`

`real(`s`)`
Domain: strings
Range: −8e+307 to 8e+307 and *missing*
Description: returns s converted to numeric, or returns *missing*.
`real("5.2")+1` = `6.2`
`real("hello")` = `.`

`regexm(`s`,`re`)`
Domain s: strings
Domain re: regular expression
Range: strings
Description: performs a match of a regular expression and evaluates to `1` if regular expression re is satisfied by the string s, otherwise returns `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`).

`regexr(`s_1`,`re`,`s_2`)`
Domain s_1: strings
Domain re: regular expression
Domain s_2: strings
Range: strings
Description: replaces the first substring within s_1 that matches re with 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`).

`regexs(`n`)`
Domain: 0 to 9
Range: strings
Description: returns 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 long.

`reverse(`s`)`
Domain: strings
Range: reversed strings
Description: returns s reversed. `reverse("hello")` = `"olleh"`

`rtrim(`s`)`
Domain: strings
Range: strings without trailing blanks
Description: returns s without trailing blanks. `rtrim("this ")` = `"this"`

`soundex(`s`)`
Domain: strings
Range: strings
Description: returns the soundex code for a string, s. The soundex code consists of a letter followed by three numbers: the letter is the first letter of the name and the numbers encode the remaining consonants. Similar sounding consonants are encoded by the same number.

```
soundex("Ashcraft") = "A226"
soundex("Robert") = "R163"
soundex("Rupert") = "R163"
```

`soundex_nara(`s`)`
Domain: strings
Range: strings
Description: returns 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 letter of the name and the numbers encode the remaining consonants. Similar sounding consonants are encoded by the same number.

```
soundex_nara("Ashcraft") = "A261"
```

`strcat(`s_1`,`s_2`)`
Domain s_1: strings
Domain s_2: strings
Range: strings
Description: There is no `strcat()` function. Instead the addition operator is used to concatenate strings:

```
"hello " + "world" = "hello world"
"a" + "b" = "ab"
```

`strdup(`s_1`,`n`)`
Domain s_1: strings
Domain n: nonnegative integers 0, 1, 2, . . .
Range: strings
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"
```

`string(`n`)`
Domain: $-8\text{e}+307$ to 8e+307 and *missing*
Range: strings
Description: returns n converted to a string.

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

`string(`n`,`s`)`
Domain n: $-8\text{e}+307$ to 8e+307 and *missing*
Domain s: strings containing %*fmt* numeric display format
Range: strings
Description: returns n converted to a string.

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

`strlen(`s`)` is a synonym for `length(`s`)`.

`strlower(`x`)` is a synonym for `lower(`x`)`.

`strltrim(`x`)` is a synonym for `ltrim(`x`)`.

`strmatch(`s_1`,`s_2`)`
Domain s: strings
Range: 0 or 1
Description: returns `1` if s_1 matches the pattern s_2; otherwise, it returns `0`.
`strmatch("17.4","1??4")` returns `1`. In s_2, `"?"` means that one character goes here, and `"*"` means that zero or more characters go here. Also see `regexm()`, `regexr()`, and `regexs()`.

`strofreal(`n`)` is a synonym for `string(`n`)`.

`strofreal(`n`,`s`)` is a synonym for `string(`n`,`s`)`.

`strpos(`s_1`,`s_2`)`
Domain s_1: strings (to be searched)
Domain s_2: strings (to search for)
Range: integers ≥ 0
Description: returns the position in s_1 at which s_2 is first found; otherwise, it returns `0`.

```
strpos("this","is") = 3
strpos("this","it") = 0
```

`strproper(`x`)` is a synonym for `proper(`x`)`.

`strreverse(`x`)` is a synonym for `reverse(`x`)`.

`strrtrim(`x`)` is a synonym for `rtrim(`x`)`.

strtoname(s,p)
Domain s: strings
Domain p: 0 or 1
Range: strings
Description: returns s translated into a Stata name. 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. The result is truncated to 32 characters.

```
strtoname("name",1) = "name"
strtoname("a name",1) = "a_name"
strtoname("5",1) = "_5"
strtoname("5:30",1) = "_5_30"
strtoname("5",0) = "5"
strtoname("5:30",0) = "5_30"
```

strtoname(s)
Domain s: strings
Range: strings
Description: returns s translated into a Stata name. 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, then the result is prefixed with an underscore. The result is truncated to 32 characters.

