Description Syntax Remarks and examples References Also see

Description

set rngstream specifies the subsequence, known as a stream, from which Stata's stream randomnumber generator should draw random numbers. When performing a bootstrap estimation or a Monte Carlo simulation in parallel on multiple machines, you should set the same seed on all machines but set a different stream on each machine. This will ensure that random numbers drawn on different machines are independent. We strongly recommend that you set the seed and the stream only once in each Stata session.

Syntax

 $\texttt{set} \texttt{rngstream} \ \#$

is any integer between 1 and 32,767.

Remarks and examples

Stata's stream random-number generator, the stream 64-bit Mersenne Twister (mt64s), allows separate instances of Stata to simultaneously draw independent random numbers. This feature enables you to use bootstrap and to run Monte Carlo simulations in parallel on multiple machines.

What we call random numbers are elements in a sequence of deterministic numbers that appear to be random. A seed specifies a starting value in this sequence. In figure 1, each tick is an element in a random sequence and setting the seed to 12345 means that the tick identified by the arrow below is the first number drawn.



Figure 1. Seed specifies first number in random sequence

A stream random-number generator partitions a sequence of random numbers into nonoverlapping subsequences known as streams. The random numbers in each stream are independent of those in other streams because they come from distinct nonoverlapping subsets of the original sequence.

Figure 2 depicts a stream version of the generator depicted in figure 1. The stream version also starts at the place implied by seed 12345, but it additionally partitions the random numbers into 4 streams and a set of unused numbers.



In contrast to nonstream random-number generators, setting the seed for a random-number generator controls not just where the sequence starts but also how the sequence is partitioned. Compare figure 2 with figure 3 for an illustration.



Figure 3. Changing the seed changes the streams

Seed 123456 specifies the first random number, and the streams of random numbers in figure 3 differ from those in figure 2.

The mt64s generator is a stream version of Stata's default generator, the 64-bit Mersenne Twister implemented in mt64; see Matsumoto and Nishimura (1998) and Random-number generators in Stata in [R] set rng for more details. Our implementation of the method discussed in Haramoto et al. (2008) partitions the mt64 sequence into 32,767 streams, each containing 2^{128} random numbers. The remaining numbers are unused. The mt64s seed determines the starting point of every stream in the Mersenne Twister sequence.

Stream 1 of mt64s has the same starting point as the mt64 generator. That is, given the same seed, mt64s with rngstream set to 1 will generate the same random numbers as mt64.

The mt64s generator is designed to simultaneously draw independent random numbers on different machines. To draw from different streams that guarantee independence, use the same seed and change the stream. For example, to draw some uniform(0,1) random numbers from stream 10 of the mt64s generator under seed 123, type

```
. set rng mt64s
. set rngstream 10
. set seed 123
. generate u = runiform()
```

If we wanted to simultaneously draw some uniform(0,1) random numbers on another machine from stream 11 of the mt64s generator, we would type

```
. set rng mt64s
. set rngstream 11
. set seed 123
. generate u = runiform()
```

Again, each seed creates a different partition of the mt64 sequence into nonoverlapping subsets.

We strongly recommend that you set the stream and the seed once in each Stata session and draw numbers only from this stream.

c(rngstream) returns the current stream number. c(rngseed_mt64s) returns the last seed that was set for mt64s. See [P] creturn for more details. See [R] set seed for details about storing and restoring the current position in the random sequence.

As with the single-stream generators, use local state = c(rngstate) to store the current position in the current random stream; see [R] set seed for details. The mt64s state encodes the seed used in addition to the stream number, because the seed determines the position of every random number in every stream. Unlike the case of single-stream generators, restoring the state also restores the seed. For example, suppose you save an mt64s state with local state = c(rngstate) change the seed and the stream, and later restore that state with set rngstate 'state'. The current mt64s seed is changed to the one encoded in state. In addition to changing the current stream to the one encoded in state, the current mt64s seed is changed to the one encoded in state. This behavior ensures any subsequent stream changes draw from nonoverlapping subsets.

set rngstream also sets the random-number generator to mt64s.

Example 1: Using stream random numbers to parallelize a bootstrap

We illustrate how to simultaneously perform 100 bootstrap replications on machine 1 and 100 bootstrap replications on machine 2. We focus on the mechanics of distributing the draws over machines using stream random numbers; see [R] bootstrap for an introduction to the bootstrap.

