WPS6278

Policy Research Working Paper 6278

Perils of Simulation

Parallel Streams and the Case of Stata's Rnormal Command

Owen Ozier

The World Bank
Development Research Group
Human Development and Public Services Team
November 2012

Policy Research Working Paper 6278

Abstract

Large-scale simulation-based studies rely on at least three properties of pseudorandom number sequences: they behave in many ways like truly random numbers; they can be replicated; and they can be generated in parallel. There has been some divergence, however, between empirical techniques employing random numbers, and the standard battery of tests used to validate them. A random number generator that passes tests for any single stream of random numbers may fail the same tests when it is used to generate multiple streams in parallel. The lack of systematic testing of parallel streams leaves statistical software with important potential vulnerabilities. This paper shows one such vulnerability in Stata's rnormal function that went unnoticed for almost four years, and how to detect it. It then shows practical implications for the use of parallel streams in existing software.

This paper is a product of the Human Development and Public Services Team, Development Research Group. It is part of a larger effort by the World Bank to provide open access to its research and make a contribution to development policy discussions around the world. Policy Research Working Papers are also posted on the Web at http://econ.worldbank.org.
The author may be contacted at oozier@worldbank.org.

Perils of simulation:
Parallel streams and the case of Stata's rnormal command

Owen Ozier*

JEL codes: C87, C15
Keywords: rnormal, uniform, random, simulation, pseudorandom, seed, parallel

*Development Economics Research Group, The World Bank, oozier@worldbank.org. Thanks to Justin McCrary, Bruce McCullough, and William Gould for very valuable comments. All errors are my own. The findings, interpretations and conclusions expressed in this paper are entirely those of the author, and do not necessarily represent the views of the World Bank, its Executive Directors, or the governments of the countries they represent.

1 Introduction

"It is good practice to set a random seed explicitly to allow reproducibility of results." (Smeeton and Cox 2003)

Simulations, whether in the form of bootstrapped confidence intervals, simulated likelihood, or other Monte Carlo methods, have grown increasingly common in the social sciences in recent years. As computing power has become cheap and ubiquitous, these methods have grown more computationally intensive, and parallelization of work across multiple processors or machines is frequently essential. These empirical methods usually involve a stream of random numbers; breaking up a large task of this kind requires either breaking up that stream, or accessing multiple independent streams. Ripley (1990) refers to this as "decimation" of the computational task. Moler (2007) suggests that running parallel simulations with different streams of random numbers is one of the easiest methods of distributed computing. This bug, and its implications, are relevant to real-world research only if researchers ever draw from multiple parallel streams, particularly with Stata, in this case. In recent work in the social sciences, Busso, DiNardo, and McCrary (2011) provide at least one clear case in which parallel simulations on separate processors were conducted exactly this way, and were described in detail.[1]

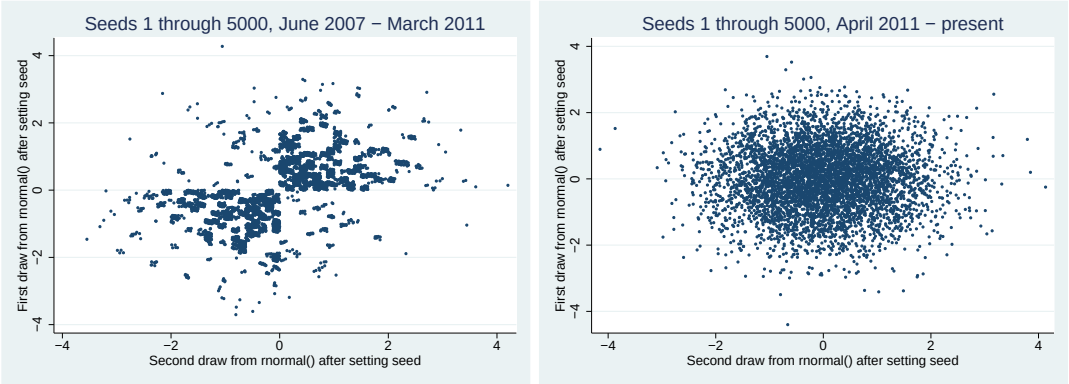
Since the classic works by Knuth (1998) and Marsaglia (1968), standardized tests of "randomness" have been developed to confirm that so-called random number generators (RNGs) have desirable statistical properties (L'Ecuyer and Simard 2007, McCullough 2006). These tests are generally designed to test a single stream of random numbers, but since most RNGs allow a stream to be initialized using a "seed," as mentioned in the Smeeton and Cox (2003) quote above, multiple streams, each started with a different "seed," can also be tested together for joint randomness. Though this facility exists, testing multiple streams of random numbers is not yet commonplace; two recent assessments of several econometric packages makes no mention of this possibility (Odeh, Featherstone, and Bergtold 2010, Keeling and Pavur 2007). This discussion only matters if, as Compagner (1995) writes, the particular simulation of interest and the RNG "interfere constructively." With modern

[1] See Busso, DiNardo, and McCrary (2011), Appendix ID, for further discussion.