```
strtoname("name") = "name"
strtoname("a name") = "a_name"
strtoname("5") = "_5"
strtoname("5:30") = "_5_30"
```

strtrim(x) is a synonym for trim(x).

strupper(x) is a synonym for upper(x).

subinstr(s_1,s_2,s_3,n)
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 and *missing*
Range: strings
Description: returns s_1, where the first n occurrences in s_1 of s_2 have been replaced with s_3. 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"
```

`subinword(`s_1`,`s_2`,`s_3`,`n`)`
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 and *missing*
Range: strings
Description: returns 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. 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"
```

`substr(`s`,`n_1`,`n_2`)`
Domain s: strings
Domain n_1: integers ≥ 1 and ≤ -1
Domain n_2: integers ≥ 1 and ≤ -1
Range: strings
Description: returns the substring of s, starting at column n_1, for a length of n_2. If $n_1 < 0$, n_1 is interpreted as 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) = ""
```

`trim(`s`)`
Domain: strings
Range: strings without leading or trailing blanks
Description: returns s without leading and trailing blanks; equivalent to `ltrim(rtrim(`s`))`. `trim(" this ") = "this"`

`upper(`s`)`
Domain: strings
Range: strings with uppercased characters
Description: returns the uppercased variant of s. `upper("this") = "THIS"`

`word(`s`, `n`)`
Domain s: strings
Domain n: integers $\ldots, -2, -1, 0, 1, 2, \ldots$
Range: strings
Description: returns the nth word in s. 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.) Returns *missing* (`""`) if n is missing.

`wordcount(`s`)`

Domain: strings

Range: nonnegative integers 0, 1, 2, . . .

Description: returns the number of words in s. A word is a set of characters that start and terminate with spaces, start with the beginning of the string, or terminate with the end of the string.

Programming functions

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

Domain x: −8e+307 to 8e+307

Domain n: integers 1 to 8e+307

Domain x_0: −8e+307 to 8e+307

Domain x_1: x_0 to 8e+307

Range: x_0 to x_1

Description: partitions the interval from x_0 to x_1 into n equal-length intervals and returns the upper bound of the interval that contains x. This function is an automated version of `recode()` (see below).
See [U] 25 Working with categorical data and factor variables for an example.

The algorithm for `autocode()` is

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

`byteorder()`

Range: 1 and 2

Description: returns `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 customary binary files can use `byteorder()` to determine the native byte ordering; see [P] **file**.

`c(`*name*`)`
Domain: names
Range: real values, strings, and *missing*
Description: returns 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.
returns a scalar if the result is scalar.
returns a string of the result containing the first 2,045 characters.

`_caller()`
Range: 1 to 13
Description: returns `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 13, so 13 is the upper end of this range. If Stata 13.1 were the current version, 13.1 would be the upper end of this range, and likewise, if Stata 14 were the current version, 14 would be the upper end of this range. This is a function for use by programmers.

`chop(`x`,` ϵ`)`
Domain x: −8e+307 to 8e+307
Domain ϵ: −8e+307 to 8e+307
Range: −8e+307 to 8e+307
Description: returns `round(`x`)` if `abs(`$x -$ `round(`x`))` $< \epsilon$; otherwise, returns x.
returns x if x is missing.

`clip(`x`,`a`,`b`)`
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
Description: returns x if $a < x < b$, b if $x \geq b$, a if $x \leq a$, and *missing* if x is missing or if $a > b$. If a or b is missing, this is interpreted as $a = -\infty$ or $b = +\infty$, respectively.
returns x if x is missing.

`cond(`x`,`a`,`b`,`c`)` or `cond(`x`,`a`,`b`)`

Domain x: $-$8e+307 to 8e+307 and *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
Description: returns a if x is *true* and nonmissing, b if x is *false*, and c if x is *missing*. returns 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.

`e(`*name*`)`

Domain: names
Range: strings, scalars, matrices, and *missing*
Description: returns 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
`e(`*name*`)` = a string containing the first 2,045 characters if the stored result is a string

`e(sample)`

Range: 0 and 1
Description: returns `1` if the observation is in the estimation sample and `0` otherwise.

`epsdouble()`

Range: a double-precision number close to 0
Description: returns 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.

`epsfloat()`
Range: a floating-point number close to 0
Description: returns 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.

`fileexists(`*f*`)`
Domain: filenames
Range: 0 and 1
Description: returns `1` if the file specified by f exists; returns `0` otherwise.

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`.

`fileread(`*f*`)`
Domain: filenames
Range: strings
Description: returns 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.

