TRINITY COLLEGE DUBLIN
Management Science and Information Systems Studies
Project Report

THE DISTRIBUTED SYSTEMS GROUP,
Computer Science Department, TCD

Random Number Generators: An Evaluation and Comparison of Random.org and Some Commonly Used Generators

April 2005

Prepared by: Charmaine Kenny ***Supervisor: Krzysztof Mosurski***

ABSTRACT

The aim of this project is to research statistical tests that detect non-randomness in a true random number generator (http://www.random.org). An industry-standard suite of tests was chosen to verify the complete randomness, from a statistical viewpoint, of the numbers generated by Random.org. It is envisaged that the test suite will be ran on Random.org numbers daily with results being displayed on the website. The performance of the output of Random.org is compared to other commonly used pseudorandom number generators and true random number generators. The paper also addresses a number of unresolved issues that need further exploration.

PREFACE

An ideal random number generator is a fiction
Schindler and Killman [4]

The client of this project is the Distributed Systems Group (DCG), a research group in the Department of Computer Science in Trinity College Dublin. They operate a public true random number service which generates randomness based on atmospheric radio noise.

The aim of the project is essentially to verify that the output of its random number service is completely random and to recommend a suite of tests that can be ran daily on the output of Random.org. Additionally the aim is to compare Random.org to some commonly used random number generators – both pseudo and true.

The work carried out in this project can undoubtedly be developed upon. Although a substantial amount was achieved given the time constraints, there are some pertinent issues raised throughout the report that need further consideration. The current opportunities to explore in this area seem endless. Random numbers are being increasingly used in all aspects of life and have become a staple in many fields, not just statistics. It seems that while a host of research has been done it is far from complete.

Acknowledgements

I would like to express my sincerest thanks to all those who helped in the realisation of this project. There are three people in particular that deserve a special mention;

I would like to thank Dr. Mads Haahr, the client contact and the builder of Random.org, for his insight and willingness to help. His prompt responses to my queries were greatly appreciated.

Secondly, I would like to thank Niall Ó Tuathail, a fellow MSISS student, for his patience in helping me run the tests. It was an arduous task!

And finally my thanks to Dr.Kris Mosurski, my supervisor, for his help and guidance during the course of my project. I am very appreciative of his insightful discussions and dedication to ironing out even the smallest of problems.

THE DISTRIBUTED SYSTEMS GROUP
Random Number Generators – An evaluation and comparison of Random.org and some commonly used generators

April 2005

TABLE OF CONTENTS

APPENDICES

1. INTRODUCTION

This chapter introduces the client and the project. It defines the terms of reference and also gives a brief overview of the remaining chapters to guide the reader.

1.1 The Client

Established in 1981, the Distributed Systems Group (DSG) [1] is both the longest standing and largest research group in the Department of Computer Science at Trinity College Dublin. DSG conducts basic and applied research into all aspects of distributed computing extending from the theoretical foundations underpinning the field to system engineering issues. They currently focus on four key overlapping topics: middleware, ubiquitous computing, mobile computing, and software engineering.

1.2 Project Background

The Distributed Systems Group is operating a public true random number service (http://www.random.org) which uses radio noise as a source of randomness to generate true random numbers. A technical description of how the noise is converted into ones and zeroes is detailed in Appendix M. The numbers are currently made available via a web server. Since it went online in October 1998, Random.org has served nearly 50 billion random bits to a variety of users. Its popularity is on the rise and at the moment the web site receives approximately 50,000 hits per day. These numbers are available free of charge on the website and are also available in many different formats. The client wishes to verify that the output of its random number service can be considered completely random.

Louise Foley, a former MSISS student, conducted a somewhat similar project in April 2001 to this for her final year project entitled “Analysis of an Online Random Number Generator”. [2] The set of statistical tests she recommended to test Random.org on a daily basis have not been implemented. John Walker’s ENT Program [3] is still used to test the numbers generated. Since the time of the former report the client has made adjustments to the random number generator. This project builds and expands upon the ideas in the former study.

1.3 Terms of Reference

Following are the terms of reference as confirmed at the interim reporting[1] stage in November, 2004:

- To conduct a literature review of the applications of random number generators and to contrast the use of true random number generators and pseudo random number generators;
- To research statistical tests that detect non-randomness, review statistical test suites available, and then propose a set of statistical tests to be applied to the numbers generated by Random.org;
- To consider other random number generators as possible comparative studies and compare Random.org to a selection of these:

[1] See Appendix B for complete interim report.

- To define, prioritise and spec the efficiency of the implementation of the proposed suite.

Note: Contrary to the original project guidelines[2] the client did not require that the author code the tests for integration with the Random.org server.
The difficulties encountered while fulfilling these terms of reference are detailed in Appendix C.

1.4 Report Summary

Chapter 2 is a summary of the findings and recommendations of the project
Chapter 3 is a literature review of the broad area of random number generation. It looks at the definition of a random sequence, the applications of random numbers, types of generators, the statistical testing approach as well as the statistical test suites that are currently being used.
Chapter 4 raises some issues that need to be considered before recommending a statistical test suite that detects non-randomness in sequences of numbers. It also addresses the methodology of the random number testing.
Chapter 5 discusses the performance of the output of Random.org and some comparative generators having been subjected to the recommended test suite.
Chapter 6 deals with some open issues in the area of testing that need further consideration.

For the reader who is looking for something specific an index has been compiled which may be useful for direction to the relevant section (Appendix S).

[2] The original project guidelines are given in Appendix A.

1. CONCLUSIONS AND RECOMMENDATIONS

This chapter summarises the key findings and recommendations of the project. The chapters that follow give a more in-depth discussion and analysis of these.

Conclusions

Random numbers are crucial ingredients for a whole range of applications – including cryptography, simulation, gaming, sampling and aesthetics - and their consumption is rapidly increasing. (Section 3.2)

There are essentially two types of random number generators – true and pseudo. The fundamental difference between the two types is that true random number generators sample a source of entropy whereas pseudorandom number generators instead use a deterministic algorithm to generate numbers. In comparing true random number generators and pseudorandom number generators each have their merits and limitations. Completely random true random number generators have no periodicities, no predictability, no dependencies, a high level of security and are conceptually nice. Nevertheless they are slow, inefficient and costly, cumbersome to install, their sequences are not reproducible and they are subject to manipulation. The reverse is true of pseudorandom number generators. Which is better depends largely on the application. (Section 3.3)

For all but trivial applications the quality of the underlying random number generator is critical and it is therefore essential that the generator be tested.

Statistical hypothesis testing is the most widely used method to verify that the numbers produced by a generator purported to be a random number generator are in fact random and it is the method employed here also. Other methods of testing include graphical examinations of the numbers or transformed numbers, using the numbers as input to a known problem and also application based testing. (Section 2.4)

The main difficulty in testing a multi-purpose generator like Random.org is that what should be deemed sufficiently random depends on the application. Furthermore, no amount of statistical testing can guarantee that a random number generator is indeed completely random. Rather, if a generator passes a wide variety of tests then confidence in its randomness increases. (Section 2.4)

Having considered many statistical test suites reported in the literature the National Institute of Standards and Technology (NIST) test suite is recommended, mainly for the pragmatic reason that it is recognised as the industry standard. The NIST test suite consists of 15 tests.

Random.org passes the NIST test suite; its pass rates and uniformity checks are in line with what NIST considers sufficient to be deemed random. Random.org can now be recognised as passing the industry standard suite of tests. (Chapter 5 & Appendix N)

Two comparative pseudorandom number generators, the Microsoft Excel RNG and the Minitab RNG are also subjected to the NIST test suite. Minitab passes the suite but Excel fails on a lack of uniformity in two of the fifteen tests.

Two comparative true random number generators, Hotbits and Randomnumbers.info were subjected to the suite of tests once. Hotbits passed all tests while Randomnumbers.info failed two tests. Although this is suggestive of non-randomness a conclusion should not be made about the generator before conducting further investigation.

While NIST is the industry standard suite of statistical tests the author has reservations about how satisfactory the suite is. Some of the statistical approaches suggested by NIST are questionable. Firstly, NIST recommends that each test be ran multiple times. The classical approach to hypothesis testing, however, is to run the test only once. The recommendation by NIST to conduct a chi-square test on the p-values is also open to discussion. Furthermore concern arises over the number of mistakes found in the NIST manual as well as the lack of clarity in how the test suite should be implemented. There is also a deficiency in the explanation of the rationale for the particular tests in the suite. (Section 4.2, Appendix J)

Recommendations

It is recommended that Random.org numbers be subjected to the NIST test suite on a regular basis. A prioritisation of the tests is made but should the client choose to implement a subset of the suite then Random.org cannot be deemed to pass the industry standard suite of tests. For this reason it is suggested that the entire suite be implemented. (Sections 3.5 & 4.1.4)

There are some unresolved issues that need consideration. Certainly, the NIST suite needs to be evaluated. The power of the tests in the suite especially needs to be investigated. The independence and coverage of tests also deserves some attention. Additionally how to interpret the results needs to be clarified. (Section 6)

Although the NIST test suite is recommended to be used to test Random.org on a regular basis this recommendation should be subject to review in time. New statistical tests will be continuously developed to gather evidence that random number generators are of high quality. While the NIST suite is currently considered the most comprehensive in the literature it will undoubtedly be replaced by a new test suite that hopefully will address the unresolved issues identified here satisfactorily.

It is suggested that Random.org run the 15 tests in the NIST suite daily but that, instead of carrying out the debatable statistical analysis that NIST recommend, adopt a more flexible approach. The p-values should be extracted and essentially looked at. The suggested graphics in Appendix O should be constructed to gain further insight.

While it is not practical for the client to tailor tests to the specific applications of Random.org numbers it is recommended that the user should perform application-based testing where possible in addition to the NIST statistical testing that is carried out at the Random.org end.

3. LITERATURE REVIEW

This chapter reviews the literature and addresses such questions as what is a random number sequence, what are random numbers used for, what kinds of random number generators are there, how are random numbers tested, and what test suites are available in the literature.

3.1 Definition of a Random Number Sequence

It may be taken for granted that any attempt at defining disorder in a formal way will lead to a contradiction. This does not mean that the notion of disorder is contradictory. It is so, however, as soon as I try to formalize it.
~ Hans Freudenthal [39]

Philosophers have discussed many ways to define randomness, but few are relevant for the purposes here[3]. The reason why the definition of random is considered here is because it is impossible to design meaningful tests without specifying what is meant by random and non-random.

Knuth puts forward the notion that in a sense, there is no such thing as a random number; for example, is 1 a random number? Rather what is spoken of is “a sequence of independent random numbers. [5] And so, it is a random number sequence, as opposed to a random number, that should be defined.

A random sequence could be interpreted as the result of the flips of an unbiased “fair” coin with sides that are labelled “0” and “1”, with each flip having the probability of exactly 0.5 of producing a “0” or “1”. Furthermore, the flips are independent of each other; the results of any previous coin flips does not affect future coin flips”. There should be complete forward (and backward) unpredictability. The unbiased “fair” coin is thus the perfect random number generator, since the “0” and “1” values will be randomly distributed. All elements of the sequence are generated independently of each other, and the value of the next element in the sequence cannot be predicted, regardless of how many elements have already been produced [6].

Obviously, the use of unbiased coins for applications that require many numbers is impractical. Nonetheless, the hypothetical idea of such a generator is useful in defining a random sequence. This, of course, is ignoring the possibility of some non-randomness being introduced in the way in which the coins are flipped. This point highlights the difficulty in adequately defining a random sequence. For the purposes of defining a random sequence here, however, it is a useful analogy and captures the two important properties; independence and equally likely.

[3] Interestingly, it is only in recent times that the concept of something random existing has been accepted. Numerous philosophers have in the past argued the case for total causality known as 'hard determinism'. This theory dictates that we have no free will and are merely acting under the illusion of possessing it. Every movement of every particle is the direct result of something before it - as is every thought we have. That is to say that everything is in effect predetermined, in that it is the only thing that could happen given the events that preceded it. [4]

Encompassing these ideas in a formal mathematical statement the following working definition is proposed:

> Let X_n be a sequence of random variables, where $n = 1,2,3...$
> X_n is a binary random variable, meaning that the possible values of X_n are 0 and 1.
> If $P(X_n = 1 \mid \text{all } o\text{thers}) = 0.5$
> Or equivalently, the joint distribution of all the sequences is 0.5^N at every point of the sample N space, of which there are 2^N points.

Should a sequence satisfy this definition then it can be considered random.

Random numbers are used in many different forms from particular distributions, for example, integer, binary, uniform, etc. depending on what their intended use is. The only numbers that are of concern here are binary numbers because once the binary numbers produced by a random number generator are deemed to be random, then it is true that their transformation to any interval can also be deemed to be random. This is provided that the appropriate transformation is carried out correctly. Essentially there must be 2^N numbers in the sequence to carry out any transformation so that the transformed numbers satisfy the independence and equally likely properties.

3.2 Applications of Random Numbers

"Chance governs all"
~ Milton

Random numbers are crucial ingredients for a whole range of usages, including cryptography, simulation, gaming, sampling, decision making and aesthetics. How random numbers are used in these fields is discussed in the subsections that follow (3.2.1-3.2.6). This is an overview of the main applications of random numbers; it is by no means a definitive list.

3.2.1 Cryptography

The story of cryptography begins when Julius Caesar sent messages to his faithful acquaintances. He did not trust the messengers and so he replaced every A in the message by a D, every B by an E, and so on through the alphabet. Only someone who knew the 'shift by 3' rule could decipher his messages. [7] Cryptography is the art or science of turning meaningful sequences into apparently random noise in such a way that only a key-holder can recover the original data. [8] Today, the rules have obviously evolved to become a lot more sophisticated, especially with the rapid evolution of computing power. The objective, however, remains largely the same; to preclude an adversary from gaining advantage through knowing what the message sent says. What is needed to achieve this are sequences that are hard to predict unless the mechanism generating them is known. And so, at the heart of all cryptographic systems is the generation of secret, unguessable numbers – random numbers. Not only is cryptography a tool used to protect national secrets and strategies like in Caesar's time but its use has crossed over into many

different areas including the securing of electronic commerce around the world and the protecting of private communication over the internet, to name but a couple.

3.2.2 Simulation

Simulation is the re-creation, albeit in a simplified manner, of a complex phenomena, environment, or experience, providing the user with the opportunity for some new level of understanding. When a computer is used to simulate natural phenomena, random numbers are required to make things realistic. Simulation covers many applied disciplines from the study of nuclear physics where particles are subject to random collisions to operations research where people come into, say, an airport at random intervals. [5] Increasingly sophisticated simulation studies are being performed that require more and more random numbers and whose results are more sensitive to the quality of the underlying random number generator.

3.2.3 Gaming

Rolling dice, shuffling decks of cards, spinning roulette wheels, etc are fascinating pastimes for just about everybody. [5] Randomness is central to games of chance and vital to the gaming industry. With the widespread adoption of online gaming, an e-industry worth billions of euro, this application is becoming an increasing consumer of random numbers.

3.2.4 Sampling

It is often impractical to examine all possible cases, but a random sample provides insight as to what constitutes “typical” behaviour. [5] Sampling with random numbers gives each member of the population an equal chance of being chosen, avoiding the problem of bias. Random numbers are used for sample selection by researchers in many fields, both in the academic and working world.

3.2.5 Aesthetics

The use of random numbers in music, art and poetry is becoming increasingly popular. Some are attracted to the idea, for example, of music that cannot be predicted and therefore become interested in the use of random numbers for music production. Random numbers have given some artists a method by which they can distinguish themselves. [9,10]

3.2.6 Applications of Random.org Numbers

Having briefly discussed the broad applications of random numbers above it is now interesting to turn to Random.org to see what its users are using numbers for. There is a page on Random.org called “Who is using Random.org?” that lists some of the things that people use its random numbers for. The list is compiled based on people emailing the client and is therefore not a definitive list of the applications of Random.org’s numbers. The uses fall under many of the application categories outlined in Sections 3.2.1-3.2.5 above. Specific examples of what Random.org users do with the random numbers include:

- A Danish TV station TV2 runs an online backgammon server for which they get more than 300,000 dice rolls per day - this is probably the biggest consumer of numbers from Random.org (Gaming)
- An American band called Technician uses numbers from Random.org to generate unique covers for the band's CDs (Aesthetics)
- Many of the users cited use the numbers for choosing winners of a draw (Sampling)
- One company uses the numbers for choosing employees at random for drug screening (Sampling)

The fact that Random.org random numbers are used for a variety of applications poses problems in the testing of the generator. This is further discussed in 3.4.1.

3.2.7 Concluding Remarks

Random numbers are an important building block in applications across various fields of work and play. The consumption or random numbers is undoubtedly increasing rapidly as is evident from the preceding discussions. The applications range from the critical to the trivial and in the former case great care has to be taken to choose the right type of random number generator as well a random number generator that generates numbers that are sufficiently random. Both of these issues will be discussed in Sections 3.3 and X respectively.

3.3 Types of Random Number Generators

There are two basic types of generators to produce random sequences – true (or physical) random number generators (TRNGs) and pseudorandom number generators (PRNGs). The essential difference between the two types is that TRNGs sample a source of entropy whereas PRNGs instead use a deterministic algorithm to generate random numbers. A brief overview of TRNGs and PRNGs are detailed below.

