# Assessing the Quality of Pseudo-Random Number Generators

P. C. S. Luizi[*] paulocesar@ufmg.br

F. R. B. Cruz[†] [‡] fcruz@est.ufmg.br

J. van de Graaf[§] jvdg@lcc.ufmg.br

March 19, 2010

*Abstract* — **In this article, we describe a new yet simple statistical procedure to better assess the quality of pseudo-random number generators. The new procedure builds on the statistical test suite proposed by the National Institute of Standards and Technology (NIST) and is especially useful for applications in economics. Making use of properties of the binomial distribution, we estimate the conjoint significance level of the test. We apply the proposed procedure to several well-known pseudo-random number generators, and the results confirm its effectiveness.**

*Keywords* — Pseudo-random numbers; test for randomness; random number generators.

**AMS Subject Classification:** 65C10; 68U20.

## 1 Introduction

Generating pseudo-random numbers seems to be a key issue across many different practical contexts. Mainly due to the popularity of Markov chain Monte Carlo (MCMC) simulation methods and bootstrap methods, which are widely used in computational economics (Kolsrud, 2008; Diks et al., 2008; Lima and Tabak, 2009), pseudo-random number generation is a pervasive topic in the literature (Kundu and Gupta, 2007). In Kimbrough and Murphy (2009), for instance, pseudo-random number generation is the basis of their study of the properties of equilibrium in oligopolies. Kimbrough and Murphy (2009) have run extensive simulations that were analyzed using appropriate statistical tools. Additionally, with the emerging success of stochastic methods for optimization, particularly the so-called evolutionary algorithms, pseudo-random number generation has also been the focus of many studies, as it is on the base of these methods (Ali, 2007). In economics, the use of genetic and other evolutionary algorithms is grounded in the ability of these methods to represent the capacity that individuals have to adapt and learn over time (Maschek, 2010). Finally, we must mention the great importance of random number generation in cryptography; in this context, unpredictable data must be inserted during transmission, and random numbers must be used to generate digital signatures and security keys to ensure security (Rukhin et al., 2001). Yet the search for even better pseudo-random number generators with respect to quality and computational efficiency still continues (Wichmann and Hill, 2006).

Testing a given pseudo-random number generator involves identifying many of its features to evaluate its computational performance, randomness and the degree of independence of the values that it generates. There is a long list of tests designed to perform such tasks. In this context, we note Crypt-X, which is a package used to efficiently test stream ciphers (*i.e.*, random bit generators) developed by the Information Security Institute of the Queensland University of Technology, Australia (see Information Security Institute, 2008, for a description of the statistical tests included in this package). Considered by Knuth (1998) to provide the most rigorous battery of tests, Marsaglia's DIEHARD (Marsaglia, 1995) is also very popular. In fact, at Duke University DIEHARD has been extended into what is called 'dieharder', which is a suite of several pseudo-random number generation tests[1]. Finally, there is a comprehensive statistical test suite developed by the National Institute of Standards and Technology (NIST) for assessing quality mainly for cryptographic applications (Rukhin et al., 2001). For our convenience, we build on the latter suite, but we note the methodology developed here could be applied to any test suite.

The procedure we propose here combines the results from the NIST suite using binomial distribution in order to control the conjoint significance level, which to the best of our knowledge has not yet been done. One could argue that a better usage of NIST suite is to use it hundreds, or even thousands, of times for different seeds in order to precisely identify when the generator fails; this is especially crucial in cryptography, as would-be crackers may find ways to reduce the apparent entropy of the

[*]Department of Statistics, Federal University of Minas Gerais, 31270-901 - Belo Horizonte - MG, Brazil

[†]Department of Statistics, Federal University of Minas Gerais, 31270-901 - Belo Horizonte - MG, Brazil. On sabbatical leave.

