Quantum Random Number Generation with Applications to Monte Carlo Simulations and Machine Learning Techniques in Finance

Anqi Zhou[1], Suhang Li[1], Yushi Zhao[1],
Professor: Dr. Rupak Chatterjee[1], and
Adivisor: Carl M. Dukatz [2]

[1]Department of Financial Engineering, Stevens Institute of Technology
[2]Accenture

February 2017

1 Introduction

1.1 Random Number Generators (RNG)

One of the main advantages of the quantum world is the ability for quantum systems to produce true random numbers as opposed to the pseudo and quasi random numbers generated by classical deterministic systems. The use of Monte Carlo numerical simulations for stochastic processes is widespread in physics, mathematics, engineering, and computational finance. These simulations depend upon the availability of large amounts (tens of millions) of random numbers. Random numbers are also crucial to cryptography as well as various machine learning algorithms. In this project, we analyze a true random number generator (TRNG) created at the Center for Distributed Quantum Computing at the Stevens Institute of Technology.

1.2 Statistical Tests

Random number generation can be divided into methods that are either deterministic based on some mathematical algorithm or non-deterministic coming from an actual physical process (Haahr, 1999). Algorithmic methods are subdivided into pseudo random number generators (PRNG) or quasi random number generators (QRNG). PRNGs are the most commonly used method for simulation and cryptography. PNRGs all start with an initial value called a seed. Starting with the exact same seed, one is able to generate identical sequences of random numbers, hence the name pseudo (as opposed to being truly random) and the categorization of deterministic. Even though PRNGs are important because of their speed in number generation, almost all of them fail basic non-random pattern detection tests (Sen et al., 2006) such as

-*Strong long range correlations* : Autocorrelation is a major problem for PRNGs where successive random numbers in a given sequence are correlated to ones that came earlier in the sequence. True random numbers should have no memory of previously generated numbers and be independent and identically distributed.

-*Periodicity* : All PRNGs will eventually repeat themselves after a certain number of steps in a sequence. Even though may approximate the length of the period (and these can be relatively large), certain seeds may lead to shorter than expected periods.

-*Discrepancy*: PRNGS are known to not have low-discrepancy in the sense that they may tend to 'bunch up' around certain points rather than be smoothly distributed in the desired range. Another way to say this is that the relative distances between where certain points occur are distributed more frequently from those in a truly random sequence. This can also lead to poor dimensional distribution properties for multi-dimensional PRNGs.

Quasi random number generators (QRNG) try to solve the discrepancy problem of PRNGs by artificially creating low-discrepancy sequences that are more equidistributed according to the measure of the space (Joy et al., 1996). This leads to better (lower) discrepancy properties than PRNGs for shorter sequences of random numbers. This is very useful for rare event generation (occurring in the tails of distributions) and high dimensional simulation. Yet, by their nature QRNGs have high autocorrelation and they provide a purely deterministic sub-random sequence of numbers that have to be used with care.

Various statistical tests have been designed to determine the quality of a set of random numbers. The most accepted test is the National Institute of Standards and Technology (NIST) Statistics Test Suite (Soto, 1999) (Rukhin et al., 2010). This test suite contains 15 packages:

- *Frequency Test* : This test is to determine whether the number of ones and zeros in a sequence are approximately the same.

- *Block Frequency Test* : This test is to determine whether the frequency of ones in an M-bit block is approximately M/2.

- *Cumulative sum test:* :This test is to find the maximal excursion (from zero) of the random walk defined by the cumulative sum of adjusted (-1, +1) digits in the sequence.

- *Runs Test* : This test is to determine whether the oscillation between such zeros and ones is too fast or too slow.

- *Longest Runs Test* : This test is to determine whether the length of the longest run of ones within the tested sequence is consistent with the length of the longest run of ones.

- *Binary Matrix Rank Test* : This test is to check for linear dependence among fixed length sub strings of the original sequence.

- *Fourier Transform (Spectral) Test* : This test is to detect periodic features (i.e., repetitive patterns that are near each other) in the tested sequence that would indicate a deviation from the assumption of randomness.