3.3.1 True RNGs

Nothing is random, only uncertain.
~ Gail Gasram

A true random number generator requires a naturally occurring source of randomness, i.e. entropy, to generate random numbers. It samples this source of entropy and processes it through a computer to produce a sequence of random numbers. True RNGs refer to physical RNGs and should not be taken as completely random because they are called “true”. True random numbers are by definition entirely unpredictable. The use of a distillation process is generally needed to overcome any weaknesses in the entropy source that results in the production of non-random numbers (e.g. the occurrence of long strings of zeroes or ones). The entropy source typically consists of some physical quantity, such as atmospheric noise from a radio (e.g. Random.org), the elapsed time between the emission of particles during radioactive decay (e.g. Hotbits [11]), the thermal noise from a semiconductor diode or the frequency instability of a free running oscillator. There are more novel sources of entropy used to generate random numbers such as the photographs of lavalamps (e.g. lavarand [12]).

3.3.2 Pseudo-RNGs

Anyone who attempts to generate random numbers by deterministic means is, of course, living in a state of sin.
~ John von Neumann

If von Neumann's declaration is to be believed then statisticians and cryptographers are living in a state of sin because PRNGs are more widely used than TRNGs. Indeed the volume of literature on PRNGs far exceeds that on TRNGs. Pseudo random numbers are not strictly random; their generation does not depend on a source of entropy. Rather they are deterministic, meaning that they are computed using a mathematical algorithm. If the algorithm and the seed (i.e. the number that is used to start the generation) are known then the numbers generated are predictable.

The very notion that a deterministic formula could generate a random sequence seems like a contradiction. The main objective with PRNGs is to obtain sequences that behave as if they are random. The output sequences of many PRNGs are statistically indistinguishable from completely random sequences and ironically, PRNGs often appear to be more random than random numbers obtained from TRNGs. [6] By their definition, however, the maximum length of sequences produced by all these algorithms is finite and these sequences are reproducible, and thus can be "random" only in some limited sense. [13] A brief discussion of the various types of PRNGs that are in use is given in Appendix D.

3.3.3 Comparison of RNGs and PRNGs

Both true random number generators and pseudorandom number generators have their advantages and disadvantages. Generally the limitation of one type is the merit of the other. Because of this only the advantages and disadvantages of TRNGs are listed in Table 3.1 (in considering PRNGs reverse the advantages and disadvantages of the TRNG). The table applies to true RNGs that are deemed to be completely random. This analysis perhaps sheds light on the suitability of particular RNGs to particular applications.

True RNGs	
Advantages	**Disadvantages**
No periodicities	Slow and inefficient
No predictability of random numbers based on knowledge of preceding sequences	Cumbersome to install and run
Certainty that there are no dependencies present	random number sequences are not reproducible
High level of security	Costly
Conceptually nice – not based on algorithm	Possibility of manipulation

Table 3.1 Advantages and Disadvantages of TRNGs

It an open question as to whether it is possible in any practical way to distinguish the output of a well designed pseudo-random number generator from a perfectly random source without knowledge of the generator's internal state. Which type is “better” or “more suitable” depends greatly on the application. It is for this reason that the suitability of Random.org is commented on in the context of each broad application as described in Section 3.2.

3.3.4 What about Random.org?

What Random.org is actually used for has already been addressed but what is more to the point is what should Random.org be used for? According to the client Random.org is intended primarily for educative purposes but also aims to be useful for non-critical applications. What follows is a comment on Random.org under the various application headings:

Cryptography Random.org is not very useful for cryptography as a generator that distributes its numbers over the internet. The security of the numbers cannot be guaranteed and the possibility that the numbers be observed by a third party while in transit is very real. However, because the Random.org RNG produces true random numbers the generator could perhaps be used in the generation of cryptographic keys if the numbers were not distributed online (and if noone hacked into the server!). A mixed approach to random number generation is often taken in cryptographic applications, that is, the pseudo RNG is seeded with output from a true RNG. This means that some entropy is introduced and a PRNG is used thereafter.

Simulation Random.org is not very useful for most simulations because it does not produce, in a reasonable amount of time, the vast quantities of numbers that would be needed. Moreover, it cannot reproduce the numbers which is a desirable for simulations. (Generally one input variable is changed to see the effect on the result. If the random numbers are changing for every run of the simulation then it would not be possible to distinguish if a change in the results was due to the random numbers or the change in an input variable) Random.org could however be used for small-scale simulations where the numbers could be saved relatively easily.

Gaming Random.org is useful for gaming and indeed this was its original intended application. Secrecy is not important here, unless of course the numbers generated are not used immediately (in which case there is again the problem of interception of the numbers in transit).

Sampling Random.org is also useful for sampling applications that do not require vast amounts of numbers, for example selecting a random sample from an electoral list for a survey.

3.4 Statistical Testing

A foolish builder builds a house
on sand. Test and verify the randomness
of your random number generator.
~ Casimir Klimasauskas [14]

A key issue in this project is how to decide if a sequence is sufficiently random. It is not satisfactory to declare a sequence random based on its appearance. Knuth [5] exemplifies this: if some randomly chosen person was given a pencil and paper and asked to write down 100 random binary numbers, the chances are very slim that he would produce a satisfactory result. People tend to avoid things that seem non-random, such as long streams of zeroes or long streams of ones (although there is a 50% chance that a number should equal its predecessor). And if the same person was shown a table of completely random numbers, he would quite probably say that they are not random at all; the eye would spot certain apparent regularities. The point of these remarks is that human beings cannot be trusted to judge by themselves whether a sequence is random or not. Some unbiased mechanical tests need to be applied to objectively decide if a sequence is sufficiently random. The phrase "sufficiently random" is used because no amount of testing can prove that a given RNG is flawless. It only improves our confidence to a certain extent [15]. In fact, verifying the randomness of a RNG should really fall under the heading of acceptance testing. Even if the RNG is completely random it will be rejected occasionally. Conversely, even if the RNG is *not* random it will be accepted occasionally.

The theory of statistics provides some quantitative measures for randomness and indeed statistical testing is the traditional and useful approach for testing random number generators, both true and pseudo. A statistical test is formulated to test a specific null hypothesis (H_0). For the purpose of this project, the null hypothesis under test is that the sequence being tested is random by the formal definition in Section 3.1. Associated with this is the alternative hypothesis (H_A) which, again for this project, is that the sequence is departs from the definition. For each applied test, a decision is derived that accepts or rejects the null hypothesis, i.e., whether the generator is (or is not) producing random numbers, based on the sequence that was produced. See Appendix E for more on hypothesis testing.

A single statistical test is not adequate to verify the randomness of a sequence because the sequence could produce various types of non-randomness[4]. However, if a generator passes a wide variety of tests, then confidence in its randomness increases [16]. Compilations of tests are generally referred to as 'suites' or 'batteries' of statistical tests. The statistical test suites are designed to measure the quality of a generator purported to be a random bit generator. While it is impossible to give a mathematical proof that a generator is indeed a RNG, the tests help detect certain kinds of weaknesses that the generator may have. This is accomplished by taking a sample output sequence of the generator and subjecting it to

[4] This is not entirely true. Theoretically the joint distribution of sequences generated could be tested which would just involve one test. However, a very large number of observations would be required. Suppose, for example, that the length of the sequence is 1000 i.e. n=1000. There are 2^{1000} possible sequences and it is necessary to test if these possible sequences are equally likely. Such a test is essentially a chi square test with 2^{1000} categories which is obviously not practical.

various statistical tests and evaluate that the sequence possesses a certain attribute that a truly random sequence would be likely to exhibit.

Each test within the suite tries to detect a different kind of non-randomness. Having said that the tests are not totally exclusive - there will generally be some overlap. Statistical test suites that have been proposed in the literature are considered in Section 3.5. If the sequence is deemed to have failed any one of the statistical tests within the suite, the generator may be rejected as being non-random; alternatively and advisably, the generator may be subjected to further testing. On the other hand, if the sequence passes all of the statistical tests, the generator is concluded to being random from a statistical point of view. This conclusion is, of course, not definite, but rather probabilistic.

Although statistical testing, as described here, is the most widely used method of testing the randomness of a RNG, there are other ways to detect non-randomness in a sequence. One way involves including looking at pictures of random bits on a plane. These have been shown to reveal spatial dependencies which are not clearly detected in the quantitative tests [13]. Perhaps these types of tests complement the quantitative tests rather than being a stand-alone approach. For the people who knows little about statistical tests and hypothesis testing these are a nice tool to display that the numbers are random. Of course this type of exploratory data analysis supports but does not prove the hypothesis that the numbers are random. See Appendix O for kinds of graphics that could be constructed. Another approach to testing is to use the numbers generated by the RNG as input to a known problem. This approach is inherently a little haphazard. Yet another way to test RNGs is application-based testing. Essentially this involves testing how appropriate the generator is for a particular application. This is discussed in Section 6.

Ideally all of these testing procedures, especially application based testing, should be employed when evaluating a RNG. The more testing conducted the more certain that the RNG is actually producing random numbers. This project, however, deals mostly with statistical hypothesis testing.

3.4.1 Choosing what tests to use

The tests simply go fishing. [17] There are an infinite number of possible statistical tests, each assessing the presence or absence of one of very many departures from what would be expected of a completely random sequence which, if detected, would indicate that the sequence is non-random. Because there are so many tests for judging whether a sequence is random or not, no specific finite set of tests is deemed “complete”. Any finite set of tests can miss certain defects and it follows that no amount of testing can guarantee that a generator is foolproof.

Which tests are the good ones? This question has no general answer. It depends on what the RNG is to be used for. The very fact that the need for random numbers arises in many different applications as discussed in Section 3.2 creates difficulty in trying to assess and evaluate the usefulness of a particular generator. Choosing a “good quality” RNG for all applications may not be trivial [13]. In any case no set of statistical tests can absolutely certify a generator as appropriate for usage in a particular application, nevermind all applications.

Having said that the RNGs that are placed in general purpose software packages must be tested without knowing the millions of things they will be used for. The general purpose RNGs should be subjected to a wide variety of tests of different natures, able to detect different types of dependence or other "simple" structures likely to be harmful. [17] In the same way, Random.org must be tested without knowing the millions of things it will be used for.

3.5 Review of test suites

This section is a catalogue of test suite sources (in no particular order). Appendix X shows the tests contained within each test suite.

3.5.1 Knuth

Donald Knuth's book "The Art of Computer Programming, Volume 2 (1st ed. 1969)" [5] is the most-quoted reference for the statistical testing of RNGs in the literature. Known to be a random number guru, his was the de facto standard set of tests for a long time but, although still a required background, is now somewhat outdated. For example he fails to mention cryptographic applications – this was perhaps a sign of the times when cryptography was not near as important as it is today. His tests are now seen to be quite mild, allowing known "bad" generators to pass the tests. Of course, what constitutes a "bad" generator depends on the application.

3.5.2 Diehard

Marsaglia comments that "in spite of the ease with which (statistical) tests of RNGs may be created, there are surprisingly few reported in the literature. The same simple, easily passed tests are reported again and again. Such is the power of the printed word." [36] Marsaglia here is alluding to the uptake of Knuth's work as the definitive gospel on the statistical testing of RNGs. With time developments have been made however and as computers become faster, more random numbers are being consumed than ever before. RNGs that once were satisfactory are no longer good enough for sophisticated applications in physics, combinatorics, stochastic geometry, etc. And so, in 1995 Marsaglia introduced a number of more stringent tests which go beyond Knuth's classical methods in order to meet these new challenges. These tests are stringent in the sense that they are more difficult to pass. Unfortunately, this test suite seems not to have been maintained for the last few years [19]. Apparently Marsaglia has retired and it seems that noone has directly taken over his work. Despite this Marsaglia's set of test is still, ten years after its publication, widely regarded as a very comprehensive collection of tests for detecting non-randomness.

3.5.3 Crypt-X

The Crypt-X suite of statistical test was developed by researchers at the Information Security Research Centre at Queensland University of Technology in Australia and is a commercial software package[5]. [20] Crypt-X tests are applied based on the type of algorithm being tested and so is obviously geared towards the testing of pseudo random number generators. Crypt-X supports stream ciphers, block ciphers and keystream generators.

[5] It costs over €350 for an academic user.

3.5.4 National Institute of Standards and Technology (NIST) [2]

Released in 2001, the NIST[6] Statistical Test Suite [6] is a statistical package consisting of 16 tests that were developed to test the randomness of arbitrary long binary sequences produced by either hardware or software based cryptographic random or pseudorandom number generators. The test suite is the result of collaborations between the Computer Security Division and the Statistical Engineering Division at NIST in response to a perceived need for a credible and comprehensive set of tests for binary (not uniform) random number generators. The test suite makes use of both existing algorithms culled from the literature and newly developed tests. NIST is now by and large the standard in the world of RNG testing.

3.5.5 ENT

John Walker's ENT Program [3] is yet another set of statistical tests to test for non-randomness in sequences. It was developed in 1998. Random.org currently uses this test set (as does Hotbits). The ENT program is described as being useful for those evaluating pseudorandom number generators for encryption and statistical sampling applications, compression algorithms, and other applications where the information density of a file is of interest.

3.5.6 Previous MSISS Project

In 2001 Louise Foley proposed a suite of five tests to replace John Walker's ENT Program in testing the output of Random.org. [2] They were recommended specifically to test Random.org numbers. They were chosen on the basis that "they were suitable, straightforward and in accordance with the clients needs".

Note that this is not a definitive list of test suites that exist for testing random number generators though it captures the more popular ones in use.

[6] NIST (National Institute of Standards and Technologoy is a non-regulatory federal agency within the U.S. Commerce Department's Technology Administration. NIST's mission is to develop and promote measurement, standards, and technology to enhance productivity, facilitate trade, and improve the quality of life. http://www.nist.gov/

4. TESTING ISSUES AND METHODOLOGY

This chapter deals firstly with a discussion of some issues stemming from the decision of which tests should be applied to examine Random.org and follows with making a decision and what the methodology involves.

4.1 Testing Issues

This section addresses some of the issues that arise in testing a random number generator.

4.1.1 Same tests for RNGs and PRNGs?

Some of the suites specify that they were developed for pseudo RNGs (Diehard, Crypt-X) while some others suites specify that they were developed for both true and pseudo RNGs (NIST). This raises the issue of whether or not the same tests should be applied to RNGs and PRNGs. This is not an easy question to answer.

It can be argued that if the numbers are random then they pass the tests and it is irrelevant how they were generated. Some argue that the testing a true random generator differs from testing a pseudo-random number generator. In particular, if one knows the design of the generator one can tailor the tests to be appropriate for that design. Certain types of PRNGs are susceptible to certain departures from randomness. PRNGs remain, by definition, constant. True RNGs, however are susceptible to aging that is, the wear and tear of components. [40] This is of concern. It is not implausible, for example, that the numbers produced by a TRNG would gradually drift towards a bias of outputting more ones than zeroes. Because such aging may be gradual it may be difficult to detect for some time.

Here, the former argument is taken on board, that is, if the numbers are random then they pass the tests and how they were generated is irrelevant. Aging should be picked up by the NIST test suite. Of course the more often the tests are run the quicker any sort of drift will be picked up. It is recommended that some graphics be displayed on the website which show the p-values over time (see Appendix O). This will highlight peculiarities in the p-values over time including perhaps the effect of aging.

4.1.2 Test Suites application dependent?

Another point to note from the list of statistical suites is that a lot of the test suites are designed for a particular application. For example, the tests detailed NIST suite were developed to detect non-randomness for cryptographic applications, while Knuth and Marsaglia's suites were developed for simulation applications. And ENT claims to test for both applications. This raises the issue of whether there should be a different set of tests depending on the application?

Since the "Ferrenberg affair"[7] it is known in the RNG user community that statistical tests alone do not suffice to determine the quality of a generator, but also application-based tests are needed. [22]. Indeed

[7] This was a famous case in 1992 where physicists discovered that even "high-quality" random-number generators, which pass a battery of randomness tests, can yield incorrect results under certain circumstances. [21]

some generators that are suitable for some applications are not suitable for others. And this is also true for different uses within a broad application. For example, for some simulations a particular RNG is fine to use but for other simulations may not be at all appropriate. Ideally, the generator should be tested for the intended application. Section 6.2 further discusses application based testing.

The test suite proposed for Random.org cannot be application dependent because as already discussed in Section X Random.org must be tested without knowing the millions of things it will be used for.

4.1.3 Why implement a new suite of tests?

Random.org currently uses the ENT suite to test its output. While there is nothing "wrong" with the ENT statistical tests it seems that more probing test suites have been developed that pick up departure from randomness in a more comprehensive manner. Additionally the ENT suite is not widely used in the industry. This makes for awkward direct comparison with other generators.

4.1.4 Which suite to use?

In the evaluation of Random.org two of the aforementioned test suites were close contenders – Diehard and NIST[8]. The NIST framework, with a few alterations (see Appendix X), has been chosen to evaluate Random.org mostly for pragmatic reasons, including:

- It is a standard that is widely recognised in the literature and industry [X, X, X]. This is the dominant argument as to why NIST is used. If it is shown that Random.org passes the NIST suite it can be said to pass the industry standard. This also makes for easy comparison between generators that have been subjected to the NIST suite, of which there are many.
- It is the source with the most stringent tests, being designed to test generators for cryptographic applications. Because Random.org is not being tested for an application, it was thought most appropriate to use a test suite that tests for application with the highest stringency requirements (cryptography) and by default satisfies the requirements of other applications. Note that this means that the test set will fail some generators that are suitable for some applications.
- It was used for the evaluation of AES (Advanced Encryption Algorithm) candidates, encryption algorithms deemed capable of protecting sensitive government information. [51]
- The NIST tests were designed to be run on binary numbers. See section 3.1 as to why this is preferable. The NIST tests are designed to test binary sequences. Many others, including Diehard, are geared towards testing uniform numbers.
- The issue of the independence and coverage of the statistical tests has been broached by NIST (though the reason was not available to examine).