RNGs, however, this can still happen quite easily.

The figure below shows two plots for the Stata `rnormal()` command, which generates normally-distributed pseudorandom numbers. In both cases, the seed is reset to a new value after each pair of draws.

Figure 1: Resetting the seed has important effects on `rnormal()`



While no problem is evident in the right panel of Figure 1, resetting the seed between each pair of draws prior to April 2011 produced the unusual pattern on the left. The first and second draws often have the same sign, and cover the plane only in patches. This problem was brought to the attention of StataCorp in March, 2011, and the company acted swiftly; in a matter of days, a software update was released that remedied the problem. Though the Stata 10 documentation did not discuss the consequences of frequently resetting the seed, the current Stata documentation is quite explicit on this topic: it instructs the user, "Do not set the seed too often." Figure 1 then arises from a mis-use of the software, albeit one that was not clearly described as such at the time. The questions remaining are why the problem arose, whether existing tests should have detected it, what it means for researchers, and what researchers and software developers and evaluators should do in the future.

2 The case of rnormal between 2007 and 2011

Stata 10, released in June, 2007, was the first version of Stata to include the `rnormal()` command. The command uses a ziggurat-style algorithm based on that of Knuth, Marsaglia, and

others to produce random normal deviates, as described in the manual (StataCorp 2009). The algorithm produces numbers only from the positive half of the normal distribution, so the final step is to randomly assign the number a positive or negative sign with equal probability. This final step was the source of the problem. While this happened correctly on average, 95.1 percent of the 2^{31} seeds that could be used to initialize the generator produced first and second draws with the same sign, yielding the pattern in Figure 1. No matter what seed was used, the signs of the draws from `rnormal()` followed almost exactly the same pattern.

This is illustrated using a cumulative sum of signs in Figure 2. After 3,400 draws, the sum of signs is near zero: these 3,400 draws include almost an equal number of positive and negative values, regardless of the seed. A consequence of this pattern is that the distribution of the mean of the first 3,400 draws has a lower variance than it should, had these draws been i.i.d. $\mathcal{N}(0, 1)$. Likewise, since there is always an imbalance between positives and negatives in the first 7,600 draws, their mean is bimodally distributed. To illustrate, Figure 3 compares several distributions: the left side shows the distribution of the mean of the first 3,400 draws from `rnormal()`, while the right shows that of the mean of the first 3,400 draws from `invnorm(uniform())`. The upper panel shows this comparison for Stata 10.1 (before the bug fix), while the lower panel shows it for Stata 12.1 (after the 2011 bug fix). Seeds 1 through 100,000 are used to produce the figure; Figure 4 is produced analogously, but for the first 7,600 draws.

This could easily "interfere constructively" with a statistical procedure: a simple case is a T-test of whether the mean of a collection of normally distributed values is significantly different from zero. No matter what range of seeds is used, resetting the seed immediately before making 3,400 draws and taking their mean yields a what should be a 5-percent level test statistic that falsely rejects less than 0.3 percent of the time. Similarly, no matter what range of seeds is used, resetting the seed immediately before making 7,600 draws and taking their mean yields a statistic that falsely rejects more than 13 percent of the time. Neither of these phenomena occur with fully updated Stata 11, or with Stata 12.

2.1 Using standard tests to detect the problem

Though the problem is apparent when shown this way, a natural question is whether existing tests would have detected it. The answer requires two decisions. First, the standard RNG testing suite, TestU01, is designed for uniform distributions between zero and one. To transform a random variable into a uniform one, the usual choice (though not the only one) is to apply the variable's cumulative distribution function to the values. Having done this, the second choice is how to test multiple parallel streams of numbers; TestU01 is designed to test only a single stream. However, I follow the suggestion of the TestU01 authors L'Ecuyer and Simard (2007) on how to handle parallel streams, taking a fixed number of draws, L, after each re-initialization with a new seed. I use $L = 20$, though other values would also work.