`filereaderror(`*f*`)`
Domain: strings
Range: integers
Description: returns `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 `fileexist()`. 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)
```

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

Domain f: filenames
Domain s: strings
Domain r: integers 1 or 2
Range: integers
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.

`float(`x`)`

Domain: −1e+38 to 1e+38
Range: −1e+38 to 1e+38
Description: returns 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] 13.11 Precision and problems therein** for more information.)

`fmtwidth(`*fmtstr*`)`

Range: strings
Description: returns the output length of the %*fmt* contained in *fmtstr*.
returns *missing* if *fmtstr* does not contain a valid %*fmt*. For example, `fmtwidth("%9.2f")` returns `9` and `fmtwidth("%tc")` returns `18`.

`has_eprop(`*name*`)`

Domain: names
Range: 0 or 1
Description: returns `1` if *name* appears as a word in `e(properties)`; otherwise, returns `0`.

`inlist(`z`,`a`,`b`,...)`
Domain: all reals or all strings
Range: 0 or 1
Description: returns `1` if z is a member of the remaining arguments; otherwise, returns `0`. All arguments must be reals or all must be strings. The number of arguments is between 2 and 255 for reals and between 2 and 10 for strings.

`inrange(`z`,`a`,`b`)`
Domain: all reals or all strings
Range: 0 or 1
Description: returns `1` if it is known that $a \leq z \leq b$; otherwise, returns `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 `""`.

`irecode(`x`,`x_1`,`x_2`,`x_3`,...,`x_n`)`
Domain x: −8e+307 to 8e+307
Domain x_i: −8e+307 to 8e+307
Range: nonnegative integers
Description: returns *missing* if x is missing or $x_1, \ldots, x_n$ is not weakly increasing.
returns `0` if $x \leq x_1$.
returns `1` if $x_1 < x \leq x_2$.
returns `2` if $x_2 < x \leq x_3$.
...
returns 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`

`matrix(`*exp*`)`
Domain: any valid expression
Range: evaluation of *exp*
Description: restricts name interpretation to scalars and matrices; see `scalar()` function below.

`maxbyte()`
Range: one integer number
Description: returns the largest value that can be stored in storage type `byte`. This function takes no arguments, but the parentheses must be included.

`maxdouble()`
Range: one double-precision number
Description: returns the largest value that can be stored in storage type `double`. This function takes no arguments, but the parentheses must be included.

`maxfloat()`
Range: one floating-point number
Description: returns the largest value that can be stored in storage type `float`. This function takes no arguments, but the parentheses must be included.

`maxint()`
Range: one integer number
Description: returns the largest value that can be stored in storage type `int`. This function takes no arguments, but the parentheses must be included.

`maxlong()`
Range: one integer number
Description: returns the largest value that can be stored in storage type `long`. This function takes no arguments, but the parentheses must be included.

`mi(`x_1`,`x_2`,...,`x_n`)` is a synonym for `missing(`x_1`,`x_2`,...,`x_n`)`.

`minbyte()`
Range: one integer number
Description: returns the smallest value that can be stored in storage type `byte`. This function takes no arguments, but the parentheses must be included.

`mindouble()`
Range: one double-precision number
Description: returns the smallest value that can be stored in storage type `double`. This function takes no arguments, but the parentheses must be included.

`minfloat()`
Range: one floating-point number
Description: returns the smallest value that can be stored in storage type `float`. This function takes no arguments, but the parentheses must be included.

`minint()`
Range: one integer number
Description: returns the smallest value that can be stored in storage type `int`. This function takes no arguments, but the parentheses must be included.

`minlong()`
Range: one integer number
Description: returns the smallest value that can be stored in storage type `long`. This function takes no arguments, but the parentheses must be included.

`missing(`x_1`,`x_2`,...,`x_n`)`
Domain x_i: any string or numeric expression
Range: 0 and 1
Description: returns `1` if any x_i evaluates to *missing*; otherwise, returns `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`==""`.

`r(`*name*`)`

Domain: names

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

Description: returns 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
`r(`*name*`)` = a string containing the first 2,045 characters if the stored result is a string

`recode(`x`,`x_1`,`x_2`,`...`,`x_n`)`

Domain x: −8e+307 to 8e+307 and *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 and *missing*

Description: returns *missing* if $x_1, \dots, x_n$ is not weakly increasing.
returns x if x is missing.
returns 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 $x_i = +\infty$.

Also see `autocode()` and `irecode()` for other styles of recode functions.

`replay()`

Range: integers 0 and 1, meaning *false* and *true*, respectively

Description: returns `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**.

`return(`*name*`)`

Domain: names

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

