Random-number functions

Contents

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

Functions

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

For information on setting the random-number seed, see [R] **set seed**.

`runiform()`

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

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

Range: `c(epsdouble)` to $1 -$ `c(epsdouble)`

`runiform(`a`,`b`)`

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

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

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

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

`runiformint(`a`,`b`)`

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

If a or b is nonintegral, `runiformint(`a`,`b`)` returns `runiformint(floor(`a`), floor(`b`))`.

Domain a: -2^{53} to 2^{53} (may be nonintegral)

Domain b: -2^{53} to 2^{53} (may be nonintegral)

Range: -2^{53} to 2^{53}

`rbeta(`a`,`b`)`

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

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

Domain a: 0.05 to 1e+5

Domain b: 0.15 to 1e+5

Range: 0 to 1 (exclusive)

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

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

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

Domain n: 1 to 1e+11

Domain p: 1e–8 to 1−1e–8

Range: 0 to n

`rchi2(`*df*`)`
Description: chi-squared, with *df* degrees of freedom, random variates
Domain *df*: 2e–4 to 2e+8
Range: 0 to `c(maxdouble)`

`rexponential(`*b*`)`
Description: exponential random variates with scale *b*
Domain *b*: 1e–323 to 8e+307
Range: 1e–323 to 8e+307

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

Methods for generating gamma variates are taken from Ahrens and Dieter (1974), Best (1983), and Schmeiser and Lal (1980).
Domain *a*: 1e–4 to 1e+8
Domain *b*: `c(smallestdouble)` to `c(maxdouble)`
Range: 0 to `c(maxdouble)`

`rhypergeometric(`*N*`,`*K*`,`*n*`)`
Description: hypergeometric random variates

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

Besides using the standard methodology for generating random variates from a given distribution, `rhypergeometric()` uses the specialized algorithms of Kachitvichyanukul (1982) and Kachitvichyanukul and Schmeiser (1985).
Domain N: 2 to 1e+6
Domain K: 1 to $N-1$
Domain n: 1 to $N-1$
Range: `max(`$0,n-N+K$`)` to `min(`K,n`)`

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

`rigaussian()` is based on a method proposed by Michael, Schucany, and Haas (1976).
Domain m: 1e–10 to 1000
Domain a: 0.001 to 1e+10
Range: 0 to `c(maxdouble)`

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

The variates x are generated by $x =$ `invlogistic(0,1,`*u*`)`, where u is a random uniform(0,1) variate.
Range: `c(mindouble)` to `c(maxdouble)`

`rlogistic(`s`)`

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

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

Domain s: 0 to `c(maxdouble)`

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

`rlogistic(`m`,`s`)`

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

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

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

Domain s: 0 to `c(maxdouble)`

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

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

Description: negative binomial random variates

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

Domain n: 1e–4 to 1e+5

Domain p: 1e–4 to $1-$1e–4

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

`rnormal()`

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

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

`rnormal(`m`)`

Description: normal(m,1) (Gaussian) random variates, where m is the mean and the standard deviation is 1

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

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

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

Description: normal(m,s) (Gaussian) random variates, where m is the mean and s is the standard deviation

The methods for generating normal (Gaussian) random variates are taken from Knuth (1998, 122–128); Marsaglia, MacLaren, and Bray (1964); and Walker (1977).

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

Domain s: 0 to `c(maxdouble)`

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

`rpoisson(`m`)`

Description: Poisson(m) random variates, where m is the distribution mean

Poisson variates are generated using the probability integral transform methods of Kemp and Kemp (1990, 1991) and the method of Kachitvichyanukul (1982).

Domain m: 1e–6 to 1e+11

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

`rt(`df`)`

Description: Student's t random variates, where df is the degrees of freedom

Student's t variates are generated using the method of Kinderman and Monahan (1977, 1980).

Domain df: 1 to $2^{53}-1$

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

`rweibull(`a`,`b`)`

Description: Weibull variates with shape a and scale b

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

Domain a: 0.01 to 1e+6

Domain b: 1e–323 to 8e+307

Range: 1e–323 to 8e+307

`rweibull(`a`,`b`,`g`)`

Description: Weibull variates with shape a, scale b, and location g

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

Domain a: 0.01 to 1e+6

Domain b: 1e–323 to 8e+307

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

Range: $g +$ `c(epsdouble)` to 8e+307

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

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

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

Domain a: 0.01 to 1e+6

Domain b: 1e–323 to 8e+307

Range: 1e–323 to 8e+307

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

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

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

Domain a: 0.01 to 1e+6

Domain b: 1e–323 to 8e+307

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

Range: $g +$ `c(epsdouble)` to 8e+307

Remarks and examples

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

```
set seed #
```

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

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

`runiform()` implements the Mersenne Twister 64-bit (MT64) and the "keep it simple stupid" 32-bit (KISS32) algorithms for generating uniform $(0, 1)$ random numbers. `runiform()` uses the MT64 algorithm by default.

`runiform()` uses the KISS32 algorithm only when the user version is less than 14 or when the random-number generator has been set to `kiss32`; see [P] **version** for details about setting the user version. We recommend that you do not change the default random-number generator, but see [R] **set rng** for details.

