

Analysis of true and pseudorandom number generators

Jonathan T. Cotton

ABSTRACT

The goal of this project is to perform various numerical analysis tests for randomness on true as well as pseudorandom number generators. The soundness of both true and pseudorandom number generators is the key issue being investigated in this report. The DIEHARD random number test suite's statistical tests were chosen for measuring the quality of the various random number generators, on the basis that these tests were suitable, straightforward, accurate and simple to implement. The implementation and interpretation of each test is defined in this report.

INTRODUCTION

A random number generator is a device which generates a sequence of numbers that lack any uniformity and, as such, appear to be completely random. Pseudorandom number generators are widely used in fields such as gambling, statistical sampling, computer simulation and cryptography to name just a few. The approaches used for pseudorandom number generators typically fall short of being truly random; however they are often statistically sound enough for practical use. Hardware based true random number generator approaches are capable of generating truly random numbers, which are generated based on observations of the chaos of some physical phenomenon in the universe. Through various analysis techniques, the suitability of both pseudorandom number generators as well as true random number generators can be proven to be sound. It can be seen that while true random number generators are preferable, well designed pseudorandom number generators can produce reasonably acceptable results as well.

A pseudorandom number generator (PRNG) is a deterministic algorithm to generate a sequence of numbers with little or no discernible pattern in the numbers, except for broad statistical properties (Black). Current PRNGs vary vastly in quality. There is a small number of low quality PRNGs used for many of the random number generation tasks typically found in every day computation. These PRNGs include those included in various operating systems and programming languages. Two such PRNGs include the `/dev/random` PRNG that is built into the LINUX operating system as well as the random number functions implemented by the JavaScript programming language. There is also a number of high quality PRNGs such as the Yarrow PRNG that is built into the Mac OS X operating system. These are the three PRNGs that are to be tested in this report for statistical soundness. The quality of PRNGs is an extremely important issue and these PRNGs must be irrevocably proven to be "random enough" before any reliance can be placed in them. As John von Neumann once joked, "Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin." (Neumann 36-38)

In addition to pseudorandom number generators, true random number generators exist as well. True random number generators will be referred to as TRNGs in this report. A

TRNG is a random number generator that is perfect in every way. The next value produced is completely uncorrelated with all previous values. These TRNGs typically use a chaotic source to power the generation of very high quality random numbers. A chaotic source is a physical system that is dominated by chaos (LavaRnd Source) The LavaRnd TRNG as well as the HotBits TRNG produce true, chaotic, random data. These two TRNGs will be analyzed in this report along side of the three chosen PRNGs.

OBJECTIVE

The objective of this paper is to compare the quality levels of various PRNGs and TRNGs. These results are to be used to show the need for a high quality PRNG when a suitable TRNG cannot be utilized. By using numerical analysis techniques on the various RNGs, a concrete comparison can be formed of the various RNGs being tested. The two low quality PRNGs obviously fail this small battery of tests, while the higher quality Yarrow PRNG, as well as the two TRNGs all pass these tests.

In order to test the five selected random number generators, George Marsaglia's DIEHARD battery of statistical tests has been chosen. The research behind the DIEHARD test suite was sponsored by the National Science Foundation. There are fifteen tests included, of which three were chosen to be utilized. These three tests were the overlapping 5-permutation test, the bit stream test and the squeeze test. Each of these tests is explained in detail in the Development section of this paper.

DEVELOPMENT

TRUE RANDOM NUMBER GENERATORS

One example of a chaotic source is acquiring digital snapshot of a physical chaotic process such as the one used by the LavaRnd Cryptographically Sound Random Number Generator.

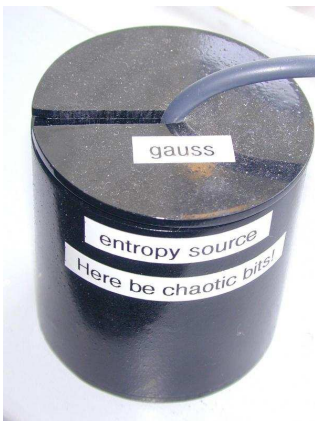


Fig. 1 - LavaRnd LavaCan. A chaotic source capture device.

LavaRnd turns real world physical chaotic events into random numbers in 3 stages (LavaRnd Process):

1. *Digitization of a chaotic source:*

A digital snapshot of a physical chaotic process is obtained. Any chaotic source that is sensitive to measurement errors can be used. The Heisenberg Uncertainty Principle suggests that you cannot measure, with perfection, any chaotic source. If the chaotic source you choose is driven by quantum events then recreating the chaotic source is impossible. Moreover, chaos will render any simulation useless as a means to predict future conditions.

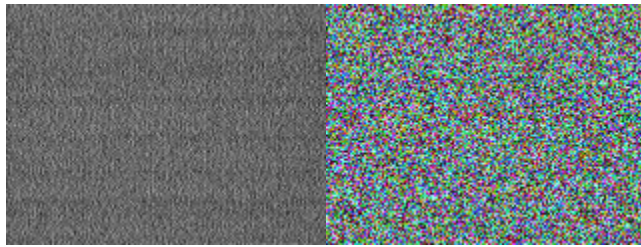


Fig. 2 - Digitized data of the background noise energy levels in a space completely devoid of light.