Description: returns 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
`return(`*name*`)` = a string containing the first 2,045 characters if the stored result is a string

`s(`*name*`)`

Domain: names

Range: strings and *missing*

Description: returns 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
`s(`*name*`)` = a string containing the first 2,045 characters if the stored result is a string

`scalar(`*exp*`)`
Domain: any valid expression
Range: evaluation of *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.

`smallestdouble()`
Range: a double-precision number close to 0
Description: returns 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.

Date and time functions

Stata's *date and time functions* are described with examples in **[U] 24 Working with dates and times** and **[D] datetime**. 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
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 (`%t`) variable is recorded as the milliseconds, days, weeks, etc., depending upon the units from 01jan1960; negative values indicate dates and times before 01jan1960. Allowable dates and times are those between 01jan0100 and 31dec9999, inclusive, but all functions are based on the Gregorian calendar, and values do not correspond to historical dates before Friday, 15oct1582.

`bofd("`*cal*`",`e_d`)`
Domain *cal*: business calendar names and formats
Domain e_d: `%td` as defined by business calendar named *cal*
Range: as defined by business calendar named *cal*
Description: returns the e_b business date corresponding to e_d.

`Cdhms(`e_d`,`h`,`m`,`s`)`
Domain e_d: `%td` dates 01jan0100 to 31dec9999 (integers −679,350 to 2,936,549)
Domain h: integers 0 to 23
Domain m: integers 0 to 59
Domain s: reals 0.000 to 60.999
Range: datetimes 01jan0100 00:00:00.000 to 31dec9999 23:59:59.999 (integers −58,695,840,000,000 to >253,717,919,999,999) and *missing*
Description: returns 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`)`
Domain h: integers 0 to 23
Domain m: integers 0 to 59
Domain s: reals 0.000 to 60.999
Range: datetimes 01jan0100 00:00:00.000 to 31dec9999 23:59:59.999 (integers −58,695,840,000,000 to >253,717,919,999,999) and *missing*
Description: returns 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]`)`
Domain s_1: strings
Domain s_2: strings
Domain Y: integers 1000 to 9998 (but probably 2001 to 2099)
Range: datetimes 01jan0100 00:00:00.000 to 31dec9999 23:59:59.999 (integers −58,695,840,000,000 to >253,717,919,999,999) and *missing*
Description: returns the e_{tC} datetime (ms. with leap seconds since 01jan1960 00:00:00.000) corresponding to s_1 based on s_2 and Y.

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

`clock(`s_1`,`s_2[`,`Y]`)`
- Domain s_1: strings
- Domain s_2: strings
- Domain Y: integers 1000 to 9998 (but probably 2001 to 2099)
- Range: datetimes 01jan0100 00:00:00.000 to 31dec9999 23:59:59.999 (integers −58,695,840,000,000 to 253,717,919,999,999) and *missing*
- Description: returns 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.

`Cmdyhms(`M`,`D`,`Y`,`h`,`m`,`s`)`
- Domain M: integers 1 to 12
- Domain D: integers 1 to 31
- Domain Y: integers 0100 to 9999 (but probably 1800 to 2100)
- Domain h: integers 0 to 23
- Domain m: integers 0 to 59
- Domain s: reals 0.000 to 60.999
- Range: datetimes 01jan0100 00:00:00.000 to 31dec9999 23:59:59.999 (integers −58,695,840,000,000 to >253,717,919,999,999) and *missing*
- Description: returns 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}`)`
- Domain e_{tc}: datetimes 01jan0100 00:00:00.000 to 31dec9999 23:59:59.999 (integers −58,695,840,000,000 to 253,717,919,999,999)
- Range: datetimes 01jan0100 00:00:00.000 to 31dec9999 23:59:59.999 (integers −58,695,840,000,000 to >253,717,919,999,999)
- Description: returns 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}`)`
- Domain e_{tC}: datetimes 01jan0100 00:00:00.000 to 31dec9999 23:59:59.999 (integers −58,695,840,000,000 to >253,717,919,999,999)
- Range: datetimes 01jan0100 00:00:00.000 to 31dec9999 23:59:59.999 (integers −58,695,840,000,000 to 253,717,919,999,999)
- Description: returns 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`)`

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

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

Description: returns 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`)`

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

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

Description: returns 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]`)` is a synonym for `date(`s_1`,`s_2[`,`Y]`)`.

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

Domain s_1: strings

Domain s_2: strings

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

Range: `%td` dates 01jan0100 to 31dec9999 (integers −679,350 to 2,936,549) and *missing*

Description: returns 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 translation** for more information.

`day(`e_d`)`

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

Range: integers 1 to 31 and *missing*

Description: returns the numeric day of the month corresponding to e_d.

`dhms(`e_d`,`h`,`m`,`s`)`
Domain e_d: `%td` dates 01jan0100 to 31dec9999 (integers −679,350 to 2,936,549)
Domain h: integers 0 to 23
Domain m: integers 0 to 59
Domain s: reals 0.000 to 59.999