With these choices, a relatively small number of draws are necessary for several of the TestU01 tests to reject that the resulting draws were independent and uniformly distributed on (0,1). One of the simplest tests—and a parallel to Figures 3 and 4—compares the empirical distribution of sample means to the theoretical distribution via a Kolmogorov-Smirnov test. With 1000 sample means of 20 consecutive draws each, the test rejects the null hypothesis with a p-value less than 10^{-16}. In contrast, the same test does not reject the null for a number of alternative RNGs, including those that result from applying the same re-initializing procedure to Stata's `uniform()` function or to the `rnormal()` function after the March, 2011 Stata update.[2]

3 Frequent re-initialization versus advancing the state

Though the problem could have been detected this way, and has now been repaired, frequently re-initializing the seed is not the intended use of the RNG, and may be harmful to RNGs in general. To illustrate, I examine the well-known algorithm Stata uses to generate uniformly distributed draws.

Stata's uniform random number generator, using Marsaglia's KISS algorithm, only re-

[2] In C, this test uses the TestU01 function as follows: `svaria_SampleMean(gen, NULL, 1000, 20, 0);` Further details on this and a number of other TestU01 tests are shown in Table 1.

peats after a very long period (around 2^{126}). Stata provides the option to set an initial "seed" that is useful for reproducing results of anything that requires random numbers, such as simulations or bootstrap-based computations. If the seed is reset too frequently, however, the KISS generator is reduced to a less sophisticated generator, with the result that such draws appear less "random." (The most recent version of Stata's documentation discusses this case explicitly as well, admonishing the user to reset the seed sparingly.)

Marsaglia (1968) pointed out a weakness in congruential generators of the form:

$$\text{new } I = K \times \text{ old } I \text{ modulo } M$$

He showed that though these numbers appear at first glance to be almost random, that n-tuples of them fall in relatively few planes in n dimensions; the KISS algorithm, used by the Stata `runiform()` function, is not a generator of this kind. Though it includes a linear congruential generator (LCG), it also includes three others, each with a different period. The resulting random numbers withstand a variety of statistical tests. As a simple illustration, the first five thousand draws from Stata's `uniform()` command are plotted against the second five thousand draws in the left panel of Figure 5. No pattern is discernible. The left panel was generated after setting the initial "seed" only once. However, if after each pair of draws, the seed is incremented and reset—so that the first pair come from seed 1, the second from seed 2, and so on—the striped pattern in the right panel appears.

This is by no means a bug, however. As explained on pp. 223-224 of the Stata Data Management manual (StataCorp 2009), setting the seed in Stata's RNG always re-initializes three of the four recursions to the same starting values, only using the user-provided seed to change the starting point for the LCG. Thus, the differences from one seed to another arise from the LCG alone, and the problems that Marsaglia identified return. It does not matter that the seeds were small integers; large seeds incremented methodically would produce the same pattern.[3]

The seed resetting facility is meant to allow replication of program results, however, not access to independent streams of random numbers. Though some software packages

[3]Though undesirable in some applications, this pattern is quite unlike the recent problem with `rnormal()`.

provide a facility for multiple random number streams—for example, so that a number of independent simulations may run in parallel—the same effect can be achieved by advancing the entire random number generator far enough down the sequence that multiple simulations will not re-use the same numbers (L'Ecuyer 2001, Law and McComas 1994). While the syntax "`set seed 1`" does not do this, Stata does provide the ability to set what it calls the "`c(seed)`," representing the entire state of all four recursions used in the KISS random number generator. The manual discusses how to capture and restore this state, but not how to quickly advance the state by a large number of draws, except by actually making those draws (StataCorp 2009). This could be carried out for streams that are as far apart as desired, up to the period of the KISS random number generator, though the computational time required to find the relevant `c(seed)` by discarding the intervening draws increases linearly, and thus impractically, for very large stream separations.[4]

4 Testing the methods

The pattern that emerges from frequent re-initialization during use of Stata's `uniform()` command is easily detected via the statistical tests included in the TestU01 suite. Stata is not alone in having a seed re-initialization facility that does not hold up to tests of randomness. To illustrate, Table 1 shows several RNGs' performance on collection of very weak tests of randomness from the TestU01 suite. The tests are weak because they only use very small samples from the random stream, but I choose these tests because the failure of an RNG on these tests illustrates the severity of the problem.