On machine 1, we type

```
. clear all
. use https://www.stata-press.com/data/r19/auto
(1978 automobile data)
. set rng mt64s
. set rngstream 1
. set seed 12345
. bootstrap, reps(100) saving(machine1, replace): regress mpg weight gear foreign
(running regress on estimation sample)
(file machine1.dta not found)
Bootstrap replications (100): .....10......20.......30.......40......
Linear regression
                                                    Number of obs =
                                                                       74
                                                    Replications =
                                                                      100
                                                    Wald chi2(3) = 191.66
                                                    Prob > chi2
                                                                 = 0.0000
                                                    R-squared
                                                                 = 0.6670
                                                    Adj R-squared = 0.6527
                                                    Root MSE
                                                                 = 3.4096
                Observed
                          Bootstrap
                                                          Normal-based
             coefficient
                          std. err.
                                             P>|z|
                                                      [95% conf. interval]
                                        z
        mpg
     weight
                -.006139
                          .0005462
                                    -11.24
                                             0.000
                                                     -.0072095
                                                                 -.0050685
                                             0.252
                                                                 3.948817
                1.457113
                          1.271301
                                      1.15
                                                      -1.03459
 gear_ratio
    foreign
               -2.221682
                          1.090115
                                     -2.04
                                             0.042
                                                     -4.358267
                                                                -.0850957
               36.10135
                          4.720623
                                      7.65
                                             0.000
                                                       26.8491
                                                                  45.3536
      cons
```

On machine 2, we type

. clear all						
. use https:// (1978 automobi	/www.stata-pre ile data)	ss.com/data	/r19/auto	D		
. set rng mt64	ls					
. set rngstrea	am 2					
. set seed 123	345					
. bootstrap, n (running regre (file machine)	reps(100) savi ss on estimat 2.dta not foun	ng(machine2 ion sample) d)	, replace	e): regro	ess mpg weight	; gear foreign
Bootstrap repl >50	Lications (100 60): 70	.10 80	20 90		40
Linear regress	sion				Number of ob Replications Wald chi2(3) Prob > chi2 R-squared Adj R-square Root MSE	$\begin{array}{rcl} s &=& 74 \\ s &=& 100 \\ s &=& 121.48 \\ s &=& 0.0000 \\ s &=& 0.6670 \\ s &=& 0.6527 \\ s &=& 3.4096 \end{array}$
mpg	Observed coefficient	Bootstrap std. err.	z	P> z	Normal [95% conf.	-based interval]
weight gear_ratio foreign	006139 1.457113 -2.221682	.0005909 1.267439 1.253503	-10.39 1.15 -1.77	0.000 0.250 0.076	0072972 -1.027022 -4.678502	0049809 3.941249 .2351393
cons	30.10135	4.419/9/	0.1/	0.000	21.430/1	44.704

After copying machine2.dta from machine 2 to the working directory on machine 1, we produce the combined results by typing

. clear all
. use machine1
(bootstrap: regress)
. append using machine2
. bstat
Bootstrap results

Number of obs = 74 Replications = 200

Command: regress mpg weight gear foreign

	Observed coefficient	Bootstrap std. err.	z	P> z	Normal [95% conf.	-based interval]
weight gear_ratio foreign _cons	006139 1.457113 -2.221682 36.10135	.0005678 1.266586 1.187847 4.562644	-10.81 1.15 -1.87 7.91	0.000 0.250 0.061 0.000	0072519 -1.02535 -4.549819 27.15873	0050262 3.939577 .1064562 45.04397

We used regress in this example, but the divide-and-conquer strategy reduces computation time for any command that works with bootstrap. In fact, problems that take longer produce more noticeable speed improvements. For computationally intensive problems, the two-machine time will be about onehalf the one-machine time. Using distinct streams on many different machines can dramatically reduce the time required for computationally intensive problems.

4

Example 2: Using stream random numbers to parallelize a Monte Carlo simulation

We want to simultaneously perform 100 Monte Carlo replications on machine 3 and 100 Monte Carlo replications on machine 4. Again, we focus entirely on the mechanics of distributing the draws over machines. See Drukker (2015) for an introduction to Monte Carlo simulations using Stata.

As discussed in [R] simulate, the simulate command uses an ado-file that draws from the population of interest; it then computes and returns the estimates. Our program chi2sim draws from a χ^2 distribution with one degree of freedom.

```
program define chi2sim, rclass
    version 19.5    // (or version 19 if you do not have StataNow)
    drop _all
    set obs 200
    tempvar z
    generate 'z' = rchi2(1)
    summarize 'z'
    return scalar mean = r(mean)
    return scalar Var = r(Var)
end
```

On machine 3, we type

```
. set rng mt64s
. set rngstream 3
```

. set seed 12345

```
. simulate mean=r(mean) var=r(Var), reps(500) saving(machine3, replace): chi2sim
Command: chi2sim
```

```
mean: r(mean)
var: r(Var)
```

```
(file machine3.dta not found)
```