Range: datetimes 01jan0100 00:00:00.000 to 31dec9999 23:59:59.999 (integers −58,695,840,000,000 to 253,717,919,999,999) and *missing*
Description: returns the e_{tc} datetime (ms. since 01jan1960 00:00:00.000) corresponding to e_d, h, m, and s.

`dofb(`e_b`,"`*cal*`")`
Domain e_b: `%tb` as defined by business calendar named *cal*
Domain *cal*: business calendar names and formats
Range: as defined by business calendar named *cal*
Description: returns the e_d datetime corresponding to e_b.

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

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

`dofh(`e_h`)`
Domain e_h: `%th` dates 0100h1 to 9999h2 (integers −3,720 to 16,079)
Range: `%td` dates 01jan0100 to 01jul9999 (integers −679,350 to 2,936,366)
Description: returns the e_d date (days since 01jan1960) of the start of half-year e_h.

`dofm(`e_m`)`
Domain e_m: `%tm` dates 0100m1 to 9999m12 (integers −22,320 to 96,479)
Range: `%td` dates 01jan0100 to 01dec9999 (integers −679,350 to 2,936,519)
Description: returns the e_d date (days since 01jan1960) of the start of month e_m.

`dofq(`e_q`)`
Domain e_q: `%tq` dates 0100q1 to 9999q4 (integers −7,440 to 32,159)
Range: `%td` dates 01jan0100 to 01oct9999 (integers −679,350 to 2,936,458)
Description: returns the e_d date (days since 01jan1960) of the start of quarter e_q.

`dofw(`e_w`)`
Domain e_w: `%tw` dates 0100w1 to 9999w52 (integers −96,720 to 418,079)
Range: `%td` dates 01jan0100 to 24dec9999 (integers −679,350 to 2,936,542)
Description: returns the e_d date (days since 01jan1960) of the start of week e_w.

`dofy(`e_y`)`
Domain e_y: `%ty` dates 0100 to 9999 (integers 0100 to 9999)
Range: `%td` dates 01jan0100 to 01jan9999 (integers −679,350 to 2,936,185)
Description: returns the e_d date (days since 01jan1960) of 01jan in year e_y.

`dow(`e_d`)`
- Domain e_d: `%td` dates 01jan0100 to 31dec9999 (integers −679,350 to 2,936,549)
- Range: integers 0 to 6 and *missing*
- Description: returns the numeric day of the week corresponding to date e_d; 0 = Sunday, 1 = Monday, . . . , 6 = Saturday.

`doy(`e_d`)`
- Domain e_d: `%td` dates 01jan0100 to 31dec9999 (integers −679,350 to 2,936,549)
- Range: integers 1 to 366 and *missing*
- Description: returns the numeric day of the year corresponding to date e_d.

`halfyear(`e_d`)`
- Domain e_d: `%td` dates 01jan0100 to 31dec9999 (integers −679,350 to 2,936,549)
- Range: integers 1, 2, and *missing*
- Description: returns the numeric half of the year corresponding to date e_d.

`halfyearly(`s_1`,`s_2[`,`Y]`)`
- Domain s_1: strings
- Domain s_2: strings `"HY"` and `"YH"`; `Y` may be prefixed with *##*
- Domain Y: integers 1000 to 9998 (but probably 2001 to 2099)
- Range: `%th` dates 0100h1 to 9999h2 (integers −3,720 to 16,079) and *missing*
- Description: returns 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}`)`
- Domain e_{tc}: datetimes 01jan0100 00:00:00.000 to 31dec9999 23:59:59.999 (integers −58,695,840,000,000 to 253,717,919,999,999)
- Range: integers 0 through 23, *missing*
- Description: returns the hour corresponding to datetime e_{tc} (ms. since 01jan1960 00:00:00.000).

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

`hms(`h`,`m`,`s`)`
- 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 and *missing*)
- Description: returns the e_{tc} datetime (ms. since 01jan1960 00:00:00.000) corresponding to h, m, s on 01jan1960.

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

`hours(`ms`)`
- Domain ms: real; milliseconds
- Range: real and *missing*
- Description: returns $ms/3{,}600{,}000$.

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

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

`minutes(`ms`)`
Domain ms: real; milliseconds
Range: real and *missing*
Description: returns $ms/60{,}000$.

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

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

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

`month(`e_d`)`
Domain e_d: `%td` dates 01jan0100 to 31dec9999 (integers −679,350 to 2,936,549)
Range: integers 1 to 12 and *missing*
Description: returns the numeric month corresponding to date e_d.

`monthly(`s_1`,`s_2[`,`Y]`)`
Domain s_1: strings
Domain s_2: strings `"MY"` and `"YM"`; `Y` may be prefixed with *##*
Domain Y: integers 1000 to 9998 (but probably 2001 to 2099)
Range: `%tm` dates 0100m1 to 9999m12 (integers −22,320 to 96,479) and *missing*
Description: returns 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`)`
Domain h: real; hours
Range: real and *missing*; milliseconds
Description: returns $h \times 3{,}600{,}000$.

`msofminutes(`m`)`
Domain m: real; minutes
Range: real and *missing*; milliseconds
Description: returns $m \times 60{,}000$.

`msofseconds(`s`)`
Domain s: real; seconds
Range: real and *missing*; milliseconds
Description: returns $s \times 1{,}000$.

`qofd(`e_d`)`
Domain e_d: `%td` dates 01jan0100 to 31dec9999 (integers −679,350 to 2,936,549)
Range: `%tq` dates 0100q1 to 9999q4 (integers −7,440 to 32,159)
Description: returns the e_q quarterly date (quarters since 1960q1) containing date e_d.

`quarter(`e_d`)`
Domain e_d: `%td` dates 01jan0100 to 31dec9999 (integers −679,350 to 2,936,549)
Range: integers 1 to 4 and *missing*
Description: returns the numeric quarter of the year corresponding to date e_d.

`quarterly(`s_1`,`s_2[`,`Y]`)`