- *Non-overlapping Template Matching Test* : This test is to detect generators that produce too many occurrences of a given non-periodic (aperiodic) pattern.

- *Overlapping Template Matching Test* :This test is to search for a specific m-bit pattern.

-*Maurer's "Universal Statistical" Test* :This test is to detect the number of bits between matching patterns, whether or not the sequence can be significantly compressed without loss of information.

- *Linear Complexity Test* :This test is to determine whether or not the sequence is complex enough to be considered random—the length of a linear feedback shift register (LFSR).

- *Serial Test* :This test is to determine whether the number of occurrences of the 2m m-bit overlapping patterns is approximately the same as would be expected for a random sequence.

-*Approximate Entropy Test* :This test is to compare the frequency of overlapping blocks of two consecutive/adjacent lengths (m and m+1) against the expected result for a random sequence.

- *Random Excursions Test* :This test is to find the number of cycles having exactly K visits in a cumulative sum random walk.

- *Random Excursions Variant Test* :This test is to detect deviations from the expected number of visits to various states in the random walk.

The Dieharder Test Suite is a very popular set of statistical tests (Brown et al., 2013). It includes all the legacy Diehard tests (Marsaglia, 2008) and part of NIST tests and other tests:

- *Diehard Birthdays Test* : Each test determines the number of matching intervals from 512 "birthdays" (by default) drawn on a 24-bit "year" (by default).

- *Diehard Overlapping 5-Permutations Test* :This test looks at a sequence of one million 32-bit random integers–counts the number of each of the 120 (5!) different configurations and finds the variance from the expected plot of these values.

- *Diehard 32x32 Binary Rank Test* :Ranks are found for $40,000$ such random matrices and a chi-square test is performed on counts for ranks $32, 31, 30$ and <=29.

- *Diehard 6x8 Binary Rank Test* : Ranks are found for $100,000$ random matrices, and a chi-square test is performed on counts for ranks 6,5 and <=4.

- *Diehard Bitstream Test* : This test counts the number of missing 20-letter (20-bit) words in a string of 2^{21} overlapping 20-letter words which should be normally distributed.

- *Diehard Overlapping Pairs Sparse Occupance (OPSO)* : Similar to bitstream test, but this test uses 10-bit letters and 2-letter words and takes different bits from a word each time.

- *Diehard Overlapping Quadruples Sparce Occupancy (OQSO) Test* : Again similar, but this test uses 5-bit letters and 4 letter words, and p-value is calculated based on variance from a mean.

- *Diehard DNA Test* : Again this is similar to the last three, but it uses 5-bit letters and 4-letter words.

- *Diehard Count the 1s (stream) Test* : This test uses a sample of $256,000$, and determines how many times 8-bit letter and 5-letter words appear on a stream of bytes and calculates variance.

- *Diehard Count the 1s Test (byte)* :This test uses a sample of $256,000$, and determines how many times 8-bit letter and 5-letter words appear for specific bytes and calculates variance.

- *Diehard Parking Lot Test* : the distribution of attempts to randomly park a square car of length 1 on a $100x100$ parking lot without crashing.

- *Diehard Minimum Distance (2d Circle) Test* :This test chooses $n = 8,000$ random points in a square and then finds d, which is the distance between all pairs of points.

- *Diehard 3d Sphere (Minimum Distance) Test* :This test chooses 4000 random points in a cube of edge 1000. At each point, center a sphere large enough to reach the next closest point.

- *Diehard Squeeze Test* : Starting with $k = 2^{31}$, it multiples by floating point values on $[0, 1)$ until $k = 1$.

- *Diehard Sums Test* : This test adds 100 consecutive integers, incrementing by one element at a time and storing each of the sums, which should have a normal distribution.

- *Diehard Runs Test* : This test iterates through $10,000$ values and determines consecutive 'runs' where elements are increasing in value or decreasing in value consecutively.

- *Diehard Craps Test* :This test plays $200,000$ games of craps, finds the number of wins and the number of throws necessary to end each game.

