JOURNAL OF APPLIED ECONOMETRICS
J. Appl. Econ. **21**: 677–682 (2006)
Published online in Wiley InterScience
(www.interscience.wiley.com) DOI: 10.1002/jae.917

SOFTWARE REVIEWS

A REVIEW OF TESTU01

B. D. McCULLOUGH*
Department of Decision Sciences, Drexel University, Philadelphia, PA, USA

1. INTRODUCTION

The scientific literature is filled with examples of bad random number generators (RNGs), and a bad RNG can completely ruin a researcher's analysis. See Coddington (1994, 1996) for examples and discussion from the field of physics. Examples from economics are difficult to find, in part, because many economists do not bother to test the RNGs that they use in their applied work. This may be due to the fact that they are unaware of any need to test the RNG. Econometrics texts simply assume (incorrectly) that RNGs work perfectly as advertised, despite the fact that many RNGs found in commercial software routinely fail randomness tests (L'Ecuyer, 1997, 2001). This is all the more shocking, since simulation results depend on the quality of the random numbers used as input—though one would hardly know this from reading books on simulation (e.g., Gourieroux and Montfort, 1996). If experts and textbooks fail to address the issue, the practicing economist can hardly be faulted.

Even among economists who are aware of the issue, there is a hesitancy to test RNGs due a common sentiment: 'I would probably just find that my RNGs fail one or more obscure tests, and I would have no idea of how to interpret such failures.' Such a sentiment is understandable, but it misconstrues the purpose of randomness testing. As a practical matter, it is the province of the RNG expert, and not the economist, to interpret these failures. The economist's job is simply to avoid RNGs that fail tests—not all tests, for there is no RNG that will pass all tests. The economist should avoid RNGs that fail 'reasonable' tests. What constitutes a reasonable test has been succinctly stated by L'Ecuyer (2004):

> [T]he number of tests is huge and all but a tiny fraction are too complicated even to be implemented. So we may say that bad RNGs are those that fail simple tests, whereas good RNGs fail only complicated tests that are hard to find and run. This common-sense compromise has been generally adopted in one way or another.

* Correspondence to: B. D. McCullough, Department of Decision Sciences, Drexel University, Philadelphia, PA 19104, USA. E-mail: bdmccullough@drexel.edu

So the simple answer is to trust acknowledged RNG experts to define ‘reasonable’ and not worry about the matter further. Persons who will settle for nothing less than a complex answer are encouraged to do a literature search and examine in detail the specific test that is failed.

An initial battery of statistical tests for uniform RNGs was offered by the 1969 first edition of Knuth (1997). In popular testing of RNGs, Knuth’s tests were supplanted by Marsaglia’s (1996) DIEHARD tests, and DIEHARD has been the standard for several years.[1] To use Marsaglia’s program, the user creates a file of three million random numbers, and the program analyzes these numbers. There are some notable difficulties with DIEHARD. First, it is not user-friendly. Second, it does not offer many tests—about 15. Third, the parameters of the tests are fixed, and it is often advantageous to vary them—see the online appendix to this article at the *JAE* archive for a demonstration of this point (www.econ.queensu.ca/jae). Fourth, it is not extensible—new tests cannot be added. Fifth, while these tests might have been stringent 10 years ago, they are not now. L’Ecuyer, long one of the world’s top RNG researchers in general, and specifically one of the very best in testing of RNGs, is just the person to remedy these deficiencies and produce the successor to DIEHARD. He has done so with the programming assistance of Simard, and together they have created TESTU01. Just as many RNGs that passed the Knuth tests failed the DIEHARD tests, so many RNGs that pass the DIEHARD tests fail the TESTU01 tests.

2. TESTING RANDOM NUMBER GENERATORS

Let us take a concrete example of a popular RNG that fails the ‘mtuple’ test in the DIEHARD battery of randomness tests, Marsaglia’s Multicarry RNG. Imagine trying to do a Value-at-Risk simulation using a random normal generator. Suppose that your statistical package uses the multicarry to generate random uniform deviates, and transforms the uniforms to normal via the Kinderman–Ramage method.[2] Consider the following program: generate a vector x of 5000 random normals. Define $y = \max[x]$. Repeat 4000 times. Plot the 4000 y’s (see Figure 1). Multiplying a loss value by the probability of that loss would produce aberrant results for tail events that fall between 3.5 and 3.7 on the ordinate—if the unsuspecting user was unfortunate enough to have a package that used Marsaglia Multicarry. No such difficulty would arise for the user whose package employed a better RNG, e.g., Wichmann–Hill.