Domain s_1: strings
Domain s_2: strings `"QY"` and `"YQ"`; `Y` may be prefixed with *##*
Domain Y: integers 1000 to 9998 (but probably 2001 to 2099)
Range: `%tq` dates 0100q1 to 9999q4 (integers −7,440 to 32,159) and *missing*
Description: returns 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`)`
Domain ms: real; milliseconds
Range: real and *missing*
Description: returns $ms/1{,}000$.

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

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

`tC(`l`)`

Domain l: datetime literal strings 01jan0100 00:00:00.000 to 31dec9999 23:59:59.999
Range: datetimes 01jan0100 00:00:00.000 to 31dec9999 23:59:59.999
(integers −58,695,840,000,000 to >253,717,919,999,999)
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`.

`tc(`l`)`

Domain l: datetime literal strings 01jan0100 00:00:00.000 to 31dec9999 23:59:59.999
Range: datetimes 01jan0100 00:00:00.000 to 31dec9999 23:59:59.999
(integers −58,695,840,000,000 to 253,717,919,999,999)
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`.

`td(`l`)`

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

`th(`l`)`

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

`tm(`l`)`

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

`tq(`l`)`

Domain l: quarter literal strings 0100q1 to 9999q4
Range: `%tq` dates 0100q1 to 9999q4 (integers −7,440 to 32,159)
Description: convenience function to make typing quarterly dates in expressions easier;
for example, typing `tq(1960q2)` is equivalent to typing `1`.

`tw(`l`)`

Domain l: week literal strings 0100w1 to 9999w52
Range: `%tw` dates 0100w1 to 9999w52 (integers −96,720 to 418,079)
Description: convenience function to make typing weekly dates in expressions easier;
for example, typing `tw(1960w2)` is equivalent to typing `1`.

`week(`e_d`)`

Domain e_d: `%td` dates 01jan0100 to 31dec9999 (integers −679,350 to 2,936,549)
Range: integers 1 to 52 and *missing*
Description: returns 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.

`weekly(`s_1`,`s_2[`,`Y]`)`
: 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: `%tw` dates 0100w1 to 9999w52 (integers −96,720 to 418,079) and *missing*
: Description: returns 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`)`
: Domain e_d: `%td` dates 01jan0100 to 31dec9999 (integers −679,350 to 2,936,549)
: Range: `%tw` dates 0100w1 to 9999w52 (integers −96,720 to 418,079)
: Description: returns the e_w weekly date (weeks since 1960w1) containing date e_d.

`year(`e_d`)`
: Domain e_d: `%td` dates 01jan0100 to 31dec9999 (integers −679,350 to 2,936,549)
: Range: integers 0100 to 9999 (but probably 1800 to 2100)
: Description: returns the numeric year corresponding to date e_d.

`yearly(`s_1`,`s_2[`,`Y]`)`
: 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: `%ty` dates 0100 to 9999 (integers 0100 to 9999) and *missing*
: Description: returns the e_y yearly date (year) corresponding to s_1 based on s_2 and Y; Y specifies *topyear*; see `date()`.

`yh(`Y`,`H`)`
: Domain Y: integers 1000 to 9999 (but probably 1800 to 2100)
: Domain H: integers 1, 2
: Range: `%th` dates 1000h1 to 9999h2 (integers −1,920 to 16,079)
: Description: returns the e_h half-yearly date (half-years since 1960h1) corresponding to year Y, half-year H.

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

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

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

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

Selecting time spans

`tin(`d_1`,`d_2`)`
Domain d_1: date or time literals recorded in units of t previously `tsset`
Domain d_2: date or time literals recorded in units of t previously `tsset`
Range: 0 and 1, 1 $\Rightarrow$ *true*
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)`.

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 `%tdnn/dd/yy` so that 5jan1992 displays as 1/5/92, you would still type the date in day–month–year order: `tin(5jan1992, 14apr2002)`.

`twithin(`d_1`,`d_2`)`
Domain d_1: date or time literals recorded in units of t previously `tsset`
Domain d_2: date or time literals recorded in units of t previously `tsset`
Range: 0 and 1, 1 $\Rightarrow$ *true*
Description: *true* if $d_1 < t < d_2$, where t is the time variable previously `tsset`; see the `tin()` function above; `twithin()` is similar, except the range is exclusive.

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`)`
Domain: $n \times n$, positive-definite, symmetric matrices
Range: $n \times n$ lower-triangular matrices
Description: returns 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.

`corr(`M`)`
Domain: $n \times n$ symmetric variance matrices
Range: $n \times n$ symmetric correlation matrices
Description: returns the correlation matrix of the variance matrix.
Row and column names are obtained from M.

`diag(`v`)`
Domain: $1 \times n$ and $n \times 1$ vectors
Range: $n \times n$ diagonal matrices
Description: returns 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.

`get(`*systemname*`)`
Domain: existing names of system matrices
Range: matrices
Description: returns a copy of Stata internal system matrix *systemname*.

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

`hadamard(`M`,`N`)`
Domain M: $m \times n$ matrices
Domain N: $m \times n$ matrices
Range: $m \times n$ matrices