The first twelve columns show RNGs provided by Stata, while the last six columns show RNGs from MATLAB. For Stata, three underlying generators are tested: the `uniform()` RNG, in Columns (1) – (4); the original `rnormal()` RNG, in Columns (5) – (8); and the

[4]In Stata 11.2, typing `set seed 0` yields a c(seed) of `X1b5b8174c43f462544a474abacbdd93d00023a41`; advancing 10^{12} draws, which advances the c(seed) to `Xd2c55174405eeffae35b1912f1a5cbf6000206cb`, took roughly 7 hours on a computer using a 2.3 GHz Intel Core i5-2410M processor. Other versions of Stata have the same underlying RNG with the same underlying state, but use different checksums in the last bits of the c(seed). Thus, in Stata 10.1, setting the seed to zero yields the c(seed) of `X1b5b8174c43f462544a474abacbdd93d4b02`, while in Stata 12.1, it yields the c(seed) of `X1b5b8174c43f462544a474abacbdd93d00043a43`. In all three versions, the 32 characters which represent the RNG state (after the `X`, underlined) are identical.

updated `rnormal()` RNG, in Columns (9) – (12). For each Stata RNG, four methods are shown: draws are made after a single initialization; re-initializing after every 20 draws; re-initializing after every draw; or skipping 10,000 draws between each draw included in the stream.

For MATLAB, the three underlying generators tested are the older `'v4'` and `'v5'` RNGs, in Columns (13) through (16), and the newer `'default'` RNG in Columns (17) and (18). Only two methods are shown for each MATLAB RNG: draws are made after a single initialization; or re-initialization to an incremented seed occurs after every draw.

Columns (1), (5), (9), (13), (15), and (17) show that all six RNGs pass these basic tests when initialized only once. Comparing Columns (2), (6), and (10), that the original Stata `rnormal()` RNG behaved particularly poorly when a small number of draws was made before re-initialization: sequences of draws made immediately after re-initializing to a new seed do not behave as if they are independent random sequences. This problem has clearly been resolved in the new version of `rnormal()`, shown in Column (10).

Columns (3), (7), (11), (14), (16), and (18) show that of all six RNGs under consideration, only MATLAB's current default RNG behaves well when re-initialized after every draw. (Note that I am not recommending re-initializing after every draw; I include this as a crude test of the relative independence of separate streams, each initialized with a different seed.) As an alternative to re-initialization, though, one might discard draws in order to advance down a single stream of random numbers, thereby effectively splitting the stream into multiple shorter ones. This possibility is explored in Columns (4), (8), and (12). By the standards set by these weak tests, this approach appears to have all the same desirable statistical properties of the single-initialization approach.

The "advancing" approach presents computational drawbacks, however. As mentioned above, Stata provides no facility for quickly advancing the state of the RNG without actually making random draws, so this process could take minutes, hours, or longer, depending on how far ahead one wants to advance.

5 Conclusion

The consistent sign pattern of `rnormal()` across seeds was problematic. A cursory inspection suggests that this problem did not affect other pseudorandom number functions, such as `rt()`. Prior to Stata 10, the `rnormal()` function did not exist, so this problem should only have affected users of Stata 10 and Stata 11 between 2007 and 2011, and only if seeds were reset frequently as a part of users' programs. The unusual pattern of signs shown by `rnormal()` was rectified in a March, 2011 update to the Stata executable, so while users should be aware of the vulnerability of programs and results between 2007 and 2011, a fully updated version of Stata should no longer display this behavior.[5]

Until tests of parallel streams become standard, a few options remain for researchers desiring to parallelize (or distribute) simulations. If one is using software that explicitly allows multiple random streams (as is the case in recent MATLAB releases), there is no impediment to parallelization. In Stata, there are presently three alternatives. First, following Stata's guidelines exactly, one may set the seed exactly once, and proceed making only serial draws from Stata's RNGs. Second, at potentially high computational expense, one may discard a large number of random draws in order to quickly advance the RNG state (the "`c(seed)`"), and make parallel draws from streams initialized in this way. These first two options should provide very similar results. Third, one may, at some risk, initialize each stream separately to an integer, knowing that the problem of Marsaglia's planes still applies, but that the bug in `rnormal()` has now been fixed.[6]

While a growing literature and collection of software tools provide statistical tests of randomness for a single stream of random numbers, use of these tests for parallel streams is not yet commonplace. That multiple-stream tests are not yet standardized presents a risk for researchers interested in parallelizing simulations; this manuscript is the first comparison of econometric software on the basis of multiple streams from RNGs. Though the case

[5] In general, the robustness of a software program's uniform RNG may not guarantee the robustness of its other RNGs; a recently uncovered bug in an algorithm for producing normal random variates illustrates one such case (Tirler, Dalgaard, H ormann, and Leydold 2004).