- *Marsaglia and Tsang GCD Test* : 10^7 samples (default) of unit rands u, v are generated and two statistics are generated: greatest common divisor (GCD) (w) and the number of steps of Euclid's Method required to find it(k). The number of times each value for k in the range 0 to 41 (with counts greater than this range lumped in with the endpoints) and the frequency of occurrence of each GCD w table.

- *STS Monobit Test* :This test counts the 1 bits in a long string of random units.

- *STS Runs Test* :This test counts the total number of 0 runs plus total number of 1 runs across a sample of bits.

- *STS Serial Test (Generalized)* :This test accumulates the frequencies of overlapping n-tuples of bits drawn from a source of random integers.

- *RGB Bit Distribution Test* :This test accumulates the frequencies of all n-tuples of bits in a list of random integers and compares the distribution thus generated with the theoretical (binomial) histogram, forming chi-square and the associated p-value.

- *RGB Generalized Minimum Distance Test* : This test is the generalized minimum distance test, based on the paper of M, which utilizes correction terms that are essential in order for the test not to fail for large numbers of trials.

- *RGB Permutations Test* : A non-overlapping test that simply counts order permutations of random numbers, pulled out n at a time.

-*RGB Lagged Sum Test* : It adds up uniform deviates sampled from the rng, skipping lag samples in between each rand used.

- *RGB Kolmogorov-Smirnov Test Test* :It extracts n independent bytes from each of k consecutive words. Increment indexed counters in each of n tables. (Total of $256xn$ counters.)

Thus, we plan to use both NIST and Dieharder to test our quantum TRNG.

1.3 Competitors

Currently, there are several companies and labs that focus on quantum random number generation. *ID Quantique* is the first company to develop a commercial quantum random number generator in 2001. The company used an optical quantum process as a random source to generate quantum random numbers. Their method can produce a high bit rate of 4 to 16 Mbits/sec of truly random bits. ID Quantique also produces a quantum key distribution protocol based on their random numbers (Quantique, 2010). The Australia National University(ANU) has a project that generates random numbers by using a laser. The hardware is constantly generating random bits at a rate of 5.7Gbits/s. This is one of the fastest random number generation methods in the world. Random numbers generated from ANU can pass both NIST and

Dieharder tests. As ANU provides these random numbers to the public, we plan to apply both test suites to their random numbers to conform their claim.

1.4 Quantum RNG in Stevens

The Stevens quantum random number generator uses single photons where the output photon modes are truly random an lie within a certain predetermined integer range. Our work is to obtain true random numbers from these modes. We will transform these quantum random sequences to specific distributions as uniform or Gaussian. Both NIST and Diehard tests will be performed for statistical analysis. Furthermore, the speed and efficiency of our TRNG will be analyzed. The Stevens TRNG will also be extended to a multidimensional and correlated random number generation.

1.5 Workflow

See Figure 1.