The NIST suite, it seems, is the standard in the world of RN generation at the moment. For this reason the client can confidently confirm that the numbers generation by Random.org are completely random if

[8] Of course an existing battery of tests need not have been chosen at all. A new suite of tests could have been developed but this was deemed beyond the scope of the time constraints of this project.

they pass the battery of tests. There is no doubt, however, that there will be another standard in the future to replace the NIST suite, just as Diehard replaced Knuth's suite and NIST has replaced the Diehard suite.

4.1.5 Description of Tests

Following is a brief overview of what kind of departures from randomness that the tests in the NIST suite aim to detect. See Appendix K for a more detailed description and step-by-step guide.

NIST Statistical Test Suite		
Test	**Defect Detected**	**Property**
Frequency (monobit)	Too many zeroes or ones	Equally likely (global)
Frequency (block)	Too many zeroes or ones	Equally likely (local)
Runs test	Oscillation of zeroes and ones too fast or too slow	Sequential dependence (locally)
Longest run of ones in a block	Oscillation of zeroes and ones too fast or too slow	Sequential dependence (globally)
Binary matrix rank	Deviation from expected rank distribution	Linear dependence
Discrete fourier transform (spectral)	Repetitive patterns	Periodic dependence
Non-overlapping template matching	Irregular occurences of a pre-specified template	Periodic dependence and equally likely
Overlapping template matching	Irregular occurences of a pre-specified template	Periodic dependence and equally likely
Maurer's universal statistical	Sequence is compressible	Dependence and equally likely
Linear complexity	Linear feedback shift register (LFSR) too short	Dependence
Serial	Non-uniformity in the joint distribution for m-length sequences	Equally likely
Approximate entropy	Non-uniformity in the joint distribution for m-length sequences	Equally likely
Cumulative sums (cusum)	Too many zeroes or ones at either an early or late stage in the sequence	Sequential dependence
Random excursions	Deviation from the distribution of the number of visits of a random walk to a certain state	Sequential dependence
Random excursions variants	Deviation from the distribution of the number of visits (across many random walks) to a certain state	Sequential dependence

Table 4.1 NIST Statistical Test Suite

Note: There are a number of alterations to the NIST statistical test suite that have been taken into account when testing (see Appendix I for details).

4.1.6 Revised Set of Tests

The client expressed an interest in a prioritisation of tests within the recommended test suite. The problem with this, however, is that it cannot be then said that the numbers satisfy the NIST statistical test suite, the adopted industry standard.

The author reluctantly categories the tests into the following three tiers:

Prioritisation of Tests			
Test	**Tier 1**	**Tier 2**	**Tier 3**
Frequency (monobit)			x
Frequency (block)	x	x	x
Runs			x
Longest run of ones in a block	x	x	x
Binary matrix rank		x	x
Discrete fourier transform (spectral)	x	x	x
Non-overlapping template matching	x	x	x
Overlapping template matching			x
Maurer's universal statistical		x	x
Linear complexity		x	x
Serial	x	x	x
Approximate entropy			x
Cumulative sums (cusum)	x	x	x
Random excursions	x	x	x
Random excursions variants			x
Number of Tests	**6**	**10**	**15**

Table 4.2 Prioritisation of the NIST Tests

Tier 1 contains seven tests and it is suggested that they pick up the most important type of non-randomness.

Tier 2 encompasses three more tests than tier one. The rationale behind why these four tests do not feature in Tier 1 is:

Binary Matrix:	Linear dependence seems to the author a bizarre feature to detect
Linear Complexity:	It is mostly for practical reasons that this test does not feature in the Tier 1 list; it takes computationally intense, taking over 2 hours to run through on an ordinary PC[9].
Maurer:	Known to have a low power.

Tier 3 encompasses five more tests than Tier 2 and is the full NIST statistical test suite. That rationale behind why these five tests do not feature in higher tiers is:

Frequency (monobit):	Similar to the frequency block test. Also, if this test is failed then other tests will definitely fail so there is perhaps a redundancy in applying it.
Runs:	Similar to the longest run within a block test in Tier 1.
Overlapping template matching:	Similar to the non-overlap test in Tier 1.
Approximate entropy:	This test is very similar to the serial test. They seem to only differ in the test statistic which they calculate.
Random excursions variants:	Similar to random excursions test in Tier 1.

[9] NIST also note that the Linear Complexity test is the most time-consuming statistical test to run. (And so this suggests that it is not just poor coding

How valid these three tiers are is subject to further analysis; the prioritisation is based predominantly on the experience of the author (over the past few months). There is a reference in the literature to the Information Technology Promotion Agency in Japan selecting a minimum set of tests from the NIST suite to include Frequency test within a block, Longest run of ones in a block, linear complexity, serial, cumulative sums. [23] The documentation of this was not identified. The minimum set chosen however is quite consistent to what the author has prioritised as tier 1.

4.2 Methodology

4.2.1 Which numbers should be tested?

As already reasoned in Section 3.1 it is desirable both from a philosophical and mathematical point of view that binary numbers be tested. The tests should *not* be run on the same set of numbers but rather on independent sets of numbers. This means that the results of each test can be deemed to be independent. NIST does not explicitly deal with this issue.

4.2.2 Multiple testing?

The ideal is to have some test which will indicate whether some accumulation of sequences is completely random. The best that any test can offer, however, is the probability of getting such an accumulation under whatever assumptions are made. That is, there is always a valid possibility, no matter how small, that even very peculiar accumulations of strings of numbers could have occurred by chance. As is noted in the statistics world, “p happens”, meaning that sometimes a RNG that is in fact completely random will fail a test (Type I error). And so it is unwise to reject outrightly a generator on the basis of the results from one hypothesis test. If “p happens” then the RNG should be judged a suspect of departing from true randomness and further testing should be done. NIST suggests that the tests be ran multiple times; taking the number of runs to be x, where x is at least the inverse of the significance level, α. As the significance level for all tests is 0.01 (see Section 4.2.4) this suggests running each test 100 times. Despite NIST recommendations, multiple testing in the context of hypothesis testing seems not to have a solid statistical basis. Classically one test is conducted on the data. It is suggested that chance peculiar accumulations of numbers do not have an effect if enough data is used in the test and so this multiple testing basically tells nothing more than running the test once. Nevertheless, the NIST suite is the standard and is adhered to in the evaluation of Random.org and comparatives. When the tests are ran daily on Random.org, however, it is sufficient to adhere to the classical approach of conducting one test, baring in mind that “p happens”.

4.2.3 Input Sizes

Practically a sample output sequence of the RNG is subjected to various statistical tests. The determination as to how long this sample output be for the purposes of statistical testing is difficult to address and, with no definitive rules on how many numbers should be tested, there is a certain amount of

arbitrariness involved. It is important to realise that not each test necessarily has to be applied to the same amount of numbers. Indeed different tests are suited to different input sizes.

NIST gives recommended minimum input sizes though do not specify a maximum. Random numbers are cheap in the sense that they are easily generated. For this reason there is perhaps a temptation to run tests on a large amount of numbers even though this tells nothing extra. The minimum input sizes that NIST recommends are chosen. These along with other parameters details are in Appendix L.

NIST notes that for many tests, the length of the sequence n is large (of the order 10^6). For such large sample sizes of n, asymptotic reference distributions have been derived and applied to carry out the tests. Most of the tests are applicable for smaller values of n. However, if used for smaller values of n, the asymptotic reference distribution might be inappropriate and would need to be replaced by exact distributions that would commonly be difficult to compute. This varies from test to test.

4.2.4 Pass/Fail Criteria

The pass/fail criteria of the testing revolves around the significance level, α. If the p-value of the test is less than α, then the RNG fails the test, otherwise the test is passed. If the RNG under inspection fails any one of the tests in the suite then it should not be accepted as random. This is because every test is detecting a different type of randomness. NIST recommends that a 0.01 significance level be used for all tests. Suppose that each of the 15 tests in the suite result in one p-value. Because each of the 15 tests in the suite is applied to a different set of numbers, it should be the case that $0.99^{15} = 86.01\%$ of completely random RNGs pass the test. The fact that some completely random RNG fail is referred to as a type I error. Intuitively it makes sense perhaps to lower the significance to say 0.05. This would mean that it is more difficult to pass the test thus capturing the “bad” generators more easily. The problem with this is that it also makes if more difficult for a “good” generator to pass. Appendix G shows the significance level and the associated pass rates if the generator is indeed completely random. For example, if a significance level of 0.05 is taken then a “good” generator will only pass 46.33% of the time. This means that a lot of good generators can be rejected if the significance level is not chosen with care.

Some of the tests in the NIST suite result in more than one p-value, namely the Serial, Cumulative sums, Excursions and Excursions Variant tests. It is unclear from the NIST documentation as to how these should be treated. For instance, the Excursions Variant test outputs 18 p-values. Should all 18 p-values be looked at? This would mean putting unequal weightings on the tests. If all the resulting p-values were examined there would be 41 per suite test. This means that there is less than the aforementioned 86% chance that a perfectly random RNG will pass the suite (because of the dependence of some p-values it is not possible to calculate the exact pass rate percentage). This 86% is already rather low. In the description of the tests NIST advise rejecting the generator if any of the p-values are unacceptable while further in the document 15 p-values are referred to suggesting that there is 1 p-value from every test.[10]

[10] What NIST might have done to get 15 p-values is taken the lowest p-value on the basis that the following is true: $P(A \cap B) \in \min(P(A), P(B))$ This, however, is just speculation.

This lack of clarity is an annoyance when every effort is made to stick to the test suite as closely as possible. It is decided, for completeness sake, to look at all 41 p-values.

The blanket significance level of 0.01 does not take into account the issue of power-versus-size, as discussed further in section 6. Each test has a different power and a different significance level and using a blanket significance level is not very efficient. Although not satisfactory, the NIST recommendation is adhered to because ultimately it is desired that a statement be made saying whether or not Random.org passed the NIST statistical test suite. Such a statement cannot be made if departures are made from the NIST recommendations.

The problem with the fail/pass decision approach employed by NIST is that much relevant information is discarded. For example, how close a fail or pass the sequence was. It seems that a more flexible approach is needed. Ultimately the pass/fail criteria should depend on the application.

4.2.4.1 Proportion of sequences passing the test

According to NIST suite the range, or confidence interval, of acceptable pass rates (where the pass rate is defined to be the proportion of times that a generator passes a particular test) needs to be determined. NIST use the normal distribution as an approximation to the binomial distribution. This is only applicable to large sample sizes. The general rule of thumb is that if npq>5 (where n=number in sample, p=probability of success and q=1-p=probability of failure) then the normal approximation can be used to develop a confidence interval for a binomial variable. This essentially approximates a discrete distribution with a continuous distribution. This is not applicable here. A sample size of at least 506 would be needed[11] but this project is only taking a sample size of 100. And so, instead the confidence interval is calculated by the modified Wald method. [24,25] The 99% confidence interval extends from 0.8995 to 1, meaning that any proportions that fall within this range are acceptable. This range, of course, could be narrowed by taking a larger sample size.

4.2.4.2 Uniform Distribution of p-values

It is not sufficient to look solely at the acceptance rates and declare that the generator be random if they seem fine. If the test sequences are truly random, the p-values calculated are expected to appear uniform in $[0,1)$. This needs to be checked. NIST recommends to conduct a chi-square test on the p-values, dividing the interval 0-1 into 10 sub-intervals. This tests the uniformity of the p-values. The degree is freedom is 9 in this case. Define F_i as the number of occurrences of the p-value in the i-th interval, then χ^2 statistic is

[11] $npq > 5$
$p = 0.99, q = 0.01$
$n * 0.99 * 0.01 > 5$
$n > \frac{5}{0.99 * 0.01}$
$n > 505.05$

$$\chi^2 = \sum_{i=1}^{10} \frac{(F_i - n/10)^2}{n/10}$$

NIST recommends its significance level as 0.01% (i.e. 0.0001). Therefore the acceptance region of statistics is $\chi^2 \leq 33.725$. This type of failure occurs when a generator fails in an inconsistent way. A generator can pass a type of test often enough to avoid being declared proportionally flawed, but be declared a uniformity failure due to otherwise inconsistent behavior. For example, having a few spectacular failures occur could produce a uniformity failure declaration. A generator that experiences periods of poor statistical performance between periods of otherwise excellent statistical performance would likely be declared a uniformity failure. A completely random RNG is classified as a generator without any proportional or uniformity flaws.

What has been described here is a recommendation of the NIST statistical test suite. However, statisticians may very well object to this on the grounds that it is testing the same thing multiple times. It seems that numbers are being ground through the mill again and again reducing the whole analysis to a single number. How this number can be interpreted at the end of this process is questionable.

What may be more useful is to plot the p-values in a chart over time and examine them graphically (see Appendix O).

5. RESULTS OF TESTING

This chapter briefly discusses the results of the NIST test suite on numbers generated by Random.org and the chosen comparative generators. Appendix N gives the details of the results and a further discussion.

(Note: the results are available in spreadsheet format on the CD that is attached to the inside of the back cover of the report)

Random.org passes the tests in terms of pass rates, where the pass rate is the proportion of times that Random.org passes a particular individual test. All pass rates are well above the lower bound of 89.95% as calculated in Section 4.2.4.1. These are shown in Table N1 of Appendix N. The distribution of p-values is deemed to be uniform by the method described in Section 4.2.4.2. A visualisation of the distribution of p-values for each test is shown in Appendix N also. It can now justifiably be claimed that Random.org passes the NIST suite of statistical tests.

It is worth bearing in mind that if a generator results in one or more significant p-values across all tests then the null hypothesis is rejected. So while all the individual pass rates are quite high, it was found that Random.org passed the suite 68% of the time (based on 100 runs). This may seem quite low but remember that “p-happens” meaning that a “good” RNG will fail an individual test with probability alpha. 41 p-values must be above 0.01 so while there is no way to calculate the expected overall suite pass rate (because of the dependence between some p-values) this 68% sounds reasonable.

Perhaps what is more insightful is a comparison of Random.org with other generators. The comparative PRNGs that have been chosen are the Microsoft Excel RNG, which is commonly used tool in almost every field and the Minitab[12] [26] RNG, which is frequently used in statistics. The comparative TRNGs that have been chosen are Hotbits [27] and Randomnumbers.info [11]. How the numbers are generated by these generators, as well as Random.org, is detailed in Appendix M.

While Random.org and the PRNGs were subjected to the each test 100 times, the TRNGs were subjected to each test only once for practical reasons (see Appendix C).

Appendix N shows the detailed results of the testing. Like Random.org, Minitab passed the test suite. Excel’s pass rates were satisfactory but the resulting p-values were lacking uniformity in two of the tests and so Excel fails the suite. Of the other two TRNGs, Hotbits passed the suite but Randomnumbers.info failed two of the tests.

For the generators that passed the tests one cannot be considered better than the other. They all meet the requirements of the NIST statistical test suite.

[12] Minitab is a statistical software package

6. OPEN ISSUES

This chapter deals with some open, mainly statistical, issues that need consideration that goes beyond the scope of the project.

6.1 Evaluation of the Test Suite

Essentially the next three sections are saying that the actual test suite needs evaluation
What is hopefully obvious from the issues raised in the preceding chapters is that the test suite needs to be evaluated. This section briefly discusses three specific unresolved issues that ought to get attention from the academic community.

6.1.1 Power of Tests

The significance level of a test is the probability that a type I error is made. A type I error is made when the statistical test classifies a “good” RNG as “bad”. The power of a test is the probability that a type II error is *not* made. A type II error is made when a statistical test classifies a “bad” RNG as “good”. The higher the power, the better.

In this context it is desirable to have high power against all possible alternatives. It is not so important to have a low significance level - The consequence of a “bad” generator being classified as “good” is worse than a “good” generator being classified as “bad” (in most applications). The latter results in merely a loss in efficiency of the testing procedure whereas the former can have detrimental effects on the application. For example, for a cryptographic purpose it could mean a potential exposure of the data intended to be encrypted and for a simulation it could mean that the results are distorted. And so, the power of the test certainly does not want to be compromised. On the other hand, rejecting perfectly acceptable generators is not desirable either. A balance has to be made between the two types of errors. The only way to achieve high enough power is to use large enough samples.

Power depends on the test type (what type of departure is being tested for), the extent of that departure and the sample size. Each statistical test should have their own power and therefore an appropriate significance level and input size should not be the same for every test. The power of the NIST tests does not seem to be not documented. What power is adequate depends on the extent of departure that can be tolerated; this leads back to the application once again.

Bayesian methods may be a technique to approach this problem. This, unfortunately, was an avenue that could not be explored due to the time constraints of this project. Although, given what has been done in this project it would now be feasible.

6.1.2 Independence and Coverage of tests

The issue of the independence of the statistical tests, whether or not there is any redundancy in applying more test than are indeed necessary has been broached by NIST. The coverage or span of the statistical tests seeks to address the problem of how many distinct types of non-randomness can be investigated, and to assess whether or not there are a sufficient number of statistical tests to detect any deviation from randomness. To address this problem research is underway which involves the application of principal components analysis and comprehensive coverage of tests have also been considered. NIST say that the results look promising. [9,51] It is not possible to go through the details of this work because they are not available. However, using principal components analysis, which assumes linearity, to look at p-values, if that is what they are doing, seems awkward because p-values are not linear. It will be impossible to ever declare that the tests detect infinite amount of non-randomness that could exist. The best that can be hoped for is that they detect the important types of non-randomness, where the importance effectively depends on the application.