[6] A researcher interested in testing whether a particular installation of Stata has been updated to fix the `rnormal()` bug can simply type these three commands: First, `set seed 4` , then twice, `di rnormal()` . If the two resulting numbers have the same sign, that Stata installation is out of date. If they have different signs, it is up to date. One can try other seeds to see the pattern.

presented here is specific to Stata, the general problem of untested parallel streams is more pervasive. Tests of econometric software must be brought back into accord with the methods of using that software. Future software evaluators would do researchers and developers a service by including not only standards of "random" number generation in their assessments, but also tests of parallel random streams.

References

Busso, M., J. DiNardo, and J. McCrary (2011): "New Evidence on the Finite Sample Properties of Propensity Score Reweighting and Matching Estimators," *mimeo*, University of California, Berkeley.

Compagner, A. (1995): "Operational conditions for random-number generation," *Physical Review E*, 52(5), 5634–5645.

Keeling, K. B., and R. J. Pavur (2007): "A comparative study of the reliability of nine statistical packages," *Computational Statistics and Data Analysis*, 51(8), 3811–3831.

Knuth, D. E. (1998): *The art of computer programming*. Addison-Wesley, Reading, MA, 3 edn.

Law, A. M., and M. G. McComas (1994): "Simulation software for communications networks: the state of the art," *IEEE Communications Magazine*, 32(3), 44–50.

L'Ecuyer, P. (2001): "Software for uniform random number generation: distinguishing the good and the bad," in *Proceedings of the 2001 Winter Simulation Conference*, ed. by B. A. Peters, J. S. Smith, D. J. Medeiros, and M. W. Rohrer, pp. 95–105.

L'Ecuyer, P., and R. Simard (2007): "TestU01: A C Library for Empirical Testing of Random Number Generators," *ACM Transactions on Mathematical Software*, 33(4), article 22.

Marsaglia, G. (1968): "Random numbers fall mainly in the planes," *Proceedings of the National Academy of Sciences*, 61(1), 25–28.

McCullough, B. D. (2006): "A Review of TESTU01," *Journal of Applied Econometrics*, 21(5), 677–682.

Moler, C. (2007): "Parallel MATLAB: Multiple processors and multiple cores," *The MathWorks News and Notes*, June.

Odeh, O. O., A. M. Featherstone, and J. S. Bergtold (2010): "Reliability of Statistical Software," *American Journal of Agricultural Economics*, 92(5), 1472–1489.

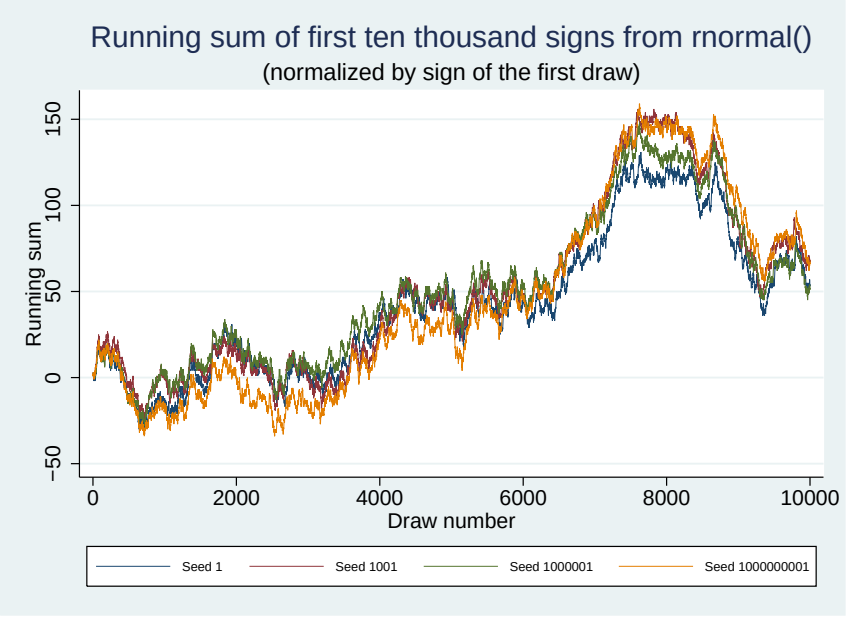
Ripley, B. D. (1990): "Thoughts on pseudorandom number generators," *Journal of Computational and Applied Mathematics*, 31(1), 153–163.

Smeeton, N., and N. J. Cox (2003): "Do-it-yourself shuffling and the number of runs under randomness," *Stata Journal*, 3(3), 270–277.

StataCorp (2009): *Stata data-management reference manual: Release 11*. StataCorp LP, College Station, TX.