Simply switching the uniform RNG to Wichmann–Hill (or some other RNG that passes all the DIEHARD tests) solves the problem. For more modest goals, e.g., checking whether a z-test really rejects 95% of the time, the combination of the Marsaglia Multicarry RNG with the Kinderman–Ramage transformation may work perfectly well, though other combinations (e.g., the Marsaglia–Zaman ‘subtract with borrow’ RNG and the Box–Muller transform to normality) might not. Monahan (2001, p. 256) noted, ‘Some generators with notorious defects may work very well for many applications, yet fail on another.’ Consequently, it is natural for a researcher to wonder, ‘Will the RNG’s failure to pass a randomness test affect my simulation results?’ The only way to tell is to run the simulation twice, once with the RNG in question, and again with a better RNG, and compare the results. But if the only way to tell is to use a better RNG, why not use the better RNG to begin with? L’Ecuyer and Hellekalek (1998, §3.1) address this specific

[1] On the importance of subjecting random number generators (RNGs) to empirical tests, see McCullough and Vinod (1999, §5) or Gentle (2003, §2.3).

[2] The original Kinderman–Ramage algorithm has an error that only recently has been discovered and corrected (Tirler *et al*., 2004). However, the error plays no role in the simulation that follows.

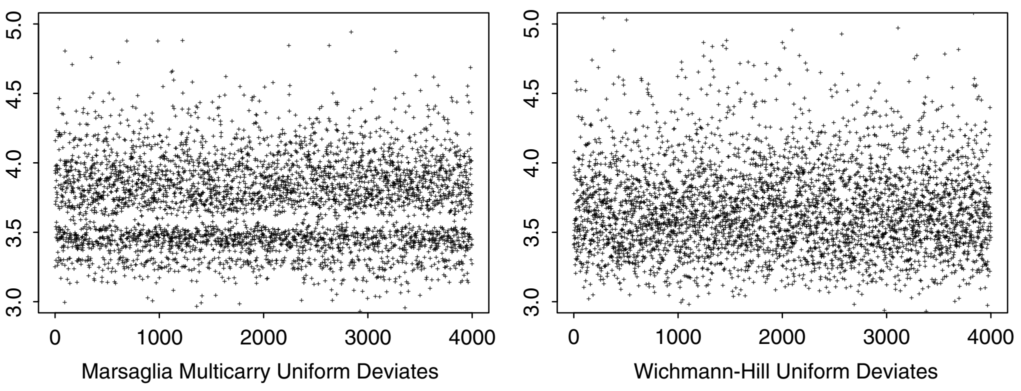

Figure 1. Plots of max$[y_i]$, $i = 1, \ldots, 4000$ where y_i is a sample of 5000 standard normals produced by applying the Kinderman–Ramage transform to uniform variates generated by Marsaglia Multicarry (left panel) and Wichmann–Hill (right panel)

issue: '[A]ny RNG has its own particular structure and bad things happen when the structure of the simulation problem at hand interacts constructively with the RNG. The difference between the good and bad RNGs is that the former have their structure better hidden, so that the problems that constructively interact with them are harder to find and occur less often in practice.'

Further simple examples of RNG failure producing inaccurate results are somewhat hard to come by, at least in economics,[3] for two reasons. First, most work in economics cannot be replicated (McCullough *et al.*, 2006); the experiment cannot be repeated with a different RNG if it cannot initially be replicated with the original RNG. Second, in those instances where work is replicable, the software package in question often does not provide a choice of RNGs, e.g., RATS or LIMDEP. A notable exception is GAUSS, which allows users to supply their own RNGs—more packages should follow this fine example. Not only does GAUSS allow experimentation with different RNGs, but the user is not wedded to the developer's choice of RNG: the user should be free to make his own choice.

3. TESTU01

TESTU01 provides a large variety of tests, which are listed in the online appendix. Each of the individual tests is actually a family of tests, since the user is free to select the parameters of the test. Additionally, TESTU01 comes with a large variety of pre-programmed RNGs. For example, the Wichmann–Hill RNG is accessed via the `ulcg_CreateCombWH3()` command. Marsaglia's KISS RNG is pre-programmed as `umarsa_CreateKISS()`. With respect to these pre-programmed RNGs, L'Ecuyer and Simard (2003, p. 8) make the following point: 'We emphasize that the generators provided here are not all recommendable; in fact, *most of them are not*' (italics in the original.) However, TESTU01 also provides facilities for combining these pre-programmed RNGs, and such combinations can produce very good RNGs.

[3] The discipline of physics stands in sharp contrast to economics, not just because physicists emphasize and expect replicability, but because this emphasis has produced a long list of simulations that failed due to bad RNGs (see, for example, Coddington, 1996, and Ferrenberg *et al.*, 1992).