6.1.3 Interpretation of Results

NIST report that “it is up to the tester to determine the correct interpretation of the test results”. Considering that NIST designed the test this is not terribly helpful advice nor is it adequate. There are probably more people from a non-statistical background that use random numbers than those from a statistical background and some clear guidance does need to be given in the interpretation of results.

6.2 Application Based Testing

Section 3.4 mentions application based testing as a method to test the performance of a particular RNG. Passing many statistical tests is never a sufficient condition for the use of a RNG in all applications. In other words, in addition to standard tests such as the NIST suite, application specific tests are also needed. To test whether a RNG is good enough for a gaming application, for example, a trial could be set up in which individuals, or more likely learning software, would collect useful statistics for a period. Neural nets could then perhaps be used to discover good betting strategies. The decision criteria to accept the generator would be that if the average loses are less than the theoretically calculable expected loss. To test whether a RNG is good enough for a cryptographic application encrypt some data and then try to break the code. Of course, this kind of an approach is not practical if testing a multi-purpose generator like Random.org from the supplier end. However, for a user of Random.org with a particular application this is the recommended approach where possible.

A. ORIGINAL PROJECT GUIDELINES

Client:	Distributed Systems Group, Computing Science Dept., Trinity College
Project:	On-line statistical analysis of a true random number generator
Location:	Distributed Systems Group, Computer Science, O'Reilly Institute
Client Contact:	Mads Haahr, Mads.Haahr@cs.tcd.ie, phone (01) 608 1543
Dept. Contact:	Simon Wilson

Client Background

The Distributed Systems Group (http://www.dsg.cs.tcd.ie/) is one of the research groups in the Computer Science Department. It conducts research in many different areas of distributed computing.

Project Background

The Distributed Systems Group is operating a public true random number service (http://www.random.org) which generates true randomness based on atmospheric noise. The numbers are currently made available via a web server. Since it went online in October 1998, Random.org has served nearly 10 billion random bits to a variety of users. Its popularity is currently on the rise and at the moment the web site receives approximately 1000 hits per day. The group is concerned to verify that the output of its random number service can truly be considered "random".

Client Requirement

The objectives of this project are first to implement a suite of statistical tests for randomness on the output of this stream. These are to be implemented using a statistical package, Excel, or, if the student wants, by writing code. Then, a comparison should be made with other 'true' random number generators and with some of the more usual 'pseudo' random generation algorithms. The second part of the project involves integrating the test functionality with the random.org number generator. This may involve managing a database containing the numbers generated (or, possibly, a summary of the numbers) and linking an analysis of the database to the web for users of the service.

What is involved for the student?

Clearly the first part of this project is overwhelmingly statistical in nature. A survey of suitable statistical tests will have to be made, and then the tests implemented, using a statistical package, Excel or through writing code explicitly. The second part of the project involves managing a database (the numbers generated) and linking an analysis of the database to the web for users of the service.

B. INTERIM REPORT

Management Science and Information Systems Studies

Project: Statistical Analysis of a True Random Number Generator
Client: Mads Haahr, Distributed Systems Group, Computer Science Department, Trinity College
Student: Charmaine Kenny
Supervisor: Kris Mosurski

Review of Background and Work to Date

The Distributed Systems Group (DSG) is a research group in the Department of Computer Science in Trinity College Dublin. They conduct basic and applied research into all aspects of distributed computing. The primary objective of this project is to analyse their on-line random number generator. The generator uses atmospheric noise to produce random numbers. It is freely available at www.random.org.

To date, familiarity with the random.org website has been established. Research into the definition of random numbers and different types of randomness has begun. The applications of random numbers have also been overviewed. Preliminary work on understanding common tests has started. A good number of academic papers pertaining to the topic of random numbers and testing random number generators have been ascertained as well as useful websites and relevant text books. Some available statistical test suites for random number generation have also been identified.

Terms of Reference

- To conduct a literature review of the applications of random number generators and to contrast the use of random number generators and pseudo random number generators;
- To research statistical tests that detect non-randomness, review statistical test suites available, and then propose a set of statistical tests to be applied to the numbers generated by random.org;
- To consider other random number generators as possible comparative studies and compare random.org to a selection of these:
- To define, prioritise and spec the efficiency of the implementation of the proposed suite.

Further Work

Micro-deadlines have been constructed to ensure the momentum of the project does not ease. The target schedule for the project is detailed in Figure 1.

Detailed Project Schedule

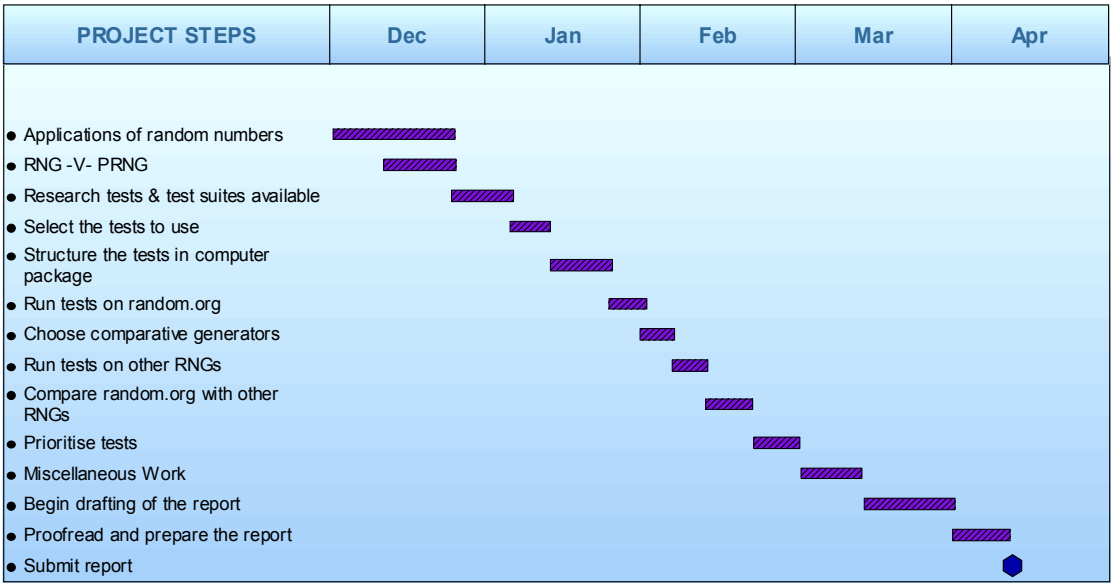


Figure 1

Conclusions

While the drawing of conclusions at this preliminary stage is resisted, a number of things have already become clearer as a result of work carried out to date, among them:

The scope of the project, as defined by the terms of reference, has been clearly set out. It is important to note that the project does not entail the implementation of the proposed statistical test suite on-line but that the work involved with this project is a huge step towards achieving this.

Finally, it is recognized that the project is doable given the inherent constraints (which mainly involve time).

C. DIFFICULTIES ENCOUNTERED

Although the terms of reference have largely been fulfilled there were a number of difficulties encountered in their fulfilment.

The difficulty in programming the statistical tests and therefore the time required to do so was underestimated. Errors in the NIST manual, which describes the tests, further delayed the process (Appendix J documents these errors). Running the tests was also a lengthy and laborious task. One test in particular proved to be awkward – the Linear Complexity Test. This is a computationally intense test, taking over two hours to run on a college PC. The NIST developers of the test suite also note that it is the most time-consuming test to run and so this lengthy run-time it is not just a product of the author's inefficient coding! Priority was given to Random.org but the test was not run 100 times for all of the comparative generators (Minitab was only subjected to the Linear Complexity test 26 times).

It was anticipated that a comparison be made between Random.org and some pseudo and true random number generators. There was awkwardness in getting output from the latter to test. While there is indeed a number of online true random number generators that provide a free service, there are restrictions on the amount of numbers that can be downloaded. These restrictions meant that the ~5.5 million numbers needed to run through the recommended suite would have taken far too long to generate. For this reason the true random number generator comparisons were ran through the suite of tests just once (which is consistent with hypothesis testing anyway) as opposed to 100 times, as with Random.org and the pseudo random number generator comparatives.

In the earlier days of researching the project it was very easy to get lost in the literature. Because the use of random numbers spans across many different fields the literature was vast if not always relevant. The different fields of random number work do not seem to converge very often in the literature. It is unfortunate that relevant work is published in so many journals in so many fields; it makes for difficulty in keeping track of new developments and it also enables many outdated methods to get in print. It perhaps also shades scope for different fields to learn from each other; it certainly makes it more difficult.

D. TYPES OF PRNGs

Pseudo random number generators (PRNGs) use an algorithm to produce sequences of random numbers. Common classes of algorithms are linear congruential generators, lagged Fibonacci generators, linear feedback shift registers and generalised feedback shift registers. Recent instances of algorithms include Blum Blum Shub and the Mersenne Twister.

Linear Congruential Generators

Linear congruential generators (LCGs) represent one of the oldest and best-known pseudorandom number generator algorithms. The theory behind them is easy to understand, and they are easily implemented and fast. It is, however, well known that the properties of this class of generator are far from ideal. LCGs are defined by the recurrence relation:
$V_{j+1} \equiv A \times V_j + B (\mathrm{mod}\, M)$, where $\mathrm{V_n}$ is the sequence of random values and A, B and M are generator-specific constants.

The period of a general LCG is at most M, and very often less than that. In addition, they tend to exhibit severe defects. For instance, if an LCG is used to choose points in an n-dimensional space, triples of points will lie on, at most, $\mathrm{M}^{1/\mathrm{n}}$ hyperplanes. This is due to serial correlation between successive values of the sequence $\mathrm{V_n}$. A further problem with LCGs is that the lower-order bits of the generated sequence have a far shorter period than the sequence as a whole if M is set to a power of 2. In general, the nth least significant digit in the base m representation of the output sequence, where mk = M for some integer k, repeats with at most period mn.

Today, with the advent of the Mersenne twister, which both runs faster than and generates higher-quality deviates than almost any LCG, only LCGs with M equal to a power of 2, most often M = 232 or M = 264, make sense at all. These are the fastest-evaluated of all random number generators; a common Mersenne twister implementation uses it to generate seed data.

LCGs should not be used for applications where high-quality randomness is critical. For example, it is not suitable for a Monte Carlo simulation because of the serial correlation (among other things). Nevertheless, LCGs may be the only option in some cases. For instance, in an embedded system, the amount of memory available is often very severely limited. Similarly, in an environment such as a video game console taking a small number of high-order bits of an LCG may well suffice.

Lagged Fibonacci Generators

The lagged Fibonacci generator (LFG) class of random number generator is aims to be an improvement on the 'standard' linear congruential generator. These are based on a generalisation of the Fibonacci sequence. The Fibonacci sequence may be described by the recurrence relation:

$$\mathrm{S_n} = \mathrm{S_{n-1}} + \mathrm{S_{n-2}}$$

Hence, the new term is the sum of the last two terms in the sequence. This can be generalised to the sequence:

$$S_n = S_{n-j} (*) S_{n-k} \pmod M, 0 < j < k$$

In which case, the new term is some combination of any two previous terms. M is usually a power of 2, often 232 or 264. The (*) operator denotes a general binary operation. This may be either addition, subtraction, multiplication, or the bitwise arithmetic exclusive-or operator. The theory of this type of generator is rather complex, and it may not be sufficient simply to choose random values for j and k. These generators also tend to be very sensitive to initialisation. Generators of this type employ k words of state (they 'remember' the last k values). If the operation used is addition, then the generator is described as an Additive Lagged Fibonacci Generator or ALFG, if multiplication is used, it is a Multiplicative Lagged Fibonacci Generator or MLFG, and if the exclusive-or operation is used, it is called a Two-tap Generalised Shift Feedback Register or GFSR. The Mersenne twister algorithm, which is discussed further on, is a variation on a GFSR.

Lagged Fibonacci generators have a maximum period of $(2^k - 1)*2^{M-1}$ if addition or exclusive-or operations are used to combine the previous values. If, on the other hand, multiplication is used, the maximum period is $(2^k - 1)*2^{M-3}$, or ¼ of period of the additive case. For the generator to achieve this maximum period, the polynomial:

$$y = x^k + x^j + 1$$

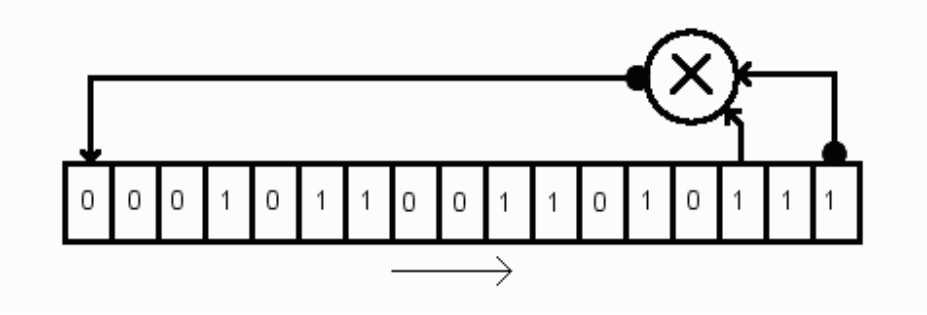
must be primitive over the integers mod 2. Values of j and k satisfying this constraint have been published in the literature. Popular pairs are: {j = 7, k = 10}, {j = 5, k = 17}, {j = 24, k = 55}, {j = 65, k = 71}, {j = 128, k = 159}. It is required that at least one of the first k values chosen to initialise the generator be odd.

There are a number of problems with LFGs. Firstly, the initialisation of LFGs is a very complex; any maximum period LFG has a large number of possible cycles, all different. Choosing a cycle is possible, but methods for doing this may endanger the randomness of subsequent outputs. Secondly, the output of LFGs is very sensitive to initial conditions, and statistical defects may appear initially but also periodically in the output sequence unless extreme care is taken. Another potential problem with LFGs is that the mathematical theory behind them is incomplete, making it necessary to rely on statistical tests rather than theoretical performance. These reasons, combined with the existence of the free and very high-quality Mersenne twister algorithm tend to make 'home-brewed' implementations of LFGs less than desirable in the presence of superior alternatives.

Linear Feedback Shift Register Generators

A linear feedback shift register is a shift register whose input is the exclusive-or of some of its outputs. The outputs that influence the input are called taps. A maximal LFSR produces an n-sequence, unless it contains all zeros. The tap sequence of an LFSR can be represented as a polynomial mod 2 - called the feedback polynomial. For example, if the taps are at positions 17

and 15 (as below), the polynomial is x17 + x15 + 1. If this polynomial is primitive, then the LFSR is maximal.



LFSRs can be implemented in hardware, and this makes them useful in applications that require very fast generation of a pseudo-random sequence, such as direct-sequence spread spectrum radio. Given an output sequence you can construct a LFSR of minimal size by using the Berlekamp-Massey algorithm. [30]

LFSRs have long been used as a pseudo-random number generator for use in stream ciphers (especially in military cryptography), due to the ease of construction from simple electromechanical or electronic circuits, long periods, and very uniformly distributed outputs. However the outputs of LFSRs are completely linear, leading to fairly easy cryptanalysis. Three general methods are employed to reduce this problem in LFSR based stream ciphers:

- Non-linear combination of several bits from the LFSR state;
- Non-linear combination of the outputs of two or more LFSRs; or
- Irregular clocking of the LFSR.

Other PRNGs

Blum Blum Shub (BBS) is a pseudorandom number generator proposed in 1986 by Lenore Blum, Manuel Blum and Michael Shub which gained a lot of recognition in the field of cryptographic. Much has been written about this generator [31,32].

The Mersenne twister is a pseudorandom number generator that was developed in 1997 by Makoto Matsumoto (松本 眞) and Takuji Nishimura (西村 拓士). It provides for fast generation of very high quality random numbers, having been designed specifically to rectify many of the flaws found in older algorithms [33].

Note that the descriptions above are largely sourced from www.answers.com [34]. Other interesting background reading on PRNGs is Knuth [5], Ripley [35], Vattulainen [13] and Menezes et al. [8].

E. HYPOTHESIS TESTING

This appendix supplements the brief discussion of hypothesis testing in Section 3.4 in the main body of the report. It is particularly useful for the reader who does not have a background in statistics.

The framework adopted to test the random number generators is based on hypothesis testing. A hypothesis test is a procedure for determining if an assertion about a characteristic of a population is reasonable. In this case, the test involves determining whether or not a specific sample sequence of zeroes and ones is random. Practically, only a sample output sequence of the RNG is subjected to various statistical tests.

Table E.1 lists some terminology associated with hypothesis testing that is needs to be defined for the unfamiliar reader.

Term	Definition
Test statistic	A statistic upon which a test of a hypothesis is based. For example, in this project the chi-square statistic is the test statistic for many of the tests.
Null hypothesis	The stated hypothesis. In this case, the null hypothesis is that a binary sequence is random from a statistical viewpoint.
Alternative hypothesis	The alternative to the null hypothesis. In this case it is any non-random characteristic.
Significance level	Usually denoted as, alpha (α), it is the least upper bound of the probability of an error of type I for all distributions consistent with the null hypothesis. The significance level is also referred to as the "size" of the test.
Type I error	The likelihood that a test rejects a binary sequence that was, in fact, produced by an acceptable random number generator.
Type II error	The likelihood that a test accepts a binary sequence that was, in fact, produced by an unacceptable random number generator.
Confidence interval	An interval which is believed, with a pre-assigned degree of confidence, to include the particular value of some parameter being estimated.
p-value	A measure of the strength of the evidence provided by the data against the hypothesis.
Critical Value	A "look up" or calculated value of a test statistic that, by construction, has a small probability of occurring when the null hypothesis is true.