2. *Digital Blender™:*

The digital snapshot containing both structured data and chaotic noise is run through a Digital Blender Algorithm. The combination of n different SHA-1 cryptographic hash operations running in parallel, and n different xor-rotate and fold operations on data containing some chaotic noise destroys the structured data portion of the digital snapshot and produces uniformly distributed random data.

3. *Presentation:*

The uniformly distributed random data is collected into a pool and used only once to produce random values in the form required by the application.

Another example of a chaotic source can be found by timing successive pairs of radioactive decays detected by a Geiger-Müller tube interfaced to a computer such as the TRNG used by the HotBits service at <http://www.fourmilab.ch/hotbits/>.



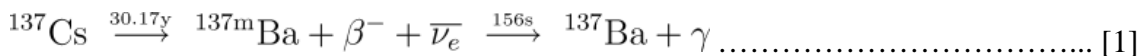
Fig. 3 – HotBits Geiger-Müller tube interfaced to a computer's serial I/O port

The HotBits service uses a Cæsium-137 check source for radiation generation. Check sources of this kind are readily available, can be shipped through the mail, and require no license in most reasonable jurisdictions; people in the United States can order them on-line from United Nuclear (Walker).



Fig. 4 - An Oxford Nuclear 5 microcurie Cæsium-137 check source

The source generates radiation through the beta decay of Cæsium-137 and the subsequent rapid gamma emission by the resulting metastable Barium-137 nucleus. The decomposition of the Cæsium-137 atom into the Barium-137 nucleus is described in Equation[1] (Walker).



PSEUDORANDOM NUMBER GENERATORS

Linux is one of the most popular open source project operating system projects. The Linux PRNG is part of the kernel of all Linux distributions and is based on generating randomness from entropy of operating system events. The output of this generator is used

for almost every security protocol, including TLS/SSL key generation, choosing TCP sequence numbers, and file system and email encryption. The Linux kernel is an open source project developed in the last 15 years by group of developers led by Linus Torvalds. The kernel is the common element in all various Linux distributions, on all types of devices. The output of the Linux PRNG can be used by internal kernel functionalities which use random bits, and by calls to its application programming interface. The Linux kernel uses random data for various purposes, such as generating random identifiers, computing TCP sequence numbers, producing passwords, and generating SSL private keys. This data can be accessed by reading from /dev/random. The data used in this test was collected by running the command

The JavaScript random number function, like most computer-driven RNGs, is driven by use of a complex algorithm (seeded by the computer's clock) that gives the appearance of randomness. The built in JavaScript function Math.random was used to generate the data that was tested by the DIEHARD test suite. The data was generated using the interface provided by <http://www.randomizer.org> for generation of random numbers for research purposes.

TEST RESULTS

The tests chosen from the DIEHARD test suite return a p-value, which should be uniform on $[0,1)$ if the input file contains truly independent random bits. Those p-values are obtained by $p=F(X)$, where F is the assumed distribution of the sample random variable X. The assumed F is an asymptotic approximation, for which the fit will be worst in the tails. Occasional p-values near 0 or 1, such as .0012 or .9983 are not to be unexpected. When a bit stream fails the tests strongly, you will get p values of 0 or 1 to six or more places of accuracy. The case in which $p < 0.025$ or $p > 0.975$ does not conclusively mean that the RNG has failed the test. Such p values are an occasional occurrence in the results that the DIEHARD tests produce. This is true even with high quality TRNGs.

Overlapping 5-permutation Test:

The first test that was run on these RNGs was the overlapping 5-permutation test. This test looks at a sequence of one million 32-bit random integers. Each set of five consecutive integers can be in one of 120 states, for the $5!$ possible orderings of five numbers. Thus the 5th, 6th, 7th...numbers each provide a state. As many thousands of state transitions are observed, cumulative counts are made of the number of occurrences of each state. Then the quadratic form in the weak inverse of the 120×120 covariance matrix yields a test equivalent to the likelihood ratio test that the 120 cell counts came from the specified (asymptotically) normal distribution with the specified 120×120 covariance matrix (with rank 99). This version uses 1,000,000 integers, twice. The results of this test on each of the five RNGs are listed in Table 1 below.

TABLE 1. RESULTS OF THE OVERLAPPING 5-PERMUNTATION TEST ON THE RNGS

Sample #	Chi Square	Degrees of freedom	p-value
<i>HotBits</i>			
1	79.227	99	0.071687
2	80.156	99	0.082712
<i>Random.org</i>			
1	78.826	99	0.067257
2	86.017	99	0.179084
<i>LINUX PRNG</i>			
1	68.568	99	0.008543
2	99.744	99	0.539879
<i>JavaScript PRNG</i>			
1		99	
2		99	

Bit stream Test:

The second numerical analysis test was the bit stream test. The random numbers generated are stored as binary numbers in a file. The file is then viewed as a stream of bits by the test. Call these bits b_1, b_2, \dots, b_n . Consider an alphabet with two "letters", 0 and 1 and think of the stream of bits as a succession of 20-letter overlapping "words". Thus the first word is $b_1b_2\dots b_{20}$, the second is $b_2b_3\dots b_{21}$, and so on. The bit stream test counts the number of missing 20-letter (20-bit) words in a string of 2^{21} overlapping 20-letter words. There are 2^{20} possible 20 letter words. For a truly random string of $2^{21}+19$ bits, the number of missing words j should be (very close to) normally distributed with mean 141,909 and sigma 428. Thus $(j-141909)/428$ should be a standard normal variate (z score) that leads to a uniform [0,1) p value. The test is repeated twenty times by the DIEHARD test suite. For a sample with 20 bits/word, 2097152 words, 20 bit streams, the number of missing words should average 141909.33 with sigma=428.00, and chi-square with 2500 degrees of freedom for a sample size of 2560000.. The results of this test can be found in Table 2.

TABLE 2. RESULTS OF THE BIT STREAM TEST ON THE RNGS

Bit stream	Chi-square	Equivalent normal	p-value
<i>HotBits</i>			
1	2403.34	-1.367	0.085814
2	2354.18	-2.062	0.019595
<i>Random.org</i>			
1	2538.33	0.542	0.706104
2	2453.41	-0.659	0.254995
<i>LINUX PRNG</i>			
1	2420.60	-1.123	0.130754
2	2445.00	-0.778	0.218321
<i>JavaScript PRNG</i>			
1			
2			

Squeeze Test:

The third and final test that was run on these RNGs was the squeeze test. In the squeeze test, random integers are floated to get uniforms on [0,1). Starting with $k=2^{31}=2147483647$, the test finds j , the number of iterations necessary to reduce k to 1, using the reduction $k=\text{ceiling}(k*U)$, with U provided by floating integers from the file being tested. Such j values are found 100,000 times, then counts for the number of times j was $\leq 6,7,\dots,47,\geq 48$ are used to provide a chi-square test for cell frequencies. The results of the squeeze test are detailed in Table 3 below.

TABLE 3. RESULTS OF THE SQUEEZE TEST ON THE RNGS

<i>HotBits</i>					
-0.1	-1.2	-0.4	-0.4	0.2	-0.1
-0.1	-0.5	1	-0.6	2.9	1
-0.3	0.8	0	0.8	-0.8	-0.5
0	1.7	0.2	-0.1	-1.3	-0.5
-1.3	0.1	-1.3	-1.5	-0.6	-1
-0.4	-0.4	-0.2	-0.1	0.3	-1.9
0	-0.7	-1.2	0.4	-0.6	-1
0.8					
Chi-square with 42 degrees of freedom: 35.665					
z-score = -0.691					
p-value = 0.255775					
<i>Random.org</i>					
3.4	-0.7	-1.3	0.2	-1.5	-1.9
1.6	-0.8	-0.6	-0.4	-0.1	0
1.3	-0.9	0.1	0.2	-0.5	-0.9
1.2	-0.6	0.6	0.8	-0.4	0.5
0.1	1	-1.7	1.1	0.2	-0.8
-0.5	-0.5	0.6	1	-0.2	0.2
0.3	-0.7	-0.8	-0.7	-1.3	0
-0.1					
Chi-square with 42 degrees of freedom: 41.475					
z-score = -0.057					
p-value = 0.506061					
<i>LINUX PRNG</i>					
-0.8	0.9	0.3	0.3	-0.5	-0.6
-0.2	-0.6	-0.8	0.1	0.4	0.1
0.2	-0.1	0.2	-0.4	0.9	1.7
-0.9	-1	0.6	0.3	-2.3	0.3
-0.6	-0.7	1.5	0.6	-0.2	1.2
0.9	-0.5	0	0.1	0.9	0.6
0.5	-1	0.1	-0.7	-1.3	-1
-1.1					
Chi-square with 42 degrees of freedom: 27.887					
z-score = -1.540					
p-value = 0.046348					

REFERENCES

- Neumann, John von . "Various techniques used in connection with random digits."
Applied Mathematics Series 12(1951): 36-38.
- Noll, Landon Curt; Cooper, Simon; Pleasant, Mel. "true random source."
LavaRnd.org. LavaRnd Source. 19 Mar 2007
<http://www.lavarnd.org/faq/true_random_src.html>.
- Noll, Landon Curt; Cooper, Simon; Pleasant, Mel. " LavaRnd: A Cryptographically
Sound Random Number Generator" LavaRnd.org. LavaRnd Process.
19 Mar 2007 < <http://lavarnd.org/what/how-it-works.html>>.
- Black, Paul E.. "pseudo-random number generator." Dictionary of Algorithms and Data
Structures . 16 October 2006. U.S. National Institute of Standards and
Technology. 21 Mar 2007
<<http://www.nist.gov/dads/HTML/pseudorandomNumberGen.html>>.
- Walker, John. "HotBits: Genuine Random Numbers." HotBits: Genuine Random
Numbers. September 2006. Fourmilab. 21 Mar 2007
<<http://www.fourmilab.ch/hotbits/>>.