[‡]Corresponding author. E-mail: fcruz@est.ufmg.br Phone: +55 31 3409 5929. Fax: +55 31 3409 5924

[§]Laboratory of Scientific Computation, Federal University of Minas Gerais, 31270-901 - Belo Horizonte - MG, Brazil

[1]See http://www.phy.duke.edu/~rgb for a list of ongoing work.

series and the space that must be exhaustively searched to crack a given encryption. However, the aim here is to develop a simple and easy-to-implement testing methodology that could be automated to produce fairly reliable classifications of pseudo-random number generators as a first step prior to more rigorous and time-consuming analysis.

The remainder of this article is organized as follows. In Section 2 we briefly describe the statistical tests from the NIST suite, and in Section 3 we explain in detail how these tests could be combined to estimate a conjoint $p$-value. The proposed procedure is applied to several well-known pseudo-random number generators, and the results are presented in Section 4. We conclude the article in Section 5 with final remarks and topics for future research.

## 2 The NIST Statistical Tests

The NIST test suite was developed in 2000 with the publication of the special report 800-22 (Rukhin et al., 2001). It is comprised of fifteen tests, as presented in Table 1. The strategy suggested by NIST is presented in Fig. 1. For the sake of conciseness, details about these statistical tests are not given here, as they can easily be found elsewhere (Rukhin et al., 2001). For each test, detailed instructions are given to compute the corresponding $p$-values and to perform the following hypotheses tests.

$$\begin{cases} H_0 : \text{the sequence is random;} \\ H_a : \text{the sequence is } \textit{not} \text{ random.} \end{cases}$$

## 3 Toward a conjoint $p$-value

### 3.1 Preliminaries

According to the scheme presented in Fig. 1, for each statistical test, a set of $p$-values corresponding to the number of sequences that were generated is produced. The statistical analysis of these data could be conducted in various ways. NIST includes (i) an analysis of the distribution of the $p$-values, which is expected to be approximately uniformly distributed under $H_0$, and (ii) an analysis of the proportion of sequences that passes the test, that is, for which $H_0$ is not rejected because the $p$-value is above a given critical value (say, $\alpha = 0.01$ for a 1% level of significance).

The idea behind the first analysis is that if a generator fails a test, it will produce some non-uniform distribution of $p$-values. If the test is run enough times, the distribution of the $p$-values will approach its non-uniform asymptotic form and produce $p$-values arbitrarily close to zero in a Kolmogorov-Smirnov test (Marks, 2007). Alternatively, a binned multinomial test could be used (or any other statistical test of uniformity) at perhaps lower resolution as well as with lower computational effort (for details on the classical $\chi^2$ test and an interesting application on fraud detection, see Geyer and Williamson, 2004).

To perform the proportion analysis, empirical results are collected for each one of the tests regarding the number of sequences that passes the test, that is, for which $p$-value $\geq \alpha$. Notice that there is an ongoing debate as to whether the use of this decision rule is appropriate. Over ten years ago, Glaser (1999) raised concerns regarding the misuses and misinterpretations of $p$-values in a detailed and well-documented analysis of the history behind significance testing. Alternatives have been proposed (Sanabria and Killeen, 2007) that claim to avoid some flaws and problems in null hypothesis significance testing, mainly in the social sciences. However, this remains an open debate; no alternative has been widely accepted, and it is beyond the scope of this paper to adjudicate on the available alternatives.

Let $S$ be the number of sequences that have passed the test. Then an estimate for the proportion that have passed is $\widehat{p} = S/m$. Six-sigma limits for the acceptable proportion under the null hypotheses $H_0$ can be given by

$$p \pm 3\sqrt{\frac{p(1-p)}{m}}, \tag{1}$$

where $p = 1 - \alpha$ and $m$ is the sample size. Notice that we are using the normal approximation for the binomial distribution and that such an approximation works better under certain requirements. Namely, the probability of success $p$ must be close to 0.5, and the sample size $m$ must be large enough such that the products $p \times n$ and $(1-p) \times n$ are both above 5 (Johnson et al., 1992). Thus, $n$ must be large enough to assure the products defined earlier are above 5; otherwise, the Normal approximation may not be appropriate. For all those tests presented in Table 1 except tests #12 and #13 (*Random Excursions Test* and *Random Excursions Variant Test*), the limits for $p = 0.99$ and $m = 1,000$, are given by

$$0.990 \pm 3\sqrt{\frac{0.99(0.01)}{1,000}} = 0.990 \pm 0.009 \Rightarrow [0.981; 0.999].$$

That is, the proportion estimates must be above 0.981 for a given generator. For tests #12 and #13, these limits depend on the number of sequences tested and are given as output.