E.1 Statistical Hypothesis Testing [28]

For each test, a relevant statistic must be chosen and used to determine the acceptance or rejection of the null hypothesis. Under an assumption of randomness, such a statistic has a distribution of possible values. A theoretical reference distribution of this statistic under the null hypothesis is determined by mathematical methods. From this reference distribution, a critical value is determined (typically, this value is “far out” in the tails of the distribution). During a test, a test statistic value is computed on the data (the sequence being tested). The test statistic value is used to compute a p-value. If a p-value for a test is determined to be equal to 1, then the sequence appears to have perfect randomness. A p-value of zero indicates that the sequence appears to be completely non-random. A significance level (alpha) is chosen for the tests. In this project the significance level or size is taken to be 0.01 for each test. The significance level of the test is the probability of rejecting the null hypothesis when it is true. If alpha>0.01, then the hypothesis is accepted, i.e., the sequence would be considered to be random with a confidence 1-alpha. If alpha<0.05, then the hypothesis is rejected, i.e., the sequence would be considered to be non-random with a confidence 1-alpha. [6]

F. TESTS WITHIN EACH SUITE

Table F.1 shows that tests that are within each of the suites described in Section 3.5 of the main body of the report. Efforts have been made to display the overlap between the test suites but the difficulty in compiling such a table is that the same test often has many different names and there are also many variations of what is essentially the same test.

	Knuth	Diehard	Crypt-X	NIST	ENT	Previous MSISS
	[5]	[19]	[20]	[6]	[3]	[2]
Frequency	1		1	1		
Serial	1			1		
Gap	1					
Poker/Partition	1					
Coupon Collector's	1					
Permutation	1					
Runs	1	1	1	1		1
Maximum-of-t	1					
Collision	1					
Birthday Spacings	1	1				
Serial Correlation	1				1	
Tests on Subsequences	1					
Overlapping Permutations		1				
Binary rank test for 32x32 matrices		1		1		1
Ranks of 6x8 matrices		1				
Monkey tests on 20-bit words		1				
Monkey tests OPSO, OQSO, DNA		1				
Count the 1's in a stream of bytes		1				
Count the 1's in specific bytes		1				
Parking lot		1				
Minimum distance		1				
Random spheres		1				
Squeeze		1				
Overlapping Sums		1				1
Craps		1				
Binary Derivative			1			
Change Point			1			
Sequence Complexity			1			
Linear Complexity			1	1		
Frequency test within a block				1		
Longest run of 1's in a block				1		
Discrete fourier transform (spectral)				1		
Non-overlapping template matching				1		
Overlapping template matching				1		
Maurer's universal statistical				1		
Approximate Entropy				1	1	
Cumulative sums (cusum)				1		
Random Excursions				1		
Random Excursions Variants				1		
Chi-square					1	1
Arithmetic Mean					1	
Monte Carlo Value for Pi					1	
Reverse arrangements						1
Number of tests in suite	**12**	**15**	**6**	**15**	**5**	**5**

F.1 Tests within each suite

G. SIGNIFICANCE LEVEL

Table G.1 shows the expected pass rates of RNGs that are completely random for various significance levels, given that 15 p-values result from the test suite and that the tests are carried out on independent samples. The pass rate in this case is $(1-\alpha)^{15}$, where α is the significance level. The table illustrates the point being made in Section 4.2.4 of the report that the pass rate decreases quite rapidly for small changes in the significance level and that using what may seem like a reasonable significance level could mean rejecting a large proportion of "good" RNGs.

Significance Level	Pass Rate
0.005	92.76%
0.010	86.01%
0.015	79.72%
0.020	73.86%
0.025	68.40%
0.030	63.33%
0.035	58.60%
0.040	54.21%
0.045	50.12%
0.050	46.33%

G.1 Significance Level –V- Pass Rates

H. CODE DOCUMENTATION

This appendix addresses what language the tests were coded in and why. It gives a brief technical commentary on the specifics of what is needed within the language environment. It also deals with code comments, the set-up of the code, error-checking and debugging.

Note: a disc with the Excel file is attached to the inside of the back cover of the project report.

Coding Language

Contrary to the original project guidelines the client did not require that the author code the tests for integration with his server. After consultation with the client it was decided to code the test using Excel. The advantages and limitations of this decision are outlined below:

Advantages

- The forte of the author is not programming but the author is a competent user of Excel and VBA.
- The Excel-VBA set-up provides excellent and easy to understand pseudo-code for the client when he starts to program the tests to be uploaded onto his server.
- Excel-VBA is simple for non-programmers to understand.
- By coding the tests from scratch a grasp of exactly what the test procedures entailed was gained.

Disadvantages

- Excel-VBA does not have all the functionality that a language like C or even an application like Matlab has. For example the calculation of the rank of a binary matrix had to be written from scratch. This was a little cumbersome and took time.
- Excel has only $2^{16} = 65536$ rows. This minor nuisance was easily surmountable.
- The Excel file is quite large which perhaps puts is at a risk of crashing. This, however, happened rarely.
- Excel is platform dependent, working only on Windows and Macintosh machines.
- Excel cannot be used to link directly to the web.

Language Environment

A number of Excel's built-in functions are used in the calculation of the p-values for tests. At least two of these functions are not available in Excel unless the Analysis ToolPak add-in has been installed. Specifically, the two functions are RANDBETWEEN(a,b) (used to generate a random integer between 0 and 1) and ERFC(x) (used to calculate the complementary ERF function integrated between x and infinity $erfc(x) = \frac{2}{\sqrt{x}} \int_x^\infty e^{-t^2} dt = 1 - erf(x)$).

If these functions are not available (Excel returns the "#NAME?" error) install and load the Analysis ToolPak add-in by carrying out the following steps:

- On the Tools menu, click 'Add-Ins'.
- In the 'Add-Ins available' list, select the 'Analysis ToolPak' box, and then click OK.
- If necessary, follow the instructions in the setup program.

The NIST manual refers to a function called igamc which is related is related to the chidist function of Excel. The relationship is $igamc(\frac{df}{2}, \frac{\chi^2}{2}) = chidist(\chi^2, df)$. The chi-square distribution is a special case of the more general gamma distribution.

Code comments

The code has been commented extensively to aid the reader in comprehension. A detailed description of each of the tests is given in Appendix K which is a type of pseudo-code should the reader wish to clarify. Perhaps the best way to understand the code is to read this simultaneously i.e. the comments in the code and description of the tests.

Set-up

For the purposes of Excel the code has been set up such that it deals with numbers formatted in 20 numbers in each row. For example, if 100 numbers are needed for a test then there should be 20 columns of numbers with 5 in each column. The reason for this is because Excel can only accommodate 2^16 i.e.65536 rows and 2^8 i.e. 256 columns.

The user can paste the data, in the correct format, into the worksheet called "Data" in the excel file. The user should then click on the "Display" worksheet and click on the appropriate button run the desired test.

The code does not run the suite of tests on the numbers simultaneously because there is a high chance of an error occurring – it would have to deal with almost ~5.5 million numbers. In any case this interactive approach is more conducive to understanding the how the tests work. Furthermore, it takes quite some time to run all the tests one after the other (the linear complexity test takes over two hours to run on a normal machine). Having said that, the code is flexible enough that it can be set up to apply the suite all at once.

The default setting is that when a test is ran once when chosen. The number of times that a test can be ran can easily be changed by going to the display page in the visual basic editor and changing the run time as appropriate.

For the tests that require small amount of numbers the data can be formatted in the usual 20-per-row. For the tests that require larger amounts of random numbers, i.e. >300,000, the code can only cater for running between five and ten tests if the data is placed side by side but this must be explicitly be taken account of in the code by again changing the run time.

Remember that different numbers should be used for each test (see Section 4.2 in the main body of the report).

The tests as programmed by NIST are available for download from their website at http://csrc.nist.gov/rng/rng2.html. This was not used in the project because it was desired to get a firm understanding of what exactly the tests do instead of taking a black-box approach. The NIST test code was developed in ANSI C. The accuracy of this code cannot be commented upon but the author has reservations about using it as it is. The number of mistakes found in the NIST manual does not instill confidence. See appendix J for details of these mistakes and corresponding corrections. Indeed it might be possible to test the NIST code by using the Excel-VBA code written for this project.

Debugging and error checking

Every effort has been made to debug the project given the time constraints. For example the binary matrix code has been cross-checked with MatLab, the mini-examples in the manual have been ran through the code to verify that the same answers are calculated, etc

It is recognised that the code needs some revision to make it completely bug-free. Some of the error-checking procedures that need to be carried out include:

- Ensuring that there are enough numbers supplied for a particular test (input sizes are given in Appendix L). If there are not enough numbers for a particular test the code currently assumes that the rest of the sequence is 0,0,0,0…
- There are parameter requirements that must be satisfied so that the test is valid. Checks need to be integrated into the code to ensure that these requirements are adhered to. This parameter requirements are detailed in the descriptions of the tests in Appendix K
- There is currently no check in place that ensures all numbers are in binary form.

Note that there is some concern in the literature about using Excel for statistical calculations, including p-values [36, 37]. The p-values calculated by Excel's "=chidist(a,b)" function are fine for their intended use here. They were cross-checked with Matlab. In any case, the p-values do not need a small percentage point accuracy for this application.

I. ALTERNATIONS TO THE NIST STATISTICAL TEST SUITE

The 'NIST manual' is shorthand for the document entitled "A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications – NIST Special Publication May 2001. [6]

There are a numbers of alterations to the NIST manual that are made in the RNG testing in this project. These are as follows:

- Discrete Fourier Transform (Spectral) test
  Kim et al [51] show that there is a fault in the setting of this test. The threshold setting of $\sqrt{3n}$ should really be $\sqrt{2.995732274n}$. This deviation makes the distribution invalid and so the correction has been adopted here. Additionally, the suggested correction of the variance σ^2 of theoretical distribution from $\frac{npq}{2}$ to $\frac{npq}{4}$ is also taken on board.
- Lempel-Ziv Compression test.
  The settings of the Lempel-Ziv test have also been showed to be flawed.

The statistical distribution of these two tests is derived from expected distributions. So P-value of this test is not uniform even if the test sequence is perfectly random and the significance level of this test is not 1%. [23] NIST recognises these inadequacies [38], advising that the threshold be decreased to the $\sqrt{2.995732274n}$ level and the Lempel-Ziv Compression test by dropped altogether.

J. CORRECTIONS TO THE NIST STATISTICAL TEST SUITE

There are a number of corrections that need to be made to the NIST manual, including:

1. Igamc

   There are a host of mistakes in relation to the igamc function in the manual. The relationship between igamc, as defined in the manual, and Excel's chidist is:

   $$igamc(\frac{df}{2},\frac{\chi^2}{2}) = chidist(\chi^2, df)$$

   The chi-square distribution is a special case of the more general gamma distribution.

- p112 The incomplete gamma function is defined as:

  $$P(a,x) = \frac{\gamma(a,x)}{\Gamma(a)} = \frac{1}{\Gamma(a)}\int_0^x e^{-t}t^{a-1}dt$$

  This is incorrect, the definition[13] is:

$$P(x,a) = \frac{\gamma(a,x)}{\Gamma(a)} = \frac{1}{\Gamma(a)}\int_0^x e^{-t}t^{a-1}dt$$

The mistake is in the parameters of P which seem to be reversed in the NIST definition. The only reasonable explanation for this is that it is a typo.

- p35 Section 2.8.4 (5) the following equation appears:

$$igamc\left(\frac{5}{2},\frac{3.167729}{2}\right) = 0.274932$$

The correct answer to this is:

$$igamc\left(\frac{5}{2},\frac{3.167729}{2}\right) = chidist(3.167729,5) = 0.674145$$

The mistake made here was that the parameters a and x were mixed up in the program used to get the answer. It seems that similar errors were made on p48 2.12.4 (5) where $igamc\left(2,\frac{1.6}{2}\right)$ is given as 0.9057 when it should be 0.808792 and $igamc\left(1,\frac{0.8}{2}\right)$ is given as 0.8805 when it should be 0.67032

2. Cumulative Sums Test p54 Example 2.14.8

[13] The incomplete gamma function is defined as this in many places. As an example here is a link to Matlab's definition http://www.mathworks.com/access/helpdesk/help/techdoc/ref/gamma.html

The test statistic z is the largest of the absolute values of the partial sums. z, therefore, must be an integer. In Example 2.14.8 z is reported to be 1.6 (forward) and 1.9 (reverse), both of which are non-integer. In the computation of the p-value z is divided by $\sqrt{n}$. In the example $n = 100$. The only logical explanation for this is that the author unintentionally skipped ahead in preparation for the p-value formula and reported $\frac{z}{\sqrt{n}}$ instead of z.

3. Overlapping Sums Test p34 Section 2.8.4 (2)

   In the examination of the blocks to identify the number of occurrences of the target template NIST says that there are 2 occurrences in block 2 whereas there is in fact only one occurrence.

4. Serial Test p47 Section 2.12.4 (2)

   There is a mistake in the illustration of how to determine the frequency of all possible 3-bit blocks. NIST says that $v_{000} = 0$ whereas $v_{000} = 1$. To avoid such a mistake it might be a good idea to add up the frequencies of all the m-bit blocks. The frequencies should add to n, where n=10 in this case. In the NIST manual n=9.

5. Serial Test p48 Section 2.12.4 (5)

   NIST suggest that the p-values be calculated as follows:
   $P - value1 = igamc(2^{m-2}, \nabla \psi_m^2)$ and
   $P - value2 = igamc(2^{m-3}, \nabla^2 \psi_m^2)$
   They should really be calculated by as follows:
   $P - value1 = igamc(2^{m-2}, \frac{\nabla \psi_m^2}{2})$ and
   $P - value2 = igamc(2^{m-3}, \frac{\nabla^2 \psi_m^2}{2})$
   Interestingly, NIST uses the latter in calculating the p-values in the example which suggests that this is yet another typo in the manual and not a statistical blunder.

6. Cumulative Sums Test p53 Section 2.14.4 (14)

   This is not so much a correction as a clarification. It relates to the calculation of the p-value. The range of the summation does not necessarily begin and end with an integer as per usual. Instead of letting k begin with a non-integer k should be rounded up to the nearest integer

while to end with k should be rounded down to the nearest integer. This is not clear from the description in the manual and one is wondering what to sum over. There are numerous other instances where clarification would be helpful.

7. Overlapping Template Matching Test p32 Section 2.8.4 (1)

   It is stated that K=2, where K is the number of degrees of freedom, in the example. This is not consistent with the function call which states that K has been fixed to 5 in the test code nor is it consistent with part (2) of the test description where there are clearly 6 categories and therefore 5 degrees of freedom. In the example NIST do not define m, the length in bits of the template. For the example m is 2 so perhaps this is just yet another typo.

K. DESCRIPTION OF THE NIST TESTS

This appendix describes in detail each of the tests within the NIST statistical test suite. It details the purpose of the test, step-by-step instructions of how the test is carried out, a conclusion and interpretation of the test results, the input size recommendations and a numerical example of how test works in practice.
The order of the applications of the tests in the suite is arbitrary. However, NIST recommend that the Frequency test be run first, since this supplies the most basic evidence for the existence of non-randomness in a sequence, specifically, non-uniformity. If this test fails, the likelihood of other tests failing is high.

Note 1: For many of the examples throughout this section, small sample sizes are used for illustrative purposes only e.g. n=10. The normal approximation is not really applicative in these examples.

Note 2: These descriptions are largely reproduced from the NIST manual and are included here for the convenience of the reader.

1. Frequency (Monobit) Test

1.1 Test Purpose

The focus of this test is the proportion of zeroes and ones for the entire sequence. The purpose of this test is to determine whether the number of ones and zeros in a sequence are approximately the same as would be expected for a truly random sequence. The test assesses the closeness of the fraction of ones to 1/2, that is, the number of ones and zeros in a sequence should be about the same.
Note that if a generator fails this test then the likelihood of other tests failing is high.

1.2 Test Parameters

n The length of the bit string

1.3 Test Description

2. Conversion to +-1. The zeros and ones of the input sequence are converted to values of -1 and +1 and are added together to produce $S_n = X_1 + X_2 + ... + X_n$, where $X_i = 2\varepsilon_i - 1$
3. Compute the test statistic $S_{obs} = \frac{|S_n|}{\sqrt{n}}$
4. Compute $P - value = erfc\left(\frac{S_{obs}}{\sqrt{2}}\right)$, where erfc is the complementary error function (defined in section X)

1.4 Decision Rule

If the computed P-value is <0.01, then conclude that the sequence is non-random. Otherwise conclude that the sequence is random

1.5 Conclusion and Interpretation of Test Results

Note that is the P-value were small (<0.01), then this would be caused by $|S_n|$ or $|S_{obs}|$ being large. Large positive values of S_n is indicative of too many ones, and large negative values of S_n are indicative of too many zeros.

1.6 Input Size Recommendations

NIST recommends that each sequence to be tested consist of a minimum of 100 bits (i.e. $n \geq 100$). This lower bound has been chosen i.e. 100 bits.

1.1 Example

$\varepsilon = 1100100100001111110110101010001000100001011010001100001000110100110001001100011001100010100010111000$

$n = 100$

$S_{100} = 1 + 1 + (-1) + (-1) + 1 + (-1) + (-1) + ... + 1 + (-1) + (-1) + (-1) = -16$

$S_{obs} = \frac{|-16|}{\sqrt{100}} = 1.6$

$P - value = 0.109599$

Since $P - value \geq 0.01$, accept the sequence as random.