Si	mulations	(500):		10			30		40		
>	60	70		80			100		110		
>			.140	1	50	160.		.170	1	180	1
>	90	200		210			230		240		
>			.270	2	80	290.		.300	3	310	3
>	20			340	350.		360		370		
>			.400	4	10	420.		.430	4	140	4
>	50	.460		470			490		500 dor	ie	

On machine 4, we run a do-file that performs

```
. set rng mt64s
. set rngstream 4
. set seed 12345
. simulate mean=r(mean) var=r(Var), reps(500) saving(machine4, replace): chi2sim
  Command: chi2sim
    mean: r(mean)
    var: r(Var)
(file machine4.dta not found)
Simulations (500): ......10......20.......30.......40.......50......
> .....130.......140.......150......160.......170.......180......1
> .....390........400........410.......420.......430.......440.......4
```

After copying machine4.dta from machine 4 to the working directory on machine 3, we combine the results by typing

. clear all					
. use machine3 (simulate: chi2	sim)				
. append using	machine4				
. summarize mea	n var				
Variable	Obs	Mean	Std. dev.	Min	Max
mean	1,000	1.00296	.1011507	.7098877	1.330305
var	1,000	2.01955	.5194171	.8931134	4.381884

As in example 1, more machines enable further parallelization.

Technical note

While mt64s has been made robust to switching between streams within a Stata session, convoluted combinations of set rngstream # and set seed # can lead to drawing the same random numbers, just as it can in the case of single-stream generators. We strongly recommend that you do not switch between streams within a session.

4

Example 3: Position within a stream is stored

This example illustrates that the sequence picks up where it left off when the stream is switched, and it illustrates that clear rngstream resets all streams to their beginning positions. These features facilitate advanced programming techniques, and we recommend against using this feature in standard use.

```
. clear all
. set obs 10
Number of observations (_N) was 0, now 10.
. set rng mt64s
. set rngstream 5
. set seed 12345
. generate x = runiform() in 1/5
(5 missing values generated)
. set rngstream 6
. generate y = runiform()
. set rngstream 5
. replace x = runiform() in 6/10
(5 real changes made)
. clear rngstream
. set rngstream 5
. generate z = runiform()
. list
              х
                                      z
                          у
  1.
       .5095264
                   .8838338
                              .5095264
```

2. .9766202 .7677673 .9766202 З. .3933811 .5665985 .3933811 4. .950057 .3141659 .950057 5. .5862163 .6635106 .5862163 6. .4837167 .6781911 .4837167 7. .1752382 .7169843 .1752382 8. .2302023 .7554966 .2302023 9. .4927879 .8685812 .4927879 10. .9114158 .5634732 .9114158

After setting the rngstream to 5 and setting the seed, we put the first 5 draws from stream 5 into observations 1-5 of x, switch to rngstream 6, put the first 10 random draws from stream 6 into y, return to rngstream 5, and put the next 5 draws from stream 5 into observations 6-10 of x. Then we use clear rngstream to initialize each rngstream at its initial position for seed 12345 and put the first 10 draws from stream 5 into z.

That the random numbers in x match those in z illustrates that the sequence picks up where it left off when the stream is switched and the seed has not been changed.

References

- Drukker, D. M. 2015. Monte Carlo simulations using Stata. The Stata Blog: Not Elsewhere Classified. https://blog.stata. com/2015/10/06/monte-carlo-simulations-using-stata/.
- Haramoto, H., M. Matsumoto, T. Nishimura, F. Panneton, and P. L'Ecuyer. 2008. Efficient jump ahead for F₂-linear random number generators. INFORMS Journal on Computing 20: 385–390. https://doi.org/10.1287/ijoc.1070.0251.
- Matsumoto, M., and T. Nishimura. 1998. Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator. ACM Transactions on Modeling and Computer Simulation 8: 3–30. https://doi.org/10.1145/ 272991.272995.
- Taylor, M. A. 2018. Simulating the central limit theorem. Stata Journal 18: 345-356.
- Vega Yon, G. G., and B. Quistorff. 2019. parallel: A command for parallel computing. Stata Journal 19: 667-684.

Also see

- [R] set Overview of system parameters
- [R] set rng Set which random-number generator (RNG) to use
- [R] set seed Specify random-number seed and state
- [D] clear Clear memory
- [FN] Random-number functions

Stata, Stata Press, Mata, NetCourse, and NetCourseNow are registered trademarks of StataCorp LLC. Stata and Stata Press are registered trademarks with the World Intellectual Property Organization of the United Nations. StataNow is a trademark of StataCorp LLC. Other brand and product names are registered trademarks or trademarks of their respective companies. Copyright © 1985–2025 StataCorp LLC, College Station, TX, USA. All rights reserved.



For suggested citations, see the FAQ on citing Stata documentation.