### 3.2 Performance evaluation based on statistical process control

There is a great deal of difficulty practically deciding whether a given generator is accepted based on a $p$-value covered (or not covered) by the intervals from Eq. (1). Theoretically, a good pseudo-random number generator should produce $p$-values in any given interval from 0 to 1 proportionate to the size of the interval. That is, it should produce $p$-values in the range of 0 to $\alpha$ around $100\%$ of the time. Indeed, as we shall see in Section 4, some generators fail some of the tests but are

Table 1: NIST tests and settings

| index | test | category | settings |
|---|---|---|---|
| 1 | Frequency Test (monobit) | non-parametric | n/a |
| 2 | Frequency Test Within a Block | parametric | block length = 128 |
| 3 | Cumulative Sums Test | non-parametric | n/a |
| 4 | Run Test | non-parametric | n/a |
| 5 | Test for the Longest Run of Ones | non-parametric | n/a |
| 6 | Binary Matrix Rank Test | non-parametric | n/a |
| 7 | Discrete Fourier Transform Test | non-parametric | n/a |
| 8 | Non-overlapping Templates Match | parametric | template length = 9 |
| 9 | Overlapping Templates Matching Test | parametric | template length = 9 |
| 10 | Maurer's Universal Statistical Test | parametric | block length = 7 and number of initialization steps = 1280 |
| 11 | Approximate Entropy Test | parametric | block length = 10 |
| 12 | Random Excursions Test | non-parametric | n/a |
| 13 | Random Excursions Variant Test | non-parametric | n/a |
| 14 | Serial Test | parametric | block length = 16 |
| 15 | Linear Complexity Test | parametric | block length = 500 |

**step 1:** *Select a pseudo-random number generator.*
**step 2:** *Generate 1,000 sequences of size 1,000,000 from the selected generator.*

$$\left.\begin{array}{rl} S_1 & = (0,0,1,1,0,1,1,1,0,\dots,1) \\ S_2 & = (1,1,0,1,0,0,0,1,0,\dots,0) \\ S_3 & = (0,1,0,1,0,1,0,0,1,\dots,0) \\ \vdots & \\ S_{1000} & = (1,0,0,1,1,0,1,0,1,\dots,1) \end{array}\right\} \text{sequences of size 1,000,000}$$

**step 3:** *Submit each sequence to the fifteen statistical tests.*
*Each test returns one or several p-values as follows.*

$$\begin{array}{c|cccc|} \text{tests} \rightarrow & 1 & 2 & \dots & 15 \\ S_1 & P_{1,1} & P_{1,2} & \dots & P_{1,15} \\ S_2 & P_{2,1} & P_{2,2} & \dots & P_{2,15} \\ S_3 & P_{3,1} & P_{3,2} & \dots & P_{3,15} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ S_{1000} & P_{1000,1} & P_{1000,2} & \dots & P_{1000,15} \end{array}$$

**step 4:** *Print the p-values, which indicate the type-I error probability.*

Figure 1: NIST Strategy of Testing

nevertheless still considered to be good. We here propose a set of decision rules based on the binomial distribution (Johnson et al., 1992) to take into account purely random fluctuations in the *estimated* proportion of time that $p$-values are deemed too low by a given test.

It is clear that modeling test results in terms of $n$ Bernoulli experiments with probability of success $p$ and the number of positive test results the random variable $X$ yields a binomial distribution with $X \sim \text{BIN}(n,p)$. Table 2 presents the cumulative probabilities for $X$ for $p = 0.99$ and the number of tests in the NIST suite $n = 15$.