2. Frequency Test within a Block

2.1 Test Purpose

The focus of this test is the proportion of ones within M-bit blocks. The purpose of this test is to determine whether the frequency of ones in an M-bit block is approximately M/2, as would be expected under the assumption of randomness.
Note that for block size M=1, this test degenerates to the Frequency (Monobit) test.

2.2 Test Parameters

M	The length of each block
n	The length of the bit string
ε	The sequence of bits as generated by the RNG or PRNG being tested. $\varepsilon \geq n$

2.3 Test Description

1. Partition the sequence into $N = \left(\frac{n}{M} \right)$ non-overlapping blocks. Discard any unused bits.
2. Determine the proportion $\pi_i = \frac{\sum_{j=1}^{M} \varepsilon_{(i-1)M+j}}{M}$, for $1 \le i \le N$.
3. Compute the χ^2 statistic: $\chi^2(obs) = 4M \sum_{i=1}^{N} (\pi_i - 1/2)^2$
4. Compute the P-value = $chidist(\chi^2(obs), df)$, where chidist returns the one-tailed probability of the chi-squared distribution and df is the degrees of freedom (the number of blocks minus 1).

2.4 Decision Rule

If the computed P-value is <0.01, then conclude that the sequence is non-random. Otherwise, conclude that the sequence is random.

2.5 Conclusion and Interpretation of Test Results

Small p-values (<0.01) would have indicated a large deviation from the equal proportion of ones and zeros in at least one of the blocks.

2.6 Input Size Recommendations

NIST recommends that each sequence to be tested consist of a minimum of 100 bits (i.e. $n \ge 100$) and that the block size M should be selected such that $M \ge 20$, $M > 0.1n$ and $N < 100$. The lower bounds of $M = 20$ has been chosen with $N = 50$ and $n = 1000$.

2.7 Example

$\varepsilon = 11001001000011111101101010100010001000010110100011000010001101001100010011000110011000101000010111000$

$n = 100$, $M = 10$

$N = \text{int}\left(\frac{100}{10} \right)$, where int(x) is the integer value of x. (this discards any bits at the end of the sequence being tested that do not make up a full block).

$\varepsilon = 1100100100|0011111101|1010101000|1000100001|0110100011|0000100011|0100110001|0011000110|0110001010|0010111000|$

Block No.	1	2	3	4	5	6	7	8	9	10
Proportion of Ones (π_i)	0.4	0.7	0.4	0.3	0.5	0.3	0.4	0.4	0.4	0.4

$$\chi^2(obs) = 4M\sum_{i=1}^{N}(\pi_i - 1/2)^2 = 4*M*\begin{bmatrix}(0.4-0.5)^2+(0.7-0.5)^2+(0.4-0.5)^2+(0.3-0.5)^2+\\(0.5-0.5)^2+(0.3-0.5)^2+(0.4-0.5)^2+(0.4-0.5)^2+\\(0.4-0.5)^2+(0.4-0.5)^2\end{bmatrix} = 7.2$$

$chidist(7.2,10) = 0.706438$
$chidist(7.2,9) = 0.616305$
Since $P-value \geq 0.01$, accept the sequence as random.

3. Runs Test

3.1 Test Purpose

The focus of this test is the total number of runs in a sequence, where a run is an uninterrupted sequence of identical bits. A run length of k consists of exactly k identical bits and is bounded before and after with a bit of opposite value. The purpose of the runs test is to determines whether the number of runs of ones and zeros of various lengths is ass expected for a random sequence. In particular, this test determines whether the oscillation[14] between such zeroes and ones is too fast or too slow.

3.2 Test Parameters

n The length of the bit string
ε The sequence of bits as generated by the RNG or PRNG being tested. $\varepsilon \geq n$

3.3 Test Description

1. Compute the pre-test proportion π of ones in the input sequence: $\pi = \frac{\sum_j \varepsilon_j}{n}$
2. Determine if the pre-test Frequency test is passed: If it can be shown that $|\pi - 1/2| \geq \tau$, then the Runs test need not be performed.
3. Compute the test statistic $V_n(obs) = \sum_{k=1}^{n-1} r(k) + 1$, where $r(k) = 0$ if $\varepsilon_k = \varepsilon_{k+1}$, and $r(k) = 1$ otherwise.
4. Compute $P-value = erfc\left(\frac{|V_n(obs) - 2n\pi(1-\pi)|}{2\sqrt{2n}\pi(1-\pi)}\right)$

3.4 Decision Rule

If the computed P-value is <0.01, then conclude that the sequence is non-random. Otherwise, conclude that the sequence is random.

[14] An oscillation is considered to be a change from a one to a zero or vice versa.

3.5 Conclusion and Interpretation of Test Results

A large value for $V_n(obs)$ indicates an oscillation in the string which is too fast; a small value would have indicates that the oscillation is too slow. A fast oscillation occurs where there are a lot of changes e.g. 010101010 oscillates with every bit. A sequence with a slow oscillation has fewer runs that would be expected in a random sequence e.g. a a sequence containing 100 ones, followed by 73 zeroes, followed by 127 ones (a total of 300 bits) would have only three runs, whereas 150 runs would be expected.

3.6 Input Size Recommendations

NIST recommends that each sequence to be tested consist of a minimum of 100 bits (i.e. $n \geq 100$). This lower bound of $n = 100$ has been chosen.

3.7 Example

$\varepsilon = 1100100100001111110110101010001000100001011010001100001000110100110$
$0001001100011001100010100010111000$

$n = 100$

$$\pi = \frac{\sum_j \varepsilon_j}{n} = \frac{1+1+0+0+1+....+0+0}{100} = 0.42$$

$|\pi - 1/2| = |0.42 - 0.5| = 0.08$ and $\tau = \frac{2}{\sqrt{n}} = \frac{2}{\sqrt{100}} = 0.2 \quad \Rightarrow |\pi - 1/2| \geq \tau \Rightarrow$ Proceed with test

$V_n(obs) = \sum_{k=1}^{n-1} r(k) + 1$ = (0+1+0+1+1+0+0+……+1+1)+1=52

$$P - value = erfc\left(\frac{|V_n(obs) - 2n\pi(1-\pi)|}{2\sqrt{2n}\pi(1-\pi)}\right) = erfc\left(\frac{|52 - 2(100)(0.42)(1-0.42)|}{2\sqrt{2(100)}(0.42)(1-0.42)}\right) = 0.500798$$

Since $P - value \geq 0.01$, accept the sequence as random.

4. Test for the Longest Run of Ones in a Block

4.1 Test Purpose

The focus of the test is the longest run of ones within M-bit blocks. The purpose of 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 that would be expected in a random sequence. Note that an irregularity ini the expected length of the longest run of ones implies that there is also an irregularity in the expected length of the longest run of zeroes. Therefore, only a test for ones is necessary.

4.2 Test Parameters

n The length of the bit string

ε The sequence of bits as generated by the RNG or PRNG being tested. $\varepsilon \geq n$

M The length of each block. The test code has been preset to accommodate three values for M: M=8, M=128 and M=10^4 in accordance with the following table:

Minimum n	M
128	8
6272	128
750000	10^4

N The number of blocks; selected in accordance with the value of M.

4.3 Test Description

1. Divide the sequence into M-bit blocks
2. Tabulate the frequencies v_i of the longest runs of ones in each block into categories, where each cell contains the number of runs of ones of a given length.
   For values of M supported by the test code, the v_i cells will hold the following counts:

Vi	M=8	M=128	M=10^4
V0	4	<=4	<=10
V1	6	5	11
V2	2	6	12
V3	4	7	13
V4		8	14
V5		>=9	15
V6			>=16

3. Compute $\chi^2(obs) = \sum_{i=0}^{K} \frac{(v_i - N\pi_i)^2}{N\pi_i}$, where the values of π_i are provided in the NIST manual Section 3.4,
   The values of K and N are determined by the value of M in accordance with the following table:

M	K	N
8	3	16
128	5	49
10000	6	75

4. Compute $P-value = igamc(\frac{df}{2},\frac{\chi^2}{2}) = chidist(\chi^2, df)$, where df=K

4.4 Decision Rule

If the computed P-value is <0.01, then conclude that the sequence is non-random. Otherwise, conclude that the sequence is random.

4.5 Conclusion and Interpretation of Test Results

Large values of $\chi^2(obs)$ indicate that the tested sequence has clusters of ones.

4.6 Input Size Recommendations

NIST recommends any of the input sizes detailed in the table in 4.2 above as suitable. The one chosen for this project is $n = 6272$ and $M = 128$.

4.7 Example

$\varepsilon = 1100110000010101011011000100110011100000000000001001001$
$1010101000100010011110101101000000011010111110011001110 0$
1101101100010110010

$n = 128$

$\varepsilon = 11001100|00010101|01101100|01001100|11100000|00000010|01001$
$101|01010001|00010011|11010110|10000000|11010111|11001100|\begin{matrix}11100\\110\end{matrix}|$
$11011000|10110010|$

	Subblock	Max-Run			Subblock	Max-Run
1	11001100	2		9	00010011	2
2	00010101	1		10	11010110	2
3	01101100	2		11	10000000	1
4	01001100	2		12	11010111	3
5	11100000	3		13	11001100	2
6	00000010	1		14	11100110	3
7	01001101	2		15	11011000	2
8	01010001	1		16	10110010	2

$v_o = 4$
$v_1 = 9$
$v_2 = 3$
$v_3 = 0$

$$\chi^2(obs) = \sum_{i=0}^{K} \frac{(v_i - N\pi_i)^2}{N\pi_i} = \frac{(4-16(0.2148))}{16(0.2148)} + \frac{(9-16(0.3672))}{16(0.3672)} + \frac{(3-16(0.2305))}{16(0.2305)} + \frac{(0-16(0.1875))}{16(0.1875} = 4.88$$

$P - value = 0.180609$

Since $P - value \geq 0.01$, accept the sequence as random.

5. Binary Matrix Rank Test

5.1 Test Purpose

The focus of the test is the rank of the disjoint sub-matrices of the entire sequence. The purpose is to check for linear dependence among fixed length substrings of the original sequence. This test also appears in the DIEHARD battery of tests.

5.2 Test Parameters

n The length of the bit string

ε The sequence of bits as generated by the RNG or PRNG being tested. $\varepsilon \geq n$

M The number of rows in each matrix.

Q The number of columns in each matrix

5.3 Test Description

1. Sequentially divide the sequence into M*Q-bit disjoint blocks; there will exist $N = \left[\frac{n}{MQ}\right]$ such blocks. Discarded bits that do not form part of a complete M*Q matrix.

   Each row of the matrix is filled with successive Q-bit blocks of the original sequence ε i.e. the sequence is read from left to right across rows.
2. Determine the binary rank (R_l) of each matrix, where l=1,….,N. The method for determining the rank is described in X.
3. Let $F_M =$ the number of matrices with $R_l = M$ (full rank)

   $F_{M-1} =$ the number of matrices with $R_l = M - 1$ (full rank – 1)

   $N - F_M - F_{M-1} =$ the number of matrices remaining.
4. Compute

   $$\chi^2(obs) = \frac{(F_M - 0.2888N)^2}{0.2888N} + \frac{(F_{M-1} - 0.5776N)^2}{0.5776N} + \frac{(N - F_M - F_{M-1} - 0.1336N)^2}{0.1336N}$$

   These probabilities are explained in Section X.
5. Compute the P-value = $chidist(\chi^2(obs), df)$, where chidist returns the one-tailed probability of the chi-squared distribution and df is the degrees of freedom (here it is 2).

5.4 Decision Rule

If the computed P-value is <0.01, then conclude that the sequence is non-random. Otherwise, conclude that the sequence is random.

5.5 Conclusion and Interpretation of Test Results

Large values of $\chi^2(obs)$ (and hence, small p-values) indicate a deviation of the rank distribution from that corresponding random sequence.

5.6 Input Size Recommendations

The probabilities for M=Q=32 have been calculated and inserted into the test code. Other choices of M and Q may be selected, but the probabilities would need to be calculated. The minimum number of bits to be tested must be such that $n \geq 38MQ$ (i.e., at least 38 matrices are created. For M=Q=32, each sequence tested should consist of a minimum of 38,912 bits. This recommendation by NIST has been adhered to with M=Q=32 and the number of matrices created, N, equal to 38. This requires n=38,192 bits.

5.7 Example

$n = 18$

$M = Q = 3$

$\varepsilon = 010110010010101011$

$N = \left(\frac{n}{3*3} \right) = 2$

The two matrices are $\begin{vmatrix} 010 \\ 110 \\ 010 \end{vmatrix}$ and $\begin{vmatrix} 010 \\ 101 \\ 011 \end{vmatrix}$. Note that the first matrix consists of the first three bits in row 1, the second set of three bits in row 2 and the third set of bits in row 3. The second matrix is similarly constructed using the next nine bits in the sequence. Here, $F_M = F_3 = 1$ (The rank of the second matrix is 3), $F_{M-1} = F_2 = 1$ (The rank of the first matrix is 2) and there is no matrix with lower rank.

6. Discrete Fourier Transform (Spectral Test)

6.1 Test Purpose

The focus of this test is the peak heights in the Discrete Fourier Transform of the sequence. The purpose of 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. The intention is to detect whether the number of peaks exceeding the 95 % threshold is significantly different than 5 %.

6.2 Test Parameters

n The length of the bit string.

e The sequence of bits as generated by the RNG or PRNG being tested

6.3 Test Description

1. The zeros and ones of the input sequence (*e*) are converted to values of –1 and +1 to create the sequence *X = x1, x2, …, xn,* where *xi = 2ei – 1.*
2. Apply a Discrete Fourier transform (DFT) on X to produce: *S = DFT(X)*. A sequence of complex variables is produced which represents periodic components of the sequence of bits at different frequencies.
3. Calculate *M = modulus(S´)* , |*S*'|, where *S´* is the substring consisting of the first *n/2* elements in *S*, and the modulus function produces a sequence of peak heights.
4. Compute *T* = $\sqrt{2.995732274n}$, the 95 % peak height threshold value. Under an assumption of randomness, 95 % of the values obtained from the test should not exceed *T*.
5. Compute N_0 = *.95n/2*. N_0 is the expected theoretical (95 %) number of peaks (under the assumption of randomness) that are less than *T*.
6. Compute N_1 = the actual observed number of peaks in *M* that are less than *T*.
7. Compute $d = \frac{(N_1 - N_0)}{\sqrt{n(.95)(.05)/2}}$
8. Compute $P - value = erfc\left(\frac{|d|}{\sqrt{2}}\right)$

2.6.5 Decision Rule (at the 1 % Level)

If the computed *P-value* is < 0.01, then conclude that the sequence is non-random. Otherwise, conclude that the sequence is random.

2.6.6 Conclusion and Interpretation of Test Results

Since the *P-value* obtained in step 8 of Section 2.6.4 is ³ 0.01 (P-value = 0.123812), the conclusion is that the sequence is random.

29

A *d* value that is too low would indicate that there were too few peaks (< 95 %) below *T*, and too many peaks (more than 5 %) above *T*.

6.6 Input Size Recommendations

It is recommended that each sequence to be tested consist of a minimum of 1000 bits (i.e., *n ³ 1000*).

6.7 Example

For example, if *n = 10* and *e = 1001010011*, then *X = 1, -1, -1, 1, -1, 1, -1, -1, 1, 1*.

N_0 = 4.75

N_1 =4

$$d = \frac{(4 - 4.75)}{\sqrt{10(.95)(.05)/2}} = -1.538968$$

$$P - value = erfc\left(\frac{|-1.538968|}{\sqrt{2}}\right) = 0.123812$$

7. Non-overlapping Template Matching Test

7.1 Test Purpose

The focus of this test is the number of occurrences of pre-specified target strings. The purpose of this test is to detect generators that produce too many occurrences of a given non-periodic (aperiodic) pattern. For this test and the Overlapping Template Matching Test of Section X), an m-bit window is used to search for a specific m-bit pattern. If the pattern is *not* found, the window slides one bit position. It a pattern *is* found, the window is reset to the bit after the found pattern, and the search resumes.

7.2 Test Parameters

n The length of the bit string

ε The sequence of bits as generated by the RNG or PRNG being tested. $\varepsilon \geq n$

m The length in bits of each template. The template is the target string.

B The m-bit template to be matched; B is a string of ones and zeroes (of length m)

M The length in bits of the substring of ε to be tested. (M has been set to 131,072 in the code)

N The number of independent blocks.

7.3 Test Description

1. Partition the sequence into N independent blocks of length M. Discard any bits that are not part of a full block.
2. Let $W_j (j = 1,..., N)$ be the number of times that B (the template) occurs within block j. The search for matches proceeds by creating an m-bit window on the sequence, comparing the bite within that window against the template. If there is *no* match, the window slides over one bit, e.g, if m=3 and the current window contains bits 3 to 5, then the next window will contain bits 4 to 6. If there *is* a match, the window slides over m bits, e.g., if the current (successful) window contains bits 3 to 5, then the next window will contain bits 6 to 8.
3. Under an assumption of randomness, compute the theoretical mean μ and variance σ^2:
$$\mu = \frac{(M - m + 1)}{2^m} \quad \sigma^2 = M\left(\frac{1}{2^m} - \frac{2m-1}{2^{2m}}\right)$$
4. Compute $\chi^2(obs) = \sum_{j=1}^{N} \frac{(w_j - \mu)^2}{\sigma^2}$
5. Compute the P-value = $chidist(\chi^2(obs), df)$, where chidist returns the one-tailed probability of the chi-squared distribution and df is the degrees of freedom (here it is 2). Note that multiple P-values will be computed i.e., one P-value will be computed for each template. For m=9, up to 148 P-values may be computed

7.4 Decision Rule

If the computed P-value is <0.01, then conclude that the sequence is non-random. Otherwise, conclude that the sequence is random.