❑ Technical note

Although we recommend that you use `runiform()`, we made generator-specific versions of `runiform()` available for advanced users who want to hardcode their generator choice. The function `runiform_mt64()` always uses the MT64 algorithm to generate uniform $(0, 1)$ random numbers, and the function `runiform_kiss32()` always uses the KISS32 algorithm to generate uniform $(0, 1)$ random numbers. In fact, generator-specific versions are available for all the implemented distributions. For example, `rnormal_mt64()` and `rnormal_kiss32()` use transforms of MT64 and KISS32 uniform variates, respectively, to generate standard normal variates.

❑

❑ Technical note

Both the MT64 and the KISS32 generators produce uniform variates that pass many tests for randomness. Many researchers prefer the MT64 to the KISS32 generator because the MT64 generator has a longer period and a finer resolution and requires a higher dimension before patterns appear; see Matsumoto and Nishimura (1998).

The MT64 generator has a period of $2^{19937} - 1$ and a resolution of 2^{-53}; see Matsumoto and Nishimura (1998). The KISS32 generator has a period of about 2^{126} and a resolution of 2^{-32}; see *Methods and formulas* below.

❑

❑ Technical note

This technical note explains how to restart a random-number generator from its current spot.

The current spot in the sequence of a random-number generator is part of the state of a random-number generator. If you tell me the state of a random-number generator, I know where it is in its sequence, and I can compute the next random number. The state of a random-number generator is a complicated object that requires more space than the integers used to seed a generator. For instance, an MT64 state is a 5008-digit, base-16 number preceded by three letters.

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

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

     +-----------+
     |         x |
     |-----------|
  1. |  .3576297 |
  2. |  .4004426 |
  3. |  .6893833 |
     +-----------+
```

We store the state of the random-number generator so that we can pick up right here in the sequence.

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

We draw some more random numbers.

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

     +-----------+
     |         x |
     |-----------|
  1. |  .5597356 |
  2. |  .5744513 |
  3. |  .2076905 |
     +-----------+
```

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

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

     +-----------+
     |         x |
     |-----------|
  1. |  .5597356 |
  2. |  .5744513 |
  3. |  .2076905 |
     +-----------+
```

❑

Methods and formulas

All the nonuniform generators are based on the uniform MT64 and KISS32 generators.

The MT64 generator is well documented in Matsumoto and Nishimura (1998) and on their website http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html. The MT64 implements the 64-bit version discussed at http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt64.html. The default seed for the MT64 generator is 123456789.

KISS32 generator

The KISS32 uniform generator implemented in `runiform()` is based on George Marsaglia's (G. Marsaglia, 1994, pers. comm.) 32-bit pseudorandom-integer generator KISS32. The integer KISS32 generator is composed of two 32-bit pseudorandom-integer generators and two 16-bit integer generators (combined to make one 32-bit integer generator). The four generators are defined by the recursions

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

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

The integer KISS32 generator produces the 32-bit random integer

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

The KISS32 uniform implemented in `runiform()` takes the output from the integer KISS32 generator and divides it by 2^{32} to produce a real number on the interval $(0, 1)$. (Zeros are discarded, and the first nonzero result is returned.)

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

$$2^{32} \tag{5}$$
$$2^{32} - 1 \tag{6}$$
$$(65184 \cdot 2^{16} - 2)/2 \approx 2^{31} \tag{7}$$
$$(63663 \cdot 2^{16} - 2)/2 \approx 2^{31} \tag{8}$$

Thus the overall period for the integer KISS32 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 KISS32 by using the seeds

$$x_0 = 123456789$$
$$y_0 = 521288629$$
$$z_0 = 362436069$$
$$w_0 = 2262615$$

Successive calls to the KISS32 uniform implemented in `runiform()` then produce the sequence

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

Hence, the KISS32 uniform implemented in `runiform()` gives the same sequence of random numbers in every Stata session (measured from the start of the session) unless you reinitialize the seed. The full seed is the set of four numbers (x, y, z, w), but you can reinitialize the seed by simply issuing the command

```
. set seed #
```

where # is any integer between 0 and $2^{31}-1$, inclusive. When this command is issued, the initial value x_0 is set equal to #, and the other three recursions are restarted at the seeds y_0, z_0, and w_0 given above. The first 100 random numbers are discarded, and successive calls to the KISS32 uniform implemented in `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 the KISS32 generator produces when Stata restarts; also see [R] **set seed**.

Acknowledgments

We thank the late George Marsaglia, formerly of Florida State University, for providing his KISS32 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.

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

References

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

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

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

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

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

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

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

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.

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

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

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

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

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

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

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

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

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

Lee, S. 2015. Generating univariate and multivariate nonnormal data. *Stata Journal* 15: 95–109.

Lukácsy, K. 2011. Generating random samples from user-defined distributions. *Stata Journal* 11: 299–304.

Marsaglia, G., M. D. MacLaren, and T. A. Bray. 1964. A fast procedure for generating normal random variables. *Communications of the Association for Computing Machinery* 7: 4–10.

Matsumoto, M., and T. Nishimura. 1998. Mersenne Twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation* 8: 3–30.

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

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

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

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

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

Also see

[D] **egen** — Extensions to generate

[M-5] **intro** — Alphabetical index to functions

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

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

[R] **set seed** — Specify random-number seed and state

[U] **13.3 Functions**