It is apparent that we could accept a generator that fails for at most two tests yet still consider it a good generator for which the type-I error is as small as $P(X \leq 13) = 0.00963$. The problem is still not completely solved, though. Some NIST tests provide output of more than one $p$-value, such as the *Non-overlapping Templates Match*, the *Random Excursions Test*, and the *Random Excursions Variant Test*. Thus, we must apply an analogous principle to the one already described here to decide if the generator fails in these tests. Note that these tests output 148, 8, and 18 $p$-values, respectively. The cumulative probabilities for $X$ are presented in Table 3.

Based on Table 3, it is acceptable that a sequence fails the *Non-overlapping Templates Match* at most 4 out of the 148 words tested before considered to fail the entire test, with $P(X \leq 144) = 0.06222$. Likewise, a sequence could fail the *Random Excursion Test* in at most 1 out of 8 excursions tested, with $P(X \leq 7) = 0.07726$. Similarly, for the *Random Excursion Variant Test*, it is acceptable that 2 out of 18 excursions fail, with $P(X \leq 16) = 0.01376$.

## 4 Experimental Results

In this section, we present results obtained by using the proposed method to analyze some of the well-known pseudo-random number generators (Rukhin et al., 2001). The six different generators presented in Table 4 are tested and ranked according to quality. All

Table 2: The binomial distribution for the NIST suite

| success ($x$) | fail ($n-x$) | proportion | $P(X=x)$ | $P(X \le x)$ |
|---|---|---|---|---|
| 15 | 0 | 1.00000 | 0.86006 | 1.00000 |
| 14 | 1 | 0.93333 | 0.13031 | 0.13994 |
| **13** | **2** | **0.86667** | **0.00921** | **0.00963** |
| 12 | 3 | 0.80000 | 0.00040 | 0.00042 |
| 11 | 4 | 0.73333 | 0.00001 | 0.00001 |

Table 3: The binomial distribution for multiple $p$-value tests

| test | success ($x$) | fail ($n-x$) | proportion | $P(X=x)$ | $P(X \le x)$ |
|---|---|---|---|---|---|
| #8: | 148 | 0 | 1.00000 | 0.22595 | 1.00000 |
| non-overlapping | 147 | 1 | 0.99324 | 0.33778 | 0.77405 |
| template match | 146 | 2 | 0.98649 | 0.25078 | 0.43627 |
| | 145 | 3 | 0.97973 | 0.12328 | 0.18549 |
| | **144** | **4** | **0.97297** | **0.04514** | **0.06222** |
| | 143 | 5 | 0.96622 | 0.01313 | 0.01708 |
| | 142 | 6 | 0.95946 | 0.00316 | 0.00395 |
| | 141 | 7 | 0.95270 | 0.00065 | 0.00078 |
| | 140 | 8 | 0.94595 | 0.00012 | 0.00014 |
| #12: | 8 | 0 | 1.00000 | 0.92275 | 1.00000 |
| random excursions | **7** | **1** | **0.87500** | **0.07457** | **0.07726** |
| | 6 | 2 | 0.75000 | 0.00264 | 0.00269 |
| | 5 | 3 | 0.62500 | 0.00005 | 0.00005 |
| | 4 | 4 | 0.50000 | 0.00000 | 0.00000 |
| #13: | 18 | 0 | 1.00000 | 0.83451 | 1.00000 |
| random excursions | 17 | 1 | 0.94444 | 0.15173 | 0.16549 |
| variant | **16** | **2** | **0.88889** | **0.01303** | **0.01376** |
| | 15 | 3 | 0.83333 | 0.00070 | 0.00073 |
| | 14 | 4 | 0.77778 | 0.00003 | 0.00003 |

tests were performed in accordance with the NIST test suite, using settings from Table 1 and Fig.1.

### 4.1 *Analysis of algorithms classified as superior*

Two well-known pseudo-random number generators of superior quality were tested. The first is the BBS method proposed by Blum et al. (1986). The second is the ANSI X9.17, which has been approved as the Federal Information Processing Standard and usually is employed for the generation of random keys. Both algorithms were submitted as part of the NIST tests.