7.5 Conclusion and Interpretation of Test Results

If the P-value is very small (<0.01), then the sequence has irregular occurrences of the possible template patterns.

7.6 Input Size Recommendations

NIST recommend that the sequence to be tested consist of a minimum of 1000 bits.

7.7 Example

$\varepsilon = 10100100101110010110$, then $n = 20$. If $N = 2$ and $M = 10$, then the two blocks would be 1010010010 and 1110010110.

If $m = 3$ and the template is $B = 001$, then the examination proceeds as follows:

Bit Positions	Block 1		Block 2	
	Bits	W_1	Bits	W_2
1-3	101	0	111	0
2-4	010	0	110	0
3-5	100	0	100	0
4-6	001 (hit)	Increment to 1	001 (hit)	Increment to 1
5-7	Not examined		Not examined	
6-8	Not examined		Not examined	
7-9	001	Increment to 2	011	1
8-10	010 (hit)	2	110	1

Thus $W_1 = 2$ and $W_2 = 1$

$$\mu = \frac{(M - m + 1)}{2^m} = \frac{(10 - 3 + 1)}{2^3} = 1$$

$$\sigma^2 = M\left(\frac{1}{2^m} - \frac{2m-1}{2^{2m}}\right) = 10\left(\frac{1}{2^3} - \frac{2(3)-1}{2^{2(3)}}\right) = 0.0098$$

$$\chi^2(obs) = \sum_{j=1}^{N} \frac{(w_j - \mu)^2}{\sigma^2} = \frac{(2-1)^2}{(0.0098)^2} + \frac{(1-1)^2}{(0.0098)^2} = 10412$$

$P - value = chidist(\chi^2(obs), df) = chidist(10412,1) = 0$ *The chi-square test is not valid here because of the small sample size and is just here for illustrative purposes.

8. Overlapping Template Matching Test

8.1 Test Purpose

The focus of the Overlapping Template Matching test is the number of occurrences of pre-specified target strings. Like the Non-overlapping Template Matching test, this test uses an m-bit

window to search for a specific m-bit pattern. In this test also the window slides one bit position if the pattern is *not* found. However, if the pattern *is* found then the window slides only one bit before resuming the search as opposed to sliding the m-bits.

8.2 Test Parameters

n The length of the bit string

ε The sequence of bits as generated by the RNG or PRNG being tested. $\varepsilon \geq n$

m The length in bits of each template. The template is the target string.

B The m-bit template to be matched; B is a string of ones and zeroes (of length m)

M The length in bits of the substring of ε to be tested. (M has been set to 131,072 in the code)

N The number of independent blocks.

8.3 Test Description

1. Partition the sequence into N independent blocks of length M. Discard any bits that do not form part of a full block.
2. Calculate the number of occurrences of B in each of the N blocks. The search for matches proceeds by creating an m-bit window on the sequence, comparing the bits within that window against B and incrementing a counter when there is a match. The window slides over one bit after each examination, e.g., if m=4 and the first window contains bits 42 to 45, the next window consists of bits 43 to 46. Record the number of occurrences of B in each block by incrementing an array v_i (where i=0,….,5), such that v_0 is incremented where there are no occurrences of B in a substring, v_1 id incremented for one occurrence of B, …and v_5 is incremented for 5 or more occurrences of B.
3. Compute the values for λ and η that will be used to compute the theoretical probabilities π_i corresponding to the classes of v_0: $\lambda = \dfrac{(M-m+1)}{2^m} \qquad \eta = \dfrac{\lambda}{2}$
4. Compute $\chi^2(obs) = \sum_{i=0}^{5} \dfrac{(v_i - N\pi_i)^2}{N\pi_i}$,

   where

   $\pi_0 = 0.367879, \pi_1 = 0.183940, \pi_2 = 0.137955, \pi_3 = 0.099634, \pi_4 = 0.069935 and \pi_5 = 0.140657$

   as computed by the equations specified in X.
5. Compute $P - value = chidist(\chi^2(obs), df)$ where chidist returns the one-tailed probability of the chi-squared distribution and df is the degrees of freedom. df=5

8.4 Decision Rule

If the computed P-value is <0.01, then conclude that the sequence is non-random. Otherwise, conclude that the sequence is random.

8.5 Conclusion and Interpretation of Test Results

Note that if a 2-bit template had been used and the entire sequence had too many 2-bit runs of ones, then: v_5 would have been too large, the test statistic would be too large and the P-value would have been small and a conclusion or non-randomness would have resulted.

8.6 Input Size Recommendations

The values of K, M and N have been chosen such that each sequence to be tested consists of a minimum of 1million bits. Various values of m may be selected, but NIST recommends m=9 or m=10. If other values are desired, choose these values as follows: blah…

8.7 Example

$\varepsilon = 1011101111001011010001110010111011111000010110 1001$

n=50

If K=2, M=10 and N=5, then the five blocks are 1011101111, 0010110100, 0111001011, 1011111000 and 0101101001.

If m=2 and B=11, then the examination of the first block 1011101111 proceeds as follows:

Bit Positions	Bits	No. of occurrences
1-2	10	0
2-3	01	0
3-4	11 (hit)	Increment to 1
4-5	11 (hit)	Increment to 2
5-6	10	2
6-7	01	2
7-8	11 (hit)	Increment to 3
8-9	11 (hit)	Increment to 4
9-10	11 (hit)	Increment to 5

Thus, after block1, there are five occurrences of 11, v_5 is incremented, and $v_0 = 0, v_1 = 0, v_2 = 0, v_3 = 0, v_4 = 0$ and $v_5 = 1$.

In a like manner, blocks 2-5 are examined. In block 2, there are 2 occurrences of 11; v_2 is incremented. In block 3, there are 3 occurrences of 11; v_3 is incremented. In block 2, there are 4 occurrences of 11; v_4 is incremented. In block 5, there is one occurrence of 11; v_1 is incremented.

$$\lambda = \frac{(M - m + 1)}{2^m} = \frac{10 - 2 + 1}{2^2} = 2.25 \text{ and } \eta = \frac{\lambda}{2} = \frac{2.25}{2} = 1.125$$

$$\chi^2(obs) = \sum_{i=0}^{5} \frac{(v_i - N\pi_i)^2}{N\pi_i} = \frac{(0-5*0.367879)^2}{5*0.367879} + \frac{(1-5*0.183940)^2}{5*0.183940} + \frac{(1-5*0.137955)^2}{5*0.137955}$$
$$+ \frac{(1-5*0.099634)^2}{5*0.099634} + \frac{(1-5*0.069935)^2}{5*0.069935} + \frac{(1-5*0.140657)^2}{5*0.140657}$$

9. Maurer's "Universal Statistical" Test

9.1 Test Purpose

The focus of this test is the number of bits between matching patterns (a measure that is related to the length of a compressed sequence). The purpose of the test is to detect whether or not the sequence can be significantly compressed without loss of information. A significantly compressible sequence is considered to be non-random.

9.2 Test Parameters

L The length if each block.

n The length of the bit string

ε The sequence of bits as generated by the RNG or PRNG being tested. $\varepsilon \geq n$

9.3 Test Description

1. The n-bit sequence is partitioned into two segments; an initialisation segment consisting of Q L-bit non-overlapping blocks, and a test segment consisting of K L-bit non-overlapping blocks. Bits remaining at the end of the sequence that do not form a complete L-bit block are discarded. The first Q blocks are used to initialise the test. The remaining K blocks are the test blocks ($K = \left\lfloor \frac{n}{L} \right\rfloor - Q$)
2. Using the initialisation segment, a table is created for each possible L-bit value (i.e., the L-bit value is used as an index into the table). The block number of the last occurrence of each L-bit block is noted in the table (i.e., For i from 1 to Q, Tj=I, where j is the decimal representation of the contents of the ith L-bit block)
3. Examine each of the K blocks in the test segment and determine the number of blocks since the last occurrence of the same L-bit block (i.e., $i - T_j$) . Replace the value in the table with the location of the current block (i.e, $T_j = i$) . Add the calculated distance between re-occurrences of the same L-bit block to an accumulating $\log_2$ sum of all the differences detected in the K blocks (i.e., $sum = sum + \log_2(i - T_j)$)
4. Compute the test statistic: $f_n = \frac{1}{K} \sum_{i=Q+1}^{Q+K} \log_2(i - T_j)$, where T_j is the table entry corresponding to the decimal representation of the contents of the ith L-bit block.

5. Compute $P-value = erfc\left(\left|\frac{f_n - \exp ectedValue(L)}{\sqrt{2}\sigma}\right|\right)$, where erfc is the complementary error function, $\exp ectedValue(L)$ and σ are taken from a table of pre-computed values from the Handbook of Applied Cryptography.

$\sigma = c\sqrt{\frac{\text{var}\, iance(L)}{K}}$, where $c = 0.7 - \frac{0.8}{L} + \left(4 + \frac{32}{L}\right)\frac{K^{-3/L}}{15}$

While it is possible to conduct this test on values of L from 6 to 16, the lower bound of 6 is chosen.

$\exp ectedValue(6) = 5.2177052$

variance when L=6 = 2.954

9.4 Decision Rule

If the computed P-value is <0.01, then conclude that the sequence is non-random. Otherwise, conclude that the sequence is random.

9.5 Conclusion and Interpretation of Test Results

If f_n differs significantly from $\exp ectedValue(L)$, then the sequence is significantly compressible.

9.6 Input Size Recommendations

NIST gives recommendations of what combination of n, L and Q should be chosen. The lower bound of the recommendations has been chosen and are as follows:

N	L	Q=10*2^L
≥387,840	6	640

9.7 Example

$\varepsilon = 01011010011101010111$, n=20

If L=2 and Q=4, then K=[n/L]-Q=[20/2]-4=6. The initialisation segment is 01011010. The L-bit blocks are shown on the following table:

Block	Type	Contents
1	Initialisation Segment	01
2		01
3		10
4		10
5	Test Segment	01
6		11
7		01
8		01
9		01
10		11

The following table is created using the 4 initialisation blocks:

	Possible L-bit Value			
	00 (saved in T_0)	01 (saved in T_1)	10 (saved in T_2)	11 (saved in T_3)
Initialisation	0	2	4	0

For block 5 (the 1st test block): 5 is placed in the "01" row of the table (i.e., T_1), and

$sum = \log_2(5-2) = 1.584962501$

For block 6: 6 is placed in the "11" row of the table (i.e., T_3), and

$sum = 1.584962501 + \log_2(6-0) = 1.584962501 + 2.584962501$

.....

For block 10: 10 is replaced in the "11" row of the table (i.e., T_3), and

$sum = 5.169925002 + \log_2(10-6) = 5.169925002 + 2 = 7.169925002$

The states of the table are

Iteration Block	Possible L-bit Value			
	00	01	10	11
4	0	2	4	0
5	0	5	4	0
6	0	5	4	6
7	0	7	4	6
8	0	8	4	6
9	0	9	4	6
10	0	9	4	10

$$f_n = \frac{7.169925002}{6} = 1.1949875$$

$$P - value = erfc\left(\left|\frac{1.1949875 - 1.5374383}{\sqrt{2\sqrt{1.338}}}\right|\right) = 0.767189$$

(Note that the expected value and variance for L=2 is not provided in the NIST manual because a block length of 2 is not recommended for testing.

10. Linear Complexity Test

10.1 Test Purpose

The focus of this test is the length of a linear feedback shift generator (LFSR). The purpose of this test is to determine whether or not the sequence is complex enough to be considered random. Random sequences are characterised by longer LFSRs. A LFSR that is too short implies non-randomness.

10.2 Test Parameters

M length in bits of a block

N length of the bit string

K the number of degrees of freedom

ε The sequence of bits as generated by the RNG or PRNG being tested. $\varepsilon \geq n$

10.3 Test Description

1. Partition the n-bit sequence into N independent blocks of M bits, where n=MN

2. Using the Berlekamp-Massey algorithm, determine the linear complexity Li of each of the N blocks (i=1,….,N). Li is the length of the shortest linear feedback shift register sequence that generates all bits in the block i. Within any li-bit sequence, some combination of the bits, when added together modulo 2, produces the next bit in the sequence (bit Li+1)

3. Under an assumption of randomness calculate the theoretical mean μ :

$$\mu = \frac{M}{2} + \frac{(9 + (-1)^{M+1})}{36} - \frac{(M/3 + 2/9)}{2^M}$$

4. For each substring, calculate a value of T_i , where $T_i = (-1)^M * (L_i - \mu) + \frac{2}{9}$

5. Record the T_i values in $v_0, ..., v_6$ as follows:

If:	$T_i \le -2.5$	Increment v_0 by one
	$-2.5 < T_i \le -1.5$	Increment v_1 by one
	$-1.5 < T_i \le -0.5$	Increment v_2 by one
	$-0.5 < T_i \le 0.5$	Increment v_3 by one
	$0.5 < T_i \le 1.5$	Increment v_4 by one
	$1.5 < T_i \le 2.5$	Increment v_5 by one
	$T_i > 2.5$	Increment v_6 by one

6. Compute $\chi^2(obs) = \sum_{i=0}^{K} \frac{(v_i - N\pi_i)^2}{N\pi_i}$, where
   $\pi_0 = 0.01047, \pi_1 = 0.03125, \pi_2 = 0.125, \pi_3 = 0.5, \pi_4 = 0.25, \pi_5 = 0.0625, \pi_6 = 0.02078$
   are the probabilities hardcoded (equations given in the manual).
7. Compute $P - value = igamc\left(\frac{K}{2}, \frac{\chi^2(obs)}{2}\right)$

10.4 Decision Rule

If the computed P-value is <0.01, then conclude that the sequence is non-random. Otherwise, conclude that the sequence is random.

10.5 Conclusion and Interpretation of Results

If the P-value were <0.01, this would have indicated that the observed frequency counts of T_i stored in the vi bins varied from the expected values.

10.6 Input Size

NIST recommends that $n \ge 10^6$, while the value of M must be in the range $500 \le M \le 5000$, and $N \ge 200$. This is so that the χ^2 result is valid. N=1000000, M=500 and N=1000 has been chosen for this project.

10.7 Example

If M=13 and the block to be tested is 1101011110001, then Li=4. The sequence is produced by adding the 1st and 2nd bits within a 4-bit sequence to produce the next bit (the 5th bit). The examination proceeded as follows:

	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5
The first 4 bits and the resulting 5th bit:	1	1	0	1	0
Bits 2-5 and the resulting 6th bit:	1	0	1	0	1
Bits 3-6 and the resulting 7th bit:	0	1	0	1	1
.	1	0	1	1	1
.	0	1	1	1	1
.	1	1	1	1	0
.	1	1	1	0	0
.	1	1	0	0	0
Bits 9-12 and the resulting 13th bit	1	0	0	0	1

$$\mu = \frac{13}{2} + \frac{(9 + (-1)^{13+1})}{36} - \frac{(13/3 + 2/9)}{2^{13}} = 6.777222$$

$$T_i = (-1)^M * (L_i - \mu) + \frac{2}{9} = 2.999444$$

11. Serial Test

11.1 Test Purpose

The focus of this test is the frequency of all possible overlapping m-bit patters across the entire sequence. The purpose of this test is to determine whether the number of occurrences of the 2^m m-bit overlapping patters s approximately the same as would be expected for a random sequence. Random sequences have uniformity; that is, every m-bit pattern has the same chance of appearing as every other m-bit pattern. Note that for m=1, the serial test is equivalent to the frequency test.

11.2 Test Parameters

m The length in bits of each block

n The length in bits of the bit string

ε The sequence of bits as generated by the RNG or PRNG being tested. $\varepsilon \geq n$

11.3 Test Description

1. Extend the sequence by appending the first m-1 bits to the end of the sequence for distinct values of n.
2. Determine the frequency of all possible overlapping m-bit blocks, all possible overlapping (m-1)-bit blocks and all possible overlapping (m-2)-bit blocks. Let $v_{i_1}...v_{i_m}$ denote the frequency of the m-bit pattern $i_{1....}i_m$; let $v_{i_1}...,v_{i_{m-1}}$ denote the frequency of the (m-1)-bit pattern $i_{1....}i_{m-1}$; and let $v_{i_1}...,v_{i_{m-2}}$ denote the frequency of the (m-2)-bit pattern $i_{1....}i_{m-2}$

3. Compute $\psi_m^2 = \frac{2^m}{n}\sum_{i_1 \ldots i_m}(v_{i_1 \ldots i_m} - \frac{n}{2^m})^2 = \frac{2^m}{n}\sum_{i_1 \ldots . i_m} v^2{}_{i_1 \ldots i_m} - n$

$$\psi_{m-1}^2 = \frac{2^{m-1}}{n}\sum_{i_1 \ldots i_{m-1}}(v_{i_1 \ldots i_{m-1}} - \frac{n}{2^{m-1}})^2 = \frac{2^{m-1}}{n}\sum_{i_1 \ldots . i_{m-1}} v^2{}_{i_1 \ldots i_{m-1}} - n$$

$$\psi_{m-2}^2 = \frac{2^{m-2}}{n}\sum_{i_1 \ldots i_{m-2}}(v_{i_1 \ldots i_2} - \frac{n}{2^{m-2}})^2 = \frac{2^{m-2}}{n}\sum_{i_1 \ldots . i_{m-2}} v^2{}_{i_1 \ldots i_{m-2}} - n$$

4. Compute $\nabla\psi_m^2 = \psi_m^2 - \psi_{m-1}^2$, and

$$\nabla^2\psi_m^2 = \psi_m^2 - 2\psi_{m-1}^2 + \psi_{m-2}^2$$

5. Calculate

$P - value1 = igamc(2^{m-2}, \frac{\nabla\psi_m^2}{2})$ and

$P - value2 = igamc(2^{m-3}, \frac{\nabla^2\psi_m^2}{2})$

11.4 Decision Rule

If the computed P-value is <0.01, then conclude that the sequence is non-random. Otherwise, conclude that the sequence is random.