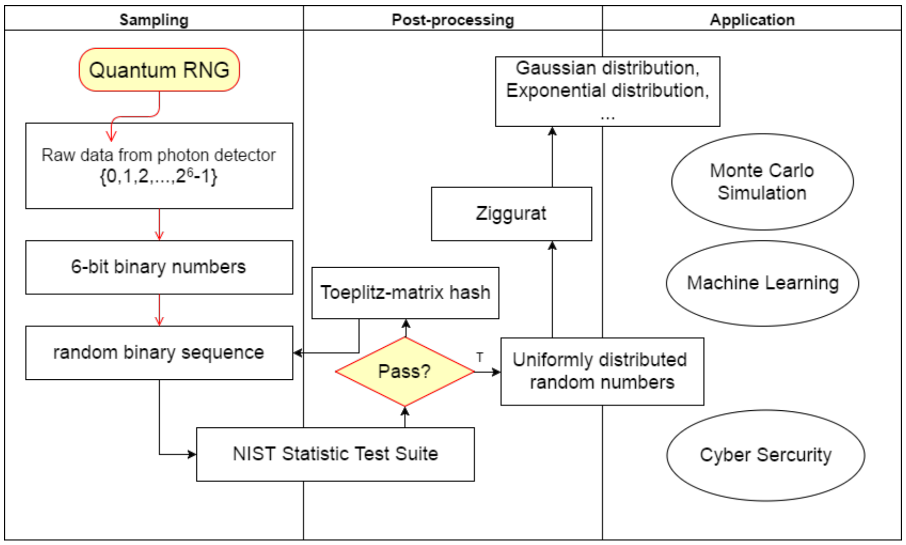

Figure 1: Workflow

2 Methodology

2.1 Binary Sequence Representation

In the very first version, each photon carries 6 bits of quantum random number (in 64 modes). Those bits from photons can be concatenated into a random binary sequence. Then, we can split the sequence into segments with longer bits in order to get a desired precision. For example, if we want numbers from $[0, 1)$ to accurate to 0.001, then we should take 10 bits ($1\frac{2}{3}$ photons) as one integer and divide it by 1024. For applications' needs, post-processing may also be conducted. For example, we plan to use the Ziggurat algorithm to transform distributions.

3 Applications

3.1 Monte Carlo Simulation

Monte Carlo simulations are used for both derivative valuation and risk analysis. The current regulatory framework of Basel III, CVA, CCAR, all demand massive simulations of a bank's financial positions feeding an endless need for random numbers for such multidimensional simulations. We plan to apply the Stevens TRNG to price and risk analyze barrier type path dependent options (El Babsiri and Noel, 1998).

3.2 Machine Learning

The Particle Swarm Optimization (PSO) method was first intended for simulating social behavior (Kennedy, 1997) but also gained interest from multiple other fields (Poli et al., 2007). This algorithm includes a critical random number component. In the following paper (Ding and Tan, 2014), a variety of PRNGs and QRNGs were tested. We plan to add our Stevens TRNG to this test to show that it will outperform both PRNGs and QRNGs.

3.3 Cyber Security

We plan to implement the BB84 (Bennett and Brassard, 1984) standard protocol for Quantum Key Distribution (QKD). Along with this implementation, our TRNG can also be used in a privacy amplification process (Bennett et al., 1995) which is a set of methodologies of distilling highly secret shared information from a larger body of information that is only partially secret.

4 Recent Results

Pseudo random numbers from EXCEL, Python and R are generated using Stevens Hanlon Lab computers. True random numbers from ANU are downloaded from ANU Quantum Random Numbers Server on which stated the generator speed.
For the NIST Statistics Test Suite, the initial result presented at table 1 is derived from 500 random number samples and each sample has 800000 random bits. For the Dieharder test suite, we use its default setting.

Table 1: NIST Statistics Test Suite Results

	Pseudo RNG			True RNG
	Excel RANDBETWEEN	Python numpy.randint	R Sample	ANU Quantum RNG
Frequency	Pass	Pass	Pass	Pass
Block Frequency	Pass	Pass	Pass	Pass
Cumulative Sums	Pass	Pass	Pass	Pass
Runs	Pass	Pass	Pass	Pass
Longest Run	Pass	Pass	Pass	Pass
Rank	Pass	Pass	Pass	Pass
FFT	Fail	Pass	Pass	Pass
Non Overlapping Template	Pass	Pass	Pass	Pass
Overlapping Template	Pass	Pass	Pass	Pass
Universal	Pass	Pass	Pass	Pass
Approximate Entropy	Pass	Pass	Pass	Pass
Random Excursions	Pass	Pass	Pass	Pass
Random Excursions Variant	Pass	Pass	Pass	Pass
Serial	Pass	Pass	Pass	Pass
Linear Complexity	Pass	Pass	Pass	Pass
Generator Speed	Incomparably Low	4.13Gbits/s	0.98Gbits/s	5.7Gbits/s