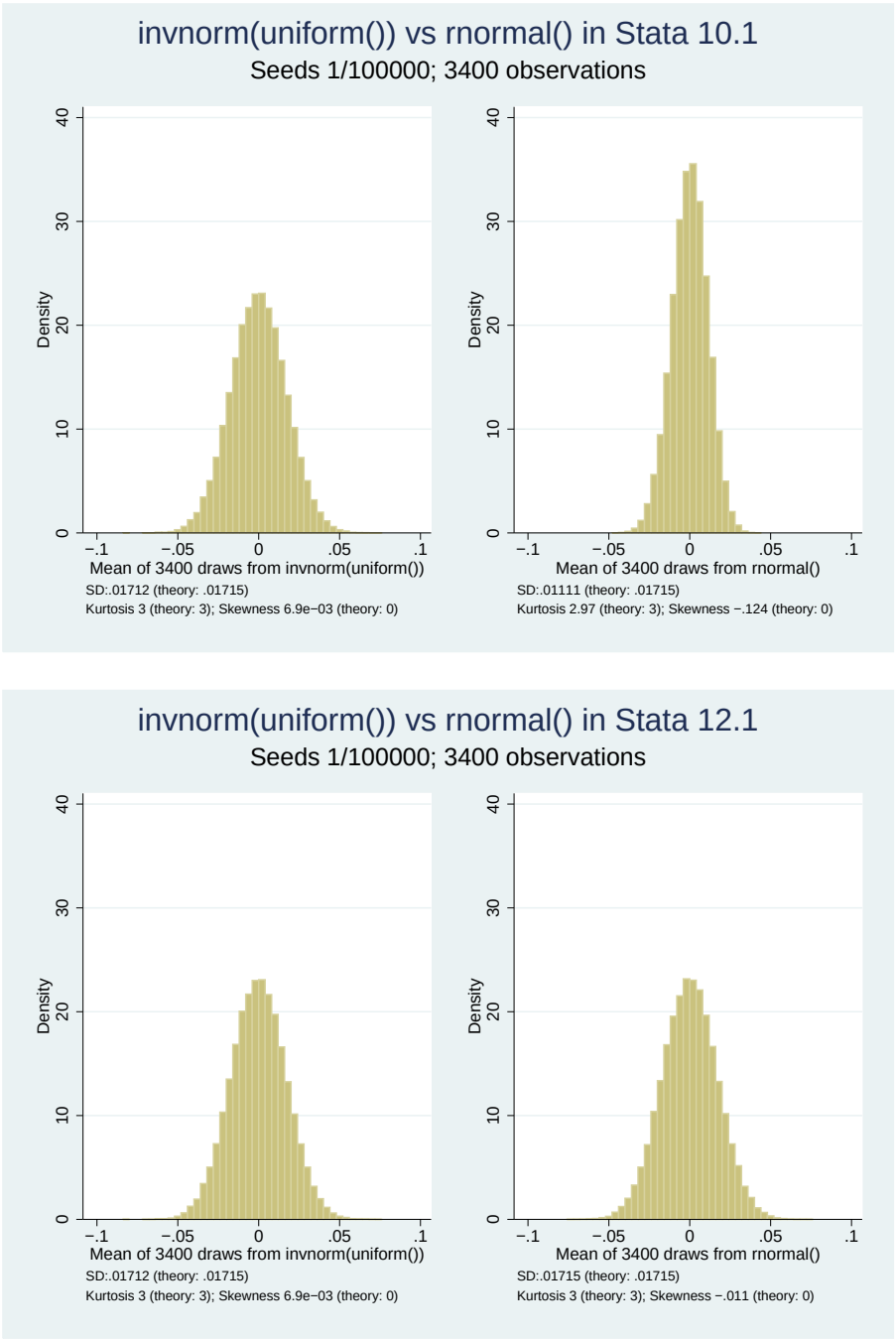
Tirler, G., P. Dalgaard, W. H ormann, and J. Leydold (2004): "An Error in the Kinderman-Ramage Method and How to Fix It," *Computational Statistics and Data Analysis*, 47(3), 433–440.