TESTU01 offers four groups of modules for analyzing RNGs: (i) implementing (pre-programmed) RNGs; (ii) implementing specific statistical tests; (iii) implementing batteries of statistical tests; and (iv) applying tests to entire families of RNGs. This review concerns itself with (i), (ii) and (iii), though (iv) is absolutely fascinating and merits a brief mention. When a specific test is applied to a sample of size n produced by an RNG, the p-value of the test usually will remain 'reasonable' as the sample size increases until the sample size hits n_0, say. After that, the p-value diverges to 0 or 1 with exponential speed. Module (iv) allows the researcher to investigate how large the sample size should be, as a function of the RNG's period, before the RNG starts to fail the test systematically.

TESTU01 is available free of charge from

http://www.iro.umontreal.ca/∼simardr/indexe.html

The source code is available for Unix/Linux and Windows, and both Cygwin and MinGW binaries are available for Windows. The manuals that accompany TESTU01 as pdf files (L'Ecuyer and Simard 2003a, 2003b, 2003c) are written for persons who are at least somewhat conversant in the C programming language. Yet it is still possible for persons who are not fluent in C to use TESTU01 to test RNGs that L'Ecuyer and Simard have already programmed into TESTU01. This process becomes easier with each successive version of TESTU01, which has been revised frequently in the past, and probably will be revised again before this review appears in print. Therefore this review presents no specific details of using TESTU01; these are presented in an appendix available at the *JAE* archive.

As far as testing is concerned, most users who are not specialists in random numbers will not want to choose from among the many statistical tests available, set parameters for each one and apply them serially; i.e., most users will not be concerned with (ii) above. Rather, most users will want to use the pre-programmed batteries of tests (iii) to test the RNGs that are used in their statistical packages. To satisfy the needs of such users, TESTU01 offers three batteries of tests called 'Small Crush' (which consists of 10 tests), 'Crush' (60 tests) and 'Big Crush' (45 tests).[4] The specific tests applied by each battery are given in the User Guide.[5] On a 1.7 GHz Pentium 4 running Red Hat Linux 9.0, for a simple RNG, Small Crush takes about 2 minutes. Crush takes about 1.7 hours. Big Crush takes about 12 hours. (For a more complex RNG, all these times increase by a factor of two or more.) By contrast, the DIEHARD tests take about 15 seconds to run.

Table I shows the results for testing all the uniform RNGs offered by version 1.6.1 of the software package 'R' (Ihaka and Gentleman, 1996). For example, Big Crush takes 15 hours when applied to the Marsaglia Multicarry, which fails 19 of the tests. The online appendix describes in detail how TESTU01 was applied to these RNGs. It gives examples of how to test a pre-programmed RNG, how to combine two or more pre-programmed RNGs and test the resulting RNG, and how to test an RNG for which C code is available.

[4] For purposes of comparison, TESTU01 offers other batteries of tests, e.g., the Marsaglia tests and the NIST tests. All the Knuth tests are offered, but not as a battery, because Knuth didn't specify parameters for his tests.

[5] L'Ecuyer and Simard make the point that parameters for the batteries may change in the future, as well as the specific tests included in each battery. So persons using TESTU01 should indicate not just that a particular RNG passed (or failed), e.g., Big Crush, but should also specify the version of TESTU01 that was applied, e.g., Big Crush version 1.10.

Table I. Crush tests applied to several RNGs from 'R' time in hours : minutes : seconds

RNG	Catastrophic failures (CPU time)		
	Small Crush	Crush	Big Crush
Marsaglia Multicarry	1 (00 : 00 : 46)	13 (01 : 44 : 34)	19 (15 : 06 : 40)
Super-Duper	1 (00 : 00 : 48)	9 (01 : 47 : 21)	12 (15 : 21 : 20)
Wichmann-Hill-1	1 (00 : 01 : 03)	3 (02 : 18 : 39)	4 (19 : 57 : 47)
Wichmann-Hill-2	1 (00 : 01 : 21)	3 (02 : 56 : 37)	4 (25 : 39 : 25)
Mersenne Twister	0 (00 : 00 : 46)	0 (01 : 52.22)	0 (15 : 58 : 25)
Knuth TAOCP	0 (00 : 00 : 44)	0 (01 : 38 : 29)	0 (14 : 20 : 26)
Knuth TAOCP 2002	0 (00 : 00 : 44)	0 (01 : 39 : 46)	0 (14 : 29 : 36)

Many RNGs pass TESTU01. Marsaglia noted that there are too many RNGs that pass all the DIEHARD tests for there to be any reason to use one that does not pass all the tests. The same is true of TESTU01.