The results are presented in Fig. 2. Note that there is only one rejection for test #8. Because the proposed evaluation procedure as described in the previous section allows up to *four* rejections for this test, we cannot reject that the generated sequences are random. With respect to the latter generator, there are two rejections for tests #12 and #13, but the evaluation procedure allows up to *one* and *two* rejections for these tests, respectively. Thus, we cannot reject that the generated sequences are random.

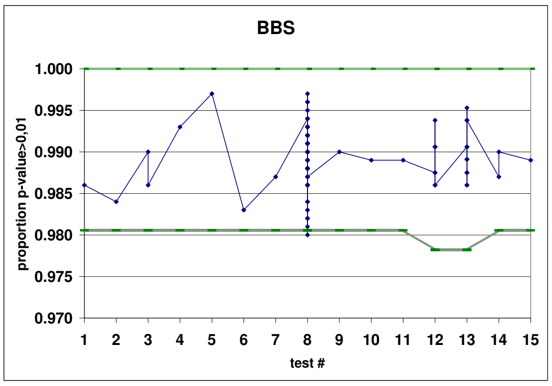
### 4.2 *Analysis of algorithms classified as average*

Here, we also analyze two algorithms. The first one is the linear congruential (LCG) algorithm, which uses a given seed z0 to generate a pseudo-random sequence from $z_{i+1} = a * z_i \mod (2^{31} - 1)$; in the experiments, we used $z_0 = 23482349$. The other generator in this category was the quadratic congruential II (QCG-II) algorithm, according to which the sequence is built from $x_{i+1} = 2x_i^2 + 3x_i + 1 \mod 2^{512}$, with $x_0 =$ 7844506a9456c564b8b8538e0cc15aff46c95 e69600f084f06 57c2401b3c244734b62ea9bb95be4923b9b7e84eeaf1a224 894ef0328d44bc3eb3e983644da3f5.

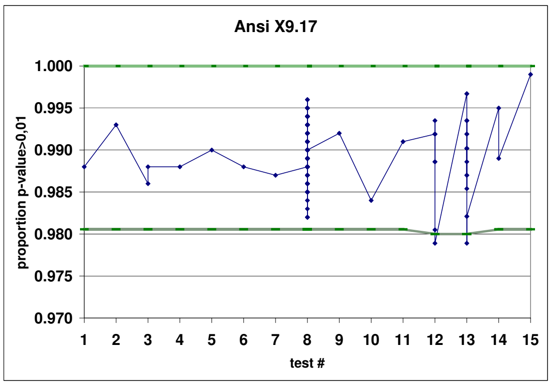
After running the NIST tests and observing the results for LCG presented in Fig. 3-a, we note two rejections for tests #8 and #12. Because our evaluation procedures accept up to four and one rejections, respectively, we do not reject that the generated sequences by LCG are random. For QCG-II, Fig. 3-b also shows two rejections for tests #7 and #8. Again, the randomness of the generated sequences is not rejected. We remark that perhaps the proposed test should have rejected these algorithms, but we note that the proposed method provides a method to make decisions concerning the quality of the PRNGs based on samples of sequences. In other words, we reduce the cost of testing the generators at the cost of increased type-I and type-II errors, which may be an issue here.

### 4.3 *Analysis of algorithms classified as low*

In the XOR algorithm, we select a sequence of bits $x_1, x_2, \ldots, x_{127}$ and build a pseudo-random sequence from $x_i = x_{i-1} \bigoplus x_{i-127}$, for $i \geq 128$. In addition, in the modular exponentiation (ModExp) algorithm, a pseudo-random sequence is generated as follows. A prime sequence $p$ of 512 bits, a base $g$, and a 160-bit seed $y_0$ are chosen. The sequence is generated from $x_1 = g^{y_0} \mod p$ and $x_{i+1} = g^{y_i} \mod p$, for $i \geq 1$, so that the sequence of $y_i$ is