Figure 2: Running sum of signs from rnormal() with different seeds



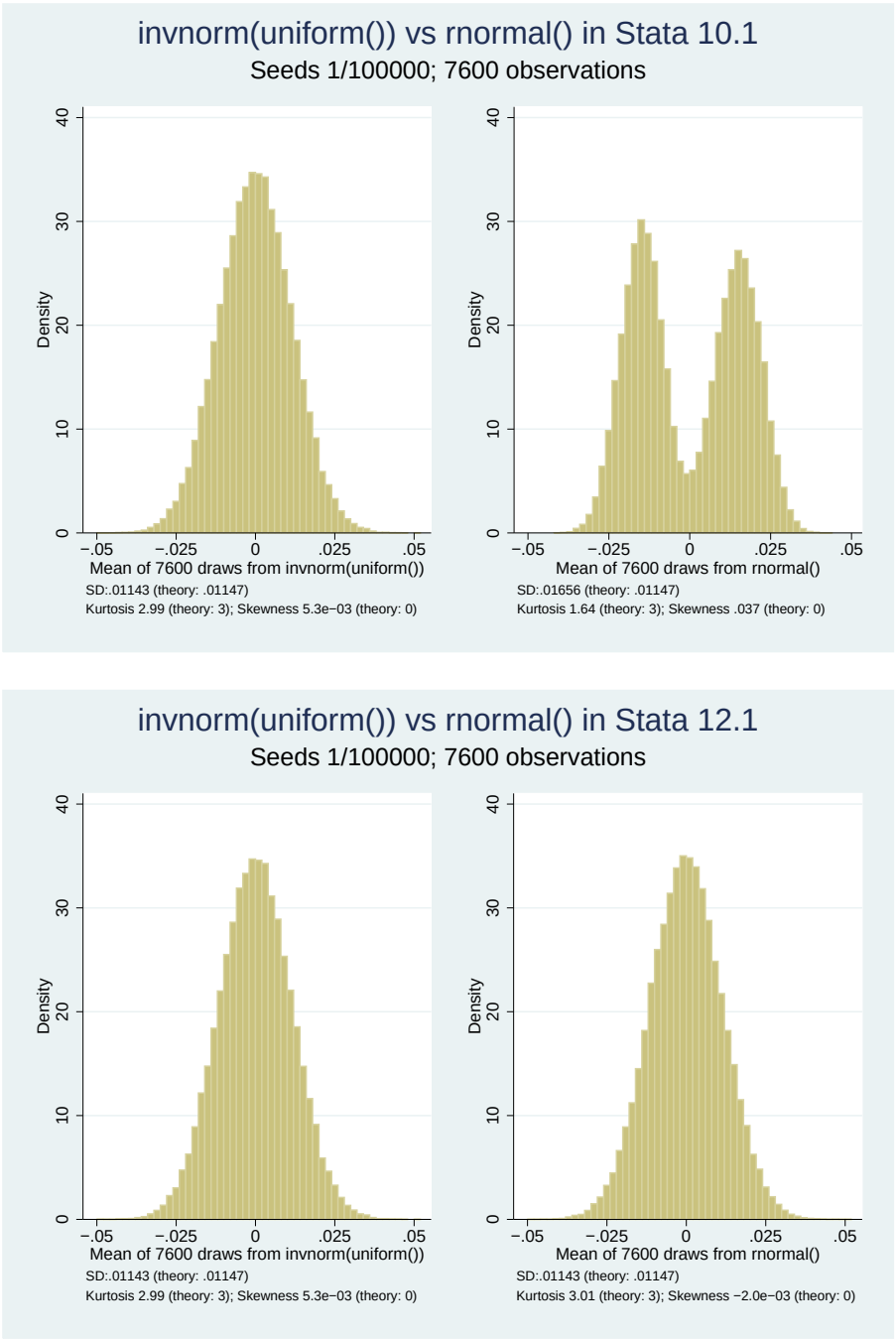
Note: this problem was resolved in the March, 2011 Stata update.

Figure 3: Distribution of sum of first 3,400 draws: `invnorm(uniform())` versus `rnormal()`



Histograms of the distribution across 100,000 initialization seeds of the mean of the first 3,400 normal draws in Stata. Upper panel shows results using Stata 10.1; lower panel shows Stata 12.1. Left histograms show results using `invnorm(uniform())`; right histograms show `rnormal()`.

Figure 4: Distribution of sum of first 7,600 draws: `invnorm(uniform())` versus `rnormal()`



Histograms of the distribution across 100,000 initialization seeds of the mean of the first 7,600 normal draws in Stata. Upper panel shows results using Stata 10.1; lower panel shows Stata 12.1. Left histograms show results using `invnorm(uniform())`; right histograms show `rnormal()`.