Table 2: Dieharder Statistics Test Suite Results

	Pseudo RNG			True RNG
	Excel RANDBETWEEN	Python numpy.randint	R Sample	ANU Quantum RNG
Diehard Birthdays Test	Pass	Pass	Pass	Pass
Diehard OPERM5 Test	Pass	Pass	Pass	Pass
Diehard 32x32 Binary Rank Test	Pass	Pass	Pass	Pass
Diehard 6x8 Binary Rank Test	Pass	Pass	Pass	Pass
Diehard Bitstream Test	Pass	Pass	Pass	Pass
Diehard OPSO	Pass	Pass	Pass	Pass
Diehard OQSO Test	Pass	Pass	Pass	Pass
Diehard DNA Test	Pass	Pass	Pass	Pass
Diehard Count the 1s (stream) Test	Pass	Pass	Pass	Pass
Diehard Count the 1s Test (byte)	Pass	Pass	Pass	Pass
Diehard Parking Lot Test	Pass	Pass	Pass	Pass
Diehard Minimum Distance (2d Circle) Test	Pass	Pass	Pass	Pass
Diehard 3d Sphere (Minimum Distance) Test	Pass	Pass	Pass	Pass
Diehard Squeeze Test	Pass	Pass	Pass	Pass
Diehard Sums Test	Pass	Pass	Pass	Pass
Diehard Runs Test	Pass	Pass	Pass	Pass
Diehard Craps Test	Pass	Pass	Pass	Pass
Marsaglia and Tsang GCD Test	Fail	Fail	Fail	Pass
STS Monobit Test	Pass	Pass	Pass	Pass
STS Runs Test	Pass	Pass	Pass	Pass
STS Serial Test (Generalized)	Pass	Pass	Pass	Pass
RGB Bit Distribution Test	Pass	Pass	Pass	Pass
RGB Generalized Minimum Distance Test	Pass	Pass	Pass	Pass
RGB Permutations Test	Pass	Pass	Pass	Pass
RGB Lagged Sum Test	Fail	Fail	Fail	Pass
RGB Kolmogorov-Smirnov Test Test	Pass	Pass	Pass	Pass
Generator Speed	Incomparably Low	4.13Gbits/s	0.98Gbits/s	5.7Gbits/s

References

Bennett, C. H. and G. Brassard (1984). Quantum cryptography: Public key distribution and con tos5.

Bennett, C. H., G. Brassard, C. Crépeau, and U. M. Maurer (1995). Generalized privacy amplification. *IEEE Transactions on Information Theory 41*(6), 1915–1923.

Brown, R. G., D. Eddelbuettel, and D. Bauer (2013). Dieharder: A random number test suite. *Open Source software library, under development*.

Ding, K. and Y. Tan (2014). Comparison of random number generators in particle swarm optimization algorithm. In *Evolutionary Computation (CEC), 2014 IEEE Congress on*, pp. 2664–2671. IEEE.

El Babsiri, M. and G. Noel (1998). Simulating path-dependent options: A new approach. *The Journal of Derivatives 6*(2), 65–83.

Haahr, M. (1999). Introduction to randomness and random numbers. *Random. org, June*.

Joy, C., P. P. Boyle, and K. S. Tan (1996). Quasi-monte carlo methods in numerical finance. *Management Science 42*(6), 926–938.

Kennedy, J. (1997). The particle swarm: social adaptation of knowledge. In *Evolutionary Computation, 1997., IEEE International Conference on*, pp. 303–308. IEEE.

Marsaglia, G. (2008). The marsaglia random number cdrom including the diehard battery of tests of randomness, 1995. *URL http://www. stat. fsu. edu/pub/diehard*.

Poli, R., J. Kennedy, and T. Blackwell (2007). Particle swarm optimization. *Swarm intelligence 1*(1), 33–57.

Quantique, I. (2010). Random number generation using quantum physics. *IDQ White Paper*.

Rukhin, A., J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, et al. (2010). Nist special publication 800-22rev1a: A statistical test suite for the validation of random number generators and pseudo random number generators for cryptographic applications, april 2010. *Avaiable from: http://csrc. nist. gov/groups/ST/toolkit/rng/documents/SP800-22rev1a. pdf*.

Sen, S. K., T. Samanta, and A. Reese (2006). Quasi-versus pseudo-random generators: Discrepancy, complexity and integration-error based comparison. *International Journal of Innovative Computing, Information and Control 2*(3), 621–651.

Soto, J. (1999). Statistical testing of random number generators. In *Proceedings of the 22nd National Information Systems Security Conference*, Volume 10, pp. 12. NIST Gaithersburg, MD.