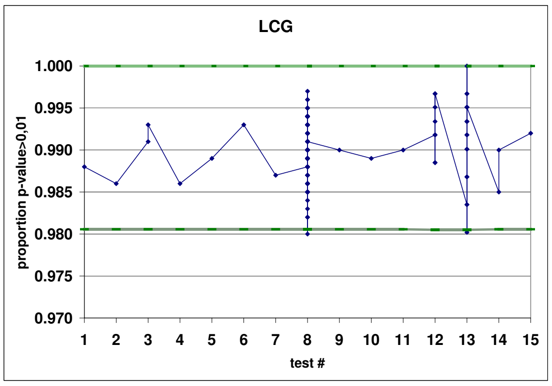


(a) BBS algorithm

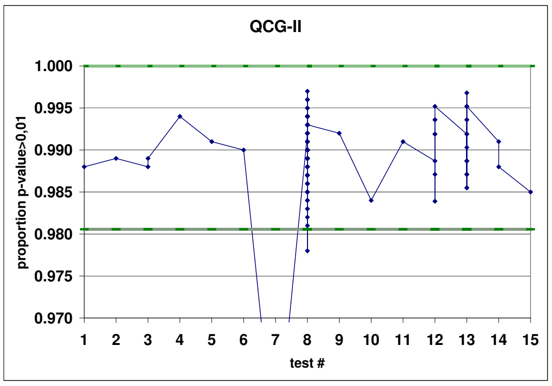


(b) ANSI X 9.17 algorithm

Figure 2: Results for two superior quality random number generators

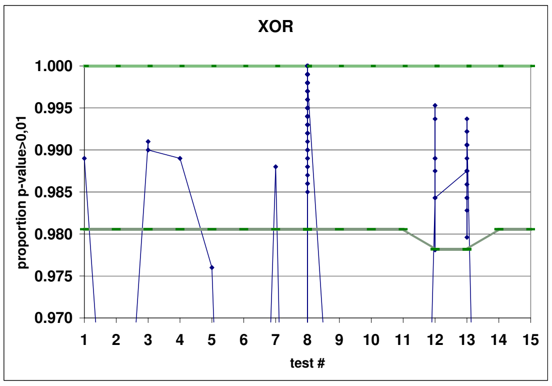


(a) LCG algorithm

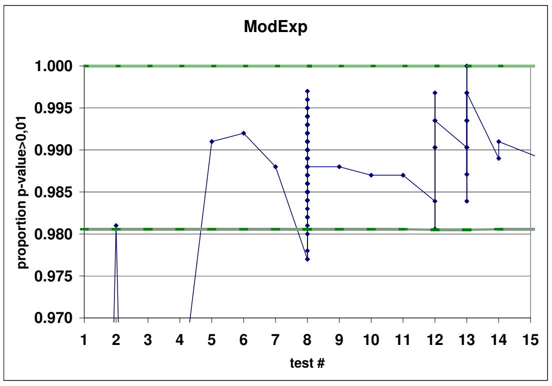


(b) QCG-II algorithm

Figure 3: Results for two average quality random number generators



(a) XOR algorithm



(b) ModExp algorithm

Figure 4: Results for two low quality random number generators

Table 4: Generators analyzed

| index | algorithm | quality† |
|---|---|---|
| 1 | Blum-Blum-Shub | superior |
| 2 | ANSI X9.17 | superior |
| 3 | Quadratic Congruential II | average |
| 4 | Linear Congruential | average |
| 5 | XOR | low |
| 6 | Exponential Modular | low |

†As described in <http://www.lavarnd.org/what/nist-test.html>.

the 160 lower-order bits from $x_i$ (this study, $y_0 =$ 7AB36982CE1ADF832019CDFEB2393CABDF0214EC, $p =$ 987b6a6bf2c56a97291c445409920032499f9ee7ad1283 01b5d0254aa1a9633fdbd378d40149f1e23a13849f3d45992 f5c4c 6b7104099bc301f6005f9d8115e1, and $g =$ 3844506a 9456c564b8b8538e0cc15aff46c95e69600f084f0657c2401b3 c244734 b62ea 9bb95be4923b9b7e84eeaf1a224894ef032 8d44bc3eb3e983644da3f5).

The results obtained after performing the NIST tests on the generated sequences by these two algorithms are presented in Fig. 4. We see that there are many more failures generated by the tests than are acceptable. Therefore, we conclude that the sequences generated by these two algorithms are not random so that these algorithms are not acceptable at the given confidence level.