If a researcher is fortunate enough to want to test an RNG that is pre-programmed in TESTU01, the researcher should, having read this review, have little difficulty in testing that RNG. If the RNG in question has not been pre-programmed, then the researcher, by careful study of the Multicarry example in the online appendix and some reading of the User Guide (L'Ecuyer and Simard, 2003), may be able to test his RNG nonetheless.

4. CONCLUSIONS

There are far too many RNGs to be tested, and L'Ecuyer and Simard cannot pre-program them all. In addition to the great service they have done to users of RNGs by creating the successor to DIEHARD, it would be another great service if L'Ecuyer and Simard would establish a web repository so that users of TESTU01 can deposit their results. Such a website would be a useful source of information for all users of RNGs.

It should be obvious that TESTU01 has supplanted DIEHARD, and TESTU01 is now the standard for testing uniform RNGs. While economists who perform simulations need not know much about TESTU01, they should at least know enough to make sure that whatever software packages they use have RNGs that have been subjected to (and passed!) the tests in TEST01.

ACKNOWLEDGEMENTS

Thanks to P. L'Ecuyer, R. Simard, B. Wichmann and J. MacKinnon for comments on previous versions.

REFERENCES

Coddington PD. 1994. Analysis of random number generators using Monte Carlo simulation. *International Journal of Modern Physics C* **3**: 547–560.

Coddington PD. 1996. Tests of random number generators using Ising model simulations. *International Journal of Modern Physics* **7**(3): 295–303.

Ferrenberg AM, Landau DP, Wong YJ. 1992. Monte Carlo simulations: hidden errors from 'good' random number generators. *Physical Review Letters* **69**: 3382–3384.

Gentle JE. 2003. *Random Number Generation and Monte Carlo Methods* (2nd edn). Springer: New York.

Gourieroux C, Montfort A. 1996. *Simulation-Based Econometric Methods*. Oxford University Press: New York.

Ihaka R, Gentleman R. 1996. R: a language for data analysis and graphics. *Journal of Computational and Graphical Statistics* **5**: 299–314.

Knuth DE. 1997. *The Art of Computer Programming. Vol. 2: Seminumerical Algorithms*. Addison-Wesley: Reading, MA.

L'Ecuyer PL. 1997. Bad lattice structures for vectors of non-successive values produced by some linear recurrences. *INFORMS Journal on Computing* **9**(1): 57–60.

L'Ecuyer PL. 2001. Software for uniform random number generation: distinguishing the good and the bad. In *Proceedings of the 2001 Winter Simulation Conference*. IEEE Press: Piscataway, NJ.

L'Ecuyer PL. 2004. Random number generation. In *Handbook of Computational Statistics*, Gentle JE, Haerdle W, Mori Y (eds). Springer: New York.

L'Ecuyer PL, Hellekalek P. 1998. Random number generators: selection criteria and testing. In *Random and Quasi-Random Point Sets, Lecture Notes in Statistics, no. 138*. Springer: Bertin.

L'Ecuyer P, Simard R. 2003a. TESTU01: a software library in ANSI C for empirical testing of random number generators, user's guide, detailed version (comes with the software TESTU01). Manuscript, Département d'informatique et de recherche opérationnelle, University of Montreal.

L'Ecuyer P, Simard R. 2003b. MYLIB-C: a small library of basic utilities in ANSI C (comes with the software TESTU01). Manuscript, Département d'informatique et de recherche opérationnelle, University of Montreal.

L'Ecuyer P, Simard R. 2003c. PROBDIST: a software library of probability distributions and goodness-of-fit statistics in ANSI C (comes with the software TESTU01). Manuscript, Département d'informatique et de recherche opérationnelle, University of Montreal.

Marsaglia G. 1996. *The Marsaglia random number CDROM including the diehard battery of tests of randomness*. www.csis.hku.hk/∼diehard [accessed 7 June 2006].

McCullough BD, Vinod HD. 1999. The numerical reliability of econometric software. *Journal of Economic Literature* **37**(2): 633–655.

McCullough BD, McGeary KA, Harrison TD. 2006. Lessons from the JMCB archive. *Journal of Money, Credit and Banking* (to appear).

Monahan JF. 2001. *Numerical Methods of Statistics*. Cambridge University Press: New York.

Tirler G, Dalgaard P, Hörmann W, Leydold J. 2004. An error in the Kinderman–Ramage method and how to fix it. *Computational Statistics and Data Analysis* **47**(3): 433–440.