Figure 5: Whether the seed is reset makes a difference

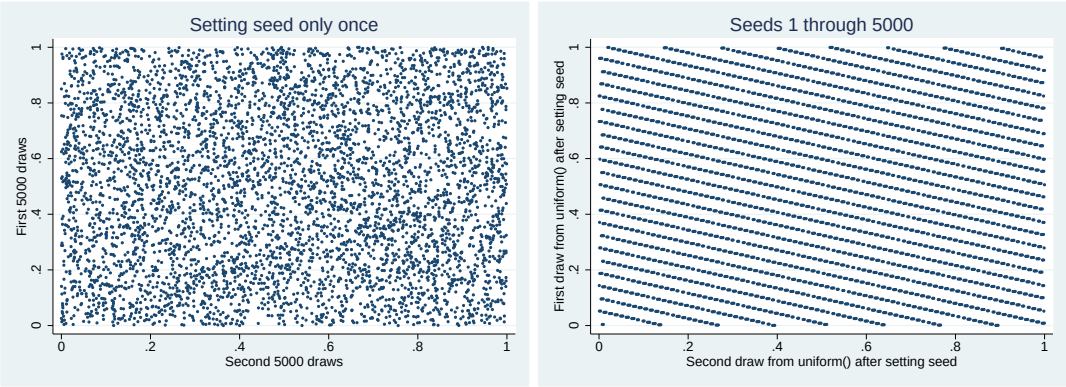


Table 1: Testing generators

Program:	Stata												MATLAB					
Generator:	`uniform()`				`normal(rnormal())`								`'v4'`		`'v5uniform'`		`'default'`	
Date:					*Jun 2007 - Mar 2011*				*Apr 2011 - present*									
Re-initialization method:[a]	0 (1)	20 (2)	1 (3)	A (4)	0 (5)	20 (6)	1 (7)	A (8)	0 (9)	20 (10)	1 (11)	A (12)	0 (13)	1 (14)	0 (15)	1 (16)	0 (17)	1 (18)
`sknuth_Collision` $N = 1$, $n = 10^3$, $d = 2^9$, $t = 2$	ok	ok	(***)	ok	ok	ok	(***)	ok	ok	ok	(***)	ok	ok	(***)	ok	(***)	ok	ok
`sknuth_Gap` $N = 1$, $n = 10^3$, $\alpha = 0.0$, $\beta = 0.125$	ok	ok	(***)	ok	ok	(***)	(***)	ok	ok	ok	(***)	ok	ok	(***)	ok	(***)	ok	ok
`svaria_WeightDistrib` $N = 1$, $n = 10^3$, $k = 20$, $\alpha = 0.0$, $\beta = 0.125$	ok	ok	(***)	ok	ok	(***)	(***)	ok	ok	ok	ok	ok	ok	(***)	ok	(***)	ok	ok
`smarsa_MatrixRank` $N = 1$, $n = 10^3$, $s = 2$, $L = K = 20$	ok	ok	(***)	ok	ok	(***)	(***)	ok	ok	ok	(***)	ok	ok	(***)	ok	ok	ok	ok
`smarsa_RandomWalk1` $N = 1$, $n = 10^3$, $s = 2$, $L_0 = L_1 = 100$	ok	ok	(***)	ok	ok	(***)	(***)	ok	ok	ok	(*)	ok	ok	(***)	ok	(*)	ok	ok
`smarsa_RandomWalk1` $N = 1$, $n = 10^4$, $s = 10$, $L_0 = L_1 = 160$	ok	ok	(***)	ok	ok	(***)	(***)	ok	ok	ok	(*)	ok	ok	(***)	ok	(*)	ok	ok
`smarsa_RandomWalk1` $N = 1$, $n = 10^5$, $s = 20$, $L_0 = L_1 = 160$	ok	ok	(***)	ok	ok	(***)	(***)	ok	ok	ok	(*)	ok	ok	(***)	ok	(***)	ok	ok
`svaria_SampleMean` $N = 10^3$, $n = 20$	ok	ok	(***)	ok	ok	(***)	(***)	ok	ok	ok	(***)	ok	ok	(***)	ok	(***)	ok	ok
`sknuth_CouponCollector` $N = 1$, $n = 10^4$, $d = 20$	ok	(***)	(***)	ok	ok	(***)	(***)	ok	ok	ok	(***)	ok	ok	(***)	ok	(***)	ok	ok

Test results are coded as three asterisks (***) if all tests performed by the specified routine returned a p-value less than 0.001; if the routine returns some p-values less than 0.001 but others that are higher, the result is coded as one asterisk (*); if all p-values exceed 0.001, the result is coded as "ok." MATLAB version R2011b was used for columns (13)-(19); An fully updated version of Stata 11.2 was used in columns (1)-(4) and (9)-(12); Stata 10.1 was used in columns (5)-(8). Tests were performed using TestU01 version 1.2.3; parameter value $r = 0$ was used for all tests.

[a] Re-initialization method descriptions: 0=one initialization at start; 20=increment seed and re-initialize after every twenty draws; 1=increment seed and re-initialize after every draw; A=one initialization at start, but advance 10,000 draws between draws.