Description: returns 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`)`
Domain: real scalars 1 to `matsize`
Range: identity matrices
Description: returns an $n \times n$ identity matrix if n is an integer; otherwise, this function returns the `round(`n`)`$\times$`round(`n`)` identity matrix.

`inv(`M`)`

Domain: $n \times n$ nonsingular matrices

Range: $n \times n$ matrices

Description: returns 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.

`invsym(`M`)`

Domain: $n \times n$ symmetric matrices

Range: $n \times n$ symmetric matrices

Description: returns 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.

`J(`r`,`c`,`z`)`

Domain r: integer scalars 1 to `matsize`

Domain c: integer scalars 1 to `matsize`

Domain z: scalars −8e+307 to 8e+307

Range: $r \times c$ matrices

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

`matuniform(`r`,`c`)`

Domain r: integer scalars 1 to `matsize`

Domain c: integer scalars 1 to `matsize`

Range: $r \times c$ matrices

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

`nullmat(`*matname*`)`
Domain: matrix names, existing and nonexisting
Range: matrices including null if *matname* does not exist
Description: `nullmat()` is for use with the row-join (`,`) and column-join (`\`) operators in programming situations. 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, $\mathtt{v} = (1)$ is formed. The second time through, `v` does exist, so $\mathtt{v} = (1, 2)$ is formed, and so on.

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

`sweep(`M`,`i`)`
Domain M: $n \times n$ matrices
Domain i: integer scalars 1 to n
Range: $n \times n$ matrices
Description: returns 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

$$B_{kk} = \frac{1}{A_{kk}}$$

$$B_{ik} = -\frac{A_{ik}}{A_{kk}}, \qquad i \neq k$$

$$B_{kj} = \frac{A_{kj}}{A_{kk}}, \qquad j \neq k$$

$$B_{ij} = A_{ij} - \frac{A_{ik}A_{kj}}{A_{kk}}, \qquad i \neq k, j \neq k$$

`vec(`M`)`
Domain: matrices
Range: column vectors ($n \times 1$ matrices)
Description: returns a column vector formed by listing the elements of M, starting with the first column and proceeding column by column.

`vecdiag(`M`)`
- Domain: $n \times n$ matrices
- Range: $1 \times n$ vectors
- Description: returns 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.

Matrix functions returning a scalar

`colnumb(`M`,`s`)`
- Domain M: matrices
- Domain s: strings
- Range: integer scalars 1 to `matsize` and *missing*
- Description: returns the column number of M associated with column name s. returns *missing* if the column cannot be found.

`colsof(`M`)`
- Domain: matrices
- Range: integer scalars 1 to `matsize`
- Description: returns the number of columns of M.

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

`diag0cnt(`M`)`
- Domain: $n \times n$ (square) matrices
- Range: integer scalars 0 to n
- Description: returns the number of zeros on the diagonal of M.

`el(`s`,`i`,`j`)`
- Domain s: strings containing matrix name
- Domain i: scalars 1 to `matsize`
- Domain j: scalars 1 to `matsize`
- Range: scalars −8e+307 to 8e+307 and *missing*
- Description: returns s`[floor(`i`),floor(`j`)]`, the i, j element of the matrix named s. returns *missing* if i or j are out of range or if matrix s does not exist.

`issymmetric(`M`)`
- Domain M: matrices
- Range: integers 0 and 1
- Description: returns `1` if the matrix is symmetric; otherwise, returns `0`.

`matmissing(`M`)`
- Domain M: matrices
- Range: integers 0 and 1
- Description: returns `1` if any elements of the matrix are missing; otherwise, returns `0`.

`mreldif(`X`,`Y`)`
- Domain X: matrices
- Domain Y: matrices with same number of rows and columns as X
- Range: scalars −8e+307 to 8e+307
- Description: returns the relative difference of X and Y, where the relative difference is defined as $\max_{i,j}\big(|x_{ij} - y_{ij}|/(|y_{ij}| + 1)\big)$.

`rownumb(`M`,`s`)`
Domain M: matrices
Domain s: strings
Range: integer scalars 1 to `matsize` and *missing*
Description: returns the row number of M associated with row name s.
returns *missing* if the row cannot be found.

`rowsof(`M`)`
Domain: matrices
Range: integer scalars 1 to `matsize`
Description: returns the number of rows of M.

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

Acknowledgments

We thank George Marsaglia of Florida State University for providing his KISS (keep it simple stupid) random-number generator.

We thank John R. Gleason of Syracuse University (retired) 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.

Jacques Salomon Hadamard (1865–1963) was born in Versailles, France. He studied at the Ecole Normale Supérieure in Paris and obtained a doctorate in 1892 for a thesis on functions defined by Taylor series. Hadamard taught at Bordeaux for 4 years and in a productive period published an outstanding theorem on prime numbers, proved independently by Charles de la Vallée Poussin, and worked on what are now called Hadamard matrices. In 1897, he returned to Paris, where he held a series of prominent posts. In his later career, his interests extended from pure mathematics toward mathematical physics. Hadamard produced papers and books in many different areas. He campaigned actively against anti-Semitism at the time of the Dreyfus affair. After the fall of France in 1940, he spent some time in the United States and then Great Britain.