## 5 Conclusions and Final Remarks

In this article, we address the quality assessment of pseudo-random number generators using an original method based on the binomial distribution. The method builds on a well-known test suite from NIST and may be useful for the analysis of new pseudo-random number generators. As an illustration, the new method was applied to known generators to confirm their quality in generating pseudo-random sequences.

Future work in this area might include the analysis of different generators and/or the same generators under different type-I errors. The development of simulation procedures to estimate type-II errors might be helpful as well.

## References

Ali, M. M., 2007. Synthesis of the $\beta$-distribution as an aid to stochastic global optimization. Computational Statistics & Data Analysis 52 (1), 133–149.

Blum, L., Blum, M., Shub, M., 1986. A simple unpredictable pseudo random number generator. SIAM Journal on Computing 15, 364–383.

Diks, C., Hommes, C., Panchenko, V., van der Weide, R., 2008. E&F chaos: A user friendly software package for nonlinear economic dynamics. Computational Economics 32(1–2), 221–244.

Geyer, C. L., Williamson, P. P., 2004. Detecting fraud in data sets using Benfords law. Communications in Statistics - Simulation and Computation 33 (1), 229–246.

Glaser, D. N., 1999. The controversy of significance testing: Misconceptions and alternatives. American Journal of Critical Care 8 (5), 291–296.

Information Security Institute, 2008. Crypt-X. Technical report, Queensland University of Technology, Australia, accessed Feb. 2008. URL `http://www.isi.qut.edu.au/resources/cryptx/tests.php`

Johnson, N. L., Kotz, S., Kemp, A. W., 1992. Univariate Discrete Distributions. Wiley, New York, NY.

Kimbrough, S. O., Murphy, F. H., 2009. Learning to collude tacitly on production levels by oligopolistic agents. Computational Economics 33(1), 4778.

Knuth, D. E., 1998. The Art of Computer Programming, 3rd Edition. Vol. 2: Seminumerical Algorithms. Addison-Wesley Series in Computer Science and Information. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Kolsrud, D. O., 2008. Stochastic *Ceteris Paribus* simulations. Computational Economics 31(1), 21–43.

Kundu, D., Gupta, R. D., 2007. A convenient way of generating gamma random variables using generalized exponential distribution. Computational Statistics & Data Analysis 51 (6), 2796–2802.

Lima, E. J. A., Tabak, B. M., 2009. Tests of random walk: A comparison of bootstrap approaches. Computational Economics 34(4), 365–382.

Marks, N. B., 2007. Kolmogorov-Smirnov test statistic and critical values for the Erlang-3 and Erlang-4 distributions. Journal of Applied Statistics 34 (8), 899–906.

Marsaglia, G., 1995. The Marsaglia random number CDROM with the DIEHARD battery of test of randomness. Technical report, Center for Information Security and Cryptography, HKU, accessed Mar. 2010. URL `http://i.cs.hku.hk/~diehard/cdrom/`

Maschek, M. K., 2010. Intelligent mutation rate control in an economic application of genetic algorithms. Computational Economics 35(1), 25–49.

Rukhin, A., Soto, J., Nechvatal, J., Smid, M., Barker, E., Leigh, S., Levenson, M., Vangel, M., Banks, D., Heckert, A., Dray, J., Vo, S., 2001. A statistical test suite for random and pseudorandom number generators for cryptographic applications. Technical report, NIST Special Publication 800-22 (with revisions dated May 15, 2001), accessed Mar. 2010. URL `http://csrc.nist.gov/groups/ST/toolkit/rng/documents/SP800-22b.pdf`

Sanabria, F., Killeen, P. R., 2007. Better statistics for better decisions: Rejecting null hypotheses statistical tests in favor of replication statistics. Psychology in the Schools 44 (5), 471–481.

Wichmann, B. A., Hill, I. D., 2006. Generating good pseudo-random numbers. Computational Statistics & Data Analysis 51 (3), 1614–1622.