References

Abramowitz, M., and I. A. Stegun, ed. 1968. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. 7th ed. Washington, DC: National Bureau of Standards.

Ahrens, J. H., and U. Dieter. 1974. Computer methods for sampling from gamma, beta, Poisson, and binomial distributions. *Computing* 12: 223–246.

Atkinson, A. C., and J. C. Whittaker. 1970. Algorithm AS 134: The generation of beta random variables with one parameter greater than and one parameter less than 1. *Applied Statistics* 28: 90–93.

——. 1976. A switching algorithm for the generation of beta random variables with at least one parameter less than 1. *Journal of the Royal Statistical Society, Series A* 139: 462–467.

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

Buis, M. L. 2007. Stata tip 48: Discrete uses for uniform(). *Stata Journal* 7: 434–435.

Cox, N. J. 2003. Stata tip 2: Building with floors and ceilings. *Stata Journal* 3: 446–447.

——. 2004. Stata tip 6: Inserting awkward characters in the plot. *Stata Journal* 4: 95–96.

——. 2006. Stata tip 39: In a list or out? In a range or out? *Stata Journal* 6: 593–595.

——. 2007. Stata tip 43: Remainders, selections, sequences, extractions: Uses of the modulus. *Stata Journal* 7: 143–145.

——. 2011a. Stata tip 98: Counting substrings within strings. *Stata Journal* 11: 318–320.

——. 2011b. Speaking Stata: Fun and fluency with functions. *Stata Journal* 11: 460–471.

Devroye, L. 1986. *Non-uniform Random Variate Generation*. New York: Springer.

Dunnett, C. W. 1955. A multiple comparison for comparing several treatments with a control. *Journal of the American Statistical Association* 50: 1096–1121.

Gentle, J. E. 2003. *Random Number Generation and Monte Carlo Methods*. 2nd ed. New York: Springer.

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/.

Hilbe, J. M. 2010. Creating synthetic discrete-response regression models. *Stata Journal* 10: 104–124.

Hilbe, J. M., and W. Linde-Zwirble. 1995. sg44: Random number generators. *Stata Technical Bulletin* 28: 20–21. Reprinted in *Stata Technical Bulletin Reprints*, vol. 5, pp. 118–121. College Station, TX: Stata Press.

——. 1998. sg44.1: Correction to random number generators. *Stata Technical Bulletin* 41: 23. Reprinted in *Stata Technical Bulletin Reprints*, vol. 7, p. 166. College Station, TX: Stata Press.

Jeanty, P. W. 2013. Dealing with identifier variables in data management and analysis. *Stata Journal* 13: 699–718.

Johnson, N. L., S. Kotz, and N. Balakrishnan. 1995. *Continuous Univariate Distributions, Vol. 2*. 2nd ed. New York: Wiley.

Kachitvichyanukul, V. 1982. Computer Generation of Poisson, Binomial, and Hypergeometric Random Variables. PhD thesis, Purdue University.

Kachitvichyanukul, V., and B. W. Schmeiser. 1985. Computer generation of hypergeometric random variates. *Journal of Statistical Computation and Simulation* 22: 127–145.

——. 1988. Binomial random variate generation. *Communications of the Association for Computing Machinery* 31: 216–222.

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

Kemp, A. W., and C. D. Kemp. 1990. A composition-search algorithm for low-parameter Poisson generation. *Journal of Statistical Computation and Simulation* 35: 239–244.

Kemp, C. D. 1986. A modal method for generating binomial variates. *Communications in Statistics, Theory and Methods* 15: 805–813.

Kemp, C. D., and A. W. Kemp. 1991. Poisson random variate generation. *Applied Statistics* 40: 143–158.

Kinderman, A. J., and J. F. Monahan. 1977. Computer generation of random variables using the ratio of uniform deviates. *ACM Transactions on Mathematical Software* 3: 257–260.

——. 1980. New methods for generating Student's t and gamma variables. *Computing* 25: 369–377.

Knuth, D. E. 1998. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*. 3rd ed. Reading, MA: Addison–Wesley.

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.

Mazýa, V. G., and T. O. Shaposhnikova. 1998. *Jacques Hadamard, A Universal mathematician*. Providence, RI: American Mathematical Society.

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.

Oldham, K. B., J. C. Myland, and J. Spanier. 2009. *An Atlas of Functions*. 2nd ed. New York: Springer.

Posten, H. O. 1993. An effective algorithm for the noncentral beta distribution function. *American Statistician* 47: 129–131.

Press, W. H., S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. 2007. *Numerical Recipes: The Art of Scientific Computing*. 3rd ed. New York: Cambridge University Press.

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

Schmeiser, B. W., and A. J. G. Babu. 1980. Beta variate generation via exponential majorizing functions. *Operations Research* 28: 917–926.

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

Tamhane, A. C. 2008. Eulogy to Charles Dunnett. *Biometrical Journal* 50: 636–637.

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

Weiss, M. 2009. Stata tip 80: Constructing a group variable with specified group sizes. *Stata Journal* 9: 640–642.

Wichura, M. J. 1988. Algorithm AS241: The percentage points of the normal distribution. *Applied Statistics* 37: 477–484.

Also see

[D] **egen** — Extensions to generate

[M-5] **intro** — Mata functions

[U] 13.3 Functions

[U] 14.8 Matrix functions