11.5 Conclusion and Interpretation of Results

If $\nabla^2\psi_m^2$ or $\nabla\psi_m^2$ is large then non-uniformity of the m-bit blocks is implied.

11.6 Input Size Recommendations

Choose m and n such that $m < [\log_2 n] - 2$

11.7 Example

$\varepsilon = 0011011101$

n=10

If m=3, then $\varepsilon' = 001101110100$

If m=2, then $\varepsilon' = 00110111010$

If m=1, then $\varepsilon' = 0011011101$ (the original sequence)

The frequency of all 3-bit blocks is:

$v_{000} = 0, v_{001} = 1, v_{010} = 1, v_{011} = 2, v_{100} = 1, v_{101} = 2, v_{110} = 2, v_{111} = 1.$

The frequency of all 2-bit blocks is:

$v_{00} = 1, v_{01} = 3, v_{10} = 3, v_{11} = 3.$

The frequency of all 1-bit blocks is:

$v_0 = 4, v_1 = 6.$

$$\psi_3^2 = \frac{2^3}{10}(0+1+1+4+1+4+4+4+1) - 10 = 12.8 - 10 = 2.8$$
$$\psi_2^2 = \frac{2^2}{10}(1+9+9+9) - 10 = 11.2 - 10 = 1.2$$
$$\psi_1^2 = \frac{2}{10}(16+36) - 10 = 10.4 - 10 = 0.4$$

$$\nabla\psi_3^2 = \psi_3^2 - \psi_2^2 = 2.8 - 1.2 = 1.6$$
$$\nabla^2\psi_3^2 = \psi_3^2 - 2\psi_2^2 + \psi_1^2 = 2.8 - 2(1.2) + 0.4 = 0.8$$

$$P-value1 = igamc(2, \frac{1.6}{2}) = chidist(1.6,4) = 0.808792$$
$$P-value2 = igamc(1, \frac{0.8}{2}) = chidist(0.8,2) = 0.67032$$

12. Approximate Entropy Test

12.1 Test Purpose

As with the Serial test, the focus of this test is the frequency of all possible overlapping m-bit patters across the entire sequence. The purpose of the test is to compare that frequency of overlapping blocks of two consecutive/adjacent lenths (m and m+1) against the expected result for a random sequence.

12.2 Test Parameters

m The length of each block – in this case, the first block length used in the test. m+1 is the second block length used.

n The length in bits of the bit string

ε The sequence of bits as generated by the RNG or PRNG being tested. $\varepsilon \geq n$

12.3 Test Description

1. Augment the n-bit sequence to create n overlapping m-bit sequences by appending m-1 bits from the beginning of the sequence to the end of the sequence.
2. A frequency count is made of the n overlapping blocks (e.g. if a block containing ε_j to ε_{j+m-1} is examined at time j, then the block containing ε_{j+1} to ε_{j+m} is examined at time j+1). Let the count of the possible m-bit ((m+1)-bit) values be represented as C_i^m where I is the m-bit value.
3. Compute $C_i^m = \frac{\#i}{n}$ for each value of i

4. Compute $\varphi^{(m)} = \sum_{i=0}^{2^m-1} \pi_i \log \pi_i$, where $\pi_i = C_j^3$, and $j = \log_2 i$
5. Repeat steps 1-4, replacing m by m+1
6. Compute the test statistic $\chi^2 = 2n[\log 2 - ApEn(m)]$, where $ApEn(m) = \varphi^{(m)} - \varphi^{(m+1)}$.
7. Compute $P - value = igamc(2^{m-1}, \frac{\chi^2}{2})$

12.4 Decision Rule

If the computed P-value is <0.01, then conclude that the sequence is non-random. Otherwise, conclude that the sequence is random.

12.5 Conclusion and Interpretation of Results

Small values of ApEn(m) would imply stong regularity. Large values would imply substantial fluctuation or irregularity.

12.6 Input Size Recommendations

Choose m and n such that $m < [\log_2 n] - 2$

12.7 Example

$\varepsilon = 0100110101$

$\varepsilon' = 010011010101$ for m=3

The overlapping m-bit blocks (where m=3) become 010, 100, 001, 011, 110, 101, 010, 101, 010 and 101. The calculated counts for the $2^m = 2^3 = 8$ possible m-bit strings are:

$\#000 = 0, \#001 = 1, \#010 = 3, \#100 = 1, \#011 = 1, \#110 = 1, \#101 = 3, \#111 = 0$

$C_{000}^3 = 0, C_{001}^3 = 0.1, C_{010}^3 = 0.3, C_{100}^3 = 0.1, C_{011}^3 = 0.1, C_{110}^3 = 0.1, C_{101}^3 = 0.3, C_{111}^3 = 0$

$\varphi^3 = 0(\log 0) + 0.1(\log 0.1) + ... + 0.3(\log 0.3) + 0(\log 0) = -1.64341772$

$\varepsilon' = 0100110101010$ for m=4

The overlapping m-bit blocks (where m=3) become 0100, 1001, 0011, 0110, 1101, 1010, 0101, 1010, 0101 and 1010. The calculated counts for the $2^m = 2^4 = 16$ possible m-bit strings are:

$\#0011 = 1, \#0100 = 1, \#0101 = 2, \#0110 = 1, \#1010 = 3, \#1101 = 1, \#1001 = 1$ and all other patterns are zero.

$C_{0011}^4 = 0.1, C_{0100}^4 = 0.1, C_{0101}^4 = 0.2, C_{0110}^4 = 0.1, C_{1010}^4 = 0.3, C_{1101}^4 = 0.1, C_{1001}^4 = 0.1$ and all other values are zero.

$\varphi^4 = 0.1(\log 0.1) + 0.1(\log 0.1) + ... + 0.3(\log 0.3) + 0.1(\log 0.1) = -1.83437197$

$ApEn(3) = \varphi^{(3)} - \varphi^{(4)} = -1.64341772 - (-1.83437197) = 0.190954$

$\chi^2 = 2(10)[\log 2 - ApEn(3)] = 2(10)(0.693147 - 0.190954) = 0.502193$

$P - value = igamc(2^{m-1}, \frac{\chi^2}{2}) = igamc(4, \frac{0.502193}{2}) = chidist(0.502193,8) = 0.99$

13. Cumulative Sums (Cusum) Test

13.1 Test Purpose

The focus of this test is the maximal excursion (from zero) of the random walk defined by the cumulative sum of adjusted (-1,+1) digits in the sequence. The purpose of the test is to determine whether the cumulative sum of the partial sequences occurring in the tested sequence is too large or too small relative to the expected behaviour of that cumulative sum for random sequences. This cumulative sum may be considered as a random walk. For a random sequence, the excursions of the random walk should be near zero. For certain types of non-random sequences, the excursions of this random walk from zero will be large.

13.2 Test Parameters

n The length in bits of the bit string

ε The sequence of bits as generated by the RNG or PRNG being tested. $\varepsilon \geq n$

Mode A switch for applying the test either forward through the input sequence (mode=0) or backward through the sequence (mode=1).

13.3 Test Description

1. Form a normalised sequence: The zeroes and the ones of the input sequence, ε, are converted to values of X_i of -1 and +1 using $X_i = 2\varepsilon_i - 1$.
2. Compute the partial sums S_i of successively larger subsequences, each starting with X_1 (if mode=0) or X_n (if mode=1).

Mode =0 (forward)	Mode=1 (backward)
$S_1 = X_1$	$S_1 = X_n$
$S_2 = X_1 + X_2$	$S_2 = X_n + X_{n-1}$
$S_3 = X_1 + X_2 + X_3$	$S_3 = X_n + X_{n-1} + X_{n-2}$
.	.
.	.
$S_k = X_1 + X_2 + X_3 + ... + X_k$	$S_k = X_n + X_{n-1} + X_{n-2} + ... + X_{n-k+1}$
.	.
.	.
$S_n = X_1 + X_2 + X_3 + ... + X_k + ... + X_n$	$S_n = X_n + X_{n-1} + X_{n-2} + ... + X_{n-k+1} + ... + X_1$

That is, $S_k = S_{k-1} + X_k$ for mode 0, and $S_k = S_{k-1} + X_{n-k+1}$ for mode=1.

3. Compute the test statistic $z = \max_{1 \leq k \leq n} |S_k|$, where $\max_{1 \leq k \leq n} |S_k|$ is the largest of the absolute values of the partial sums S_k.

4. Compute P-value

$$= 1 - \sum_{k=\left(\frac{-n}{z}+1\right)/4}^{\left(\frac{n}{z}-1\right)/4} \left[\Phi\left(\frac{(4k+1)z}{\sqrt{n}} \right) - \Phi\left(\frac{(4k-1)z}{\sqrt{n}} \right) \right] +$$

$$\sum_{k=\left(\frac{-n}{z}-3\right)/4}^{\left(\frac{n}{z}-1\right)/4} \left[\Phi\left(\frac{(4k+3)z}{\sqrt{n}} \right) - \Phi\left(\frac{(4k+1)z}{\sqrt{n}} \right) \right]$$

where Φ is the standard normal cumulative probability distribution function.

13.4 Decision Rule

If the computed P-value is <0.01, then conclude that the sequence is non-random. Otherwise, conclude that the sequence is random.

13.5 Conclusion and Interpretation of Results

When mode=0, large values of this statistic indicate that there are either “too many ones” or “too many zeroes” at early stages of the sequence; mode=1, large values of this statistic indicate that there are either “too many ones” or “too many zeroes” at the late stages. Small values of the statistic would indicate that ones and zeros are intermixed too evenly.

13.6 Input Size Recommendations

NIST recommends that each sequence to be tested consist of a minimum of 100 bits.

13.7 Example

$\varepsilon = 1011010111$

$X = 1,(-1),1,1,(-1),1,(-1),1,1,1$

When mode=0 then

$S_1 = 1$

$S_2 = 1 + (-1) = 0$

$S_3 = 1 + (-1) + 1 = 1$

$S_4 = 1 + (-1) + 1 + 1 = 2$

$S_5 = 1 + (-1) + 1 + 1 + (-1) = 1$

$S_6 = 1 + (-1) + 1 + 1 + (-1) + 1 = 2$

$S_7 = 1 + (-1) + 1 + 1 + (-1) + 1 + (-1) = 1$

$S_8 = 1 + (-1) + 1 + 1 + (-1) + 1 + (-1) + 1 = 2$

$S_9 = 1 + (-1) + 1 + 1 + (-1) + 1 + (-1) + 1 + 1 = 3$

$S_{10} = 1 + (-1) + 1 + 1 + (-1) + 1 + (-1) + 1 + 1 + 1 = 4$

z=4

P-value=0.4116588

14. Random Excursions Test

14.1 Test Purpose

The focus of this test is the number of cycles having exactly K visits in a cumulative sum random walk. The cumulative sum random walk is derived from partial sums after the (0,1) sequence is transferred to the appropriate (-1,1) sequence. A cycle of a random walk consists of a sequence of steps of unit length taken at random that begin at and return to the origin. The purpose of this test is to determine if the number or visits to a particular state within a cycle deviates from what one would expect for a random sequence. This test is actually a series of eight tests (and eight conclusions), one test and one conclusion for each of the states: -4, -3, -2, -1 and +1, +2, +3, +4

14.2 Test Parameters

n The length in bits of the bit string

ε The sequence of bits as generated by the RNG or PRNG being tested. $\varepsilon \geq n$

14.3 Test Description

1. Form a normalised (-1,+1) sequence X: The zeroes and ones of the input sequence (ε) are changed to values of -1 and +1 via $X_i = 2\varepsilon_i - 1.$
2. Compute the partial sums S_i of successively large subsequences, each starting with X_1. For the set $S = \{S_i\}.$

$S_1 = X_1$

$S_2 = X_1 + X_2$

$S_3 = X_1 + X_2 + X_3$

.

.

$S_k = X_1 + X_2 + X_3 + ... + X_k$

.

.

$S_n = X_1 + X_2 + X_3 + ... + X_k + ... + X_n$

3. Form a new sequence S' by attaching zeroes before and after S. That is $S' = 0, s_1, s_2, ..., s_n, 0$
4. Let J = the total number of zero crossings in S', where a zero crossing is a value of zero in S' that occurs after the starting zero. J is also the number of cycles in S', where a cycle of S' is a subsequence consisting of an occurrence of zero, followed by no-zero values, and ending with another zero. The ending zero in one cycle may be the beginning zero in another cycle. The number of cycles in S' is the number of zero crossings. If J<500, discontinue the test.
5. For each cycle and for each non-zero state value x having values $-4 \leq x \leq -1$ and $1 \leq x \leq 4$, compute the frequency of each x within each cycle.

6. For each of the eight states of x, compute $v_k(x) =$ the total number of cycles in which state x occurs exactly k times among all cycles, for k=0, 1, …, 5 (for k=5, all frequencies ≥ 5 are stored in $v_5(x)$). Note that $\sum_{k=0}^{5} v_k(x) = J$.

7. For each of the eight states of x, compute the test statistic $\chi^2(obs) = \sum_{k=0}^{5} \frac{(v_k(x) - J\pi_k(x))^2}{J\pi_k(x)}$, where $\pi_k(x)$ is the probability that the state x occurs k times in a random distribution. The values for $\pi_k(x)$ and their method of calculation are provided in the NIST manual. Note that the eight χ^2 statistics will be produced (i.e., for x=-4, -3, -2, -1, 1, 2, 3, 4)

8. For each state of x, compute $P - value = igamc(\frac{5}{2}, \frac{\chi^2}{2})$. Eight p-values will be produced.

14.4 Decision Rule

If the computed P-value is <0.01, then conclude that the sequence is non-random. Otherwise, conclude that the sequence is random.

14.5 Conclusion and Interpretation of Results

If χ^2 (obs) were too large, then the sequence would have displayed a deviation from the theoretical distribution for a given state across all cycles.

14.6 Input size recommendations

NIST recommend that each sequence to be tested consist of a minimum of 1,000,000bits.

14.7 Example

$\varepsilon = 0110110101$

$X = -1,1,1,-1,1,1,-1,1,-1,1$

$S = \{-1,0,1,0,1,2,1,2,1,2\}$

$S' = 0,-1,0,1,0,1,2,1,2,1,2,0$

$J = 3$

The 3 cycles are {0,-1,0}, {0,1,0}, {0,1,2,1,2,1,2,0}

State x	Cycles		
	{0,-1,0}	{0,1,0}	{0,1,2,1,2,1,2,0}
-4	0	0	0
-3	0	0	0
-2	0	0	0
-1	1	0	0
1	0	1	3
2	0	0	3
3	0	0	0
4	0	0	0

$v_0(-1) = 2$ (the -1 state occurs exactly 0 times in 2 cycles),
$v_1(-1) = 1$ (the -1 state occurs only once in 1 cycle) and
$v_2(-1) = v_3(-1) = v_4(-1) = v_5(-1) = 0$ (the -1 state occurs exactly {2,3,4,$\geq$5} times in 0 cycles.
And so on for each state…

This can be shown using the following table:

State x	Number of Cycles					
	0	1	2	3	4	5
-4	3	0	0	0	0	0
-3	3	0	0	0	0	0
-2	3	0	0	0	0	0
-1	2	1	0	0	0	0
1	1	1	0	1	0	0
2	2	0	0	1	0	0
3	3	0	0	0	0	0
4	3	0	0	0	0	0

15. Random Excursions Variant Test

15.1 Test Purpose

The focus of this test is the total number of times that a particular state is visited (i.e., occurs) in a cumulative sum random walk. The purpose of this test is to detect deviations from the expected number of visits to various states in the random walk. This test is actually a series of eighteen tests (and conclusions), one test and conclusion for each of the states: -9, -8, …, -1 and +1, +2, …, +9.

15.2 Test Parameters

n The length in bits of the bit string

ε The sequence of bits as generated by the RNG or PRNG being tested. $\varepsilon \geq n$

15.3 Test Description

1. Form the normalized (-1, +1) sequence *X* in which the zeros and ones of the input sequence (*e*) are converted to values of –1 and +1 via *X = X1, X2, … , Xn,* where *Xi = 2ei– 1.*
2. Compute partial sums *Si* of successively larger subsequences, each starting with *x1*. Form the set *S* = {*Si*}.

S1 = X1
S2 = X1 + X2
S3 = X1 + X2 + X3
.
.
.
Sk = X1 + X2 + X3 + . . . + Xk
.
.
Sn = X1 + X2 + X3 + . . . + Xk + . . .+ Xn

3. Form a new sequence *S*' by attaching zeros before and after the set *S*. That is, *S' = 0, s1,s2, … , sn*, 0.

4.

5. For each of the eighteen non-zero states of *x*, compute $\xi(x)$ = the total number of times that state *x* occurred across all *J* cycles.

(5) For each $\xi(x)$, compute *P-value* = $erfc\left(\frac{|\xi(x) - J|}{\sqrt{2J(4|x| - 2}}\right)$. Eighteen *P-values* are computed.

15.4 Decision Rule

If the computed *P-value* is < 0.01, then conclude that the sequence is non-random. Otherwise, conclude that the sequence is random.

15.5 Input Size Recommendations

It is recommended that each sequence to be tested consist of a minimum of 1,000,000 bits (i.e.,*n ³ 106*).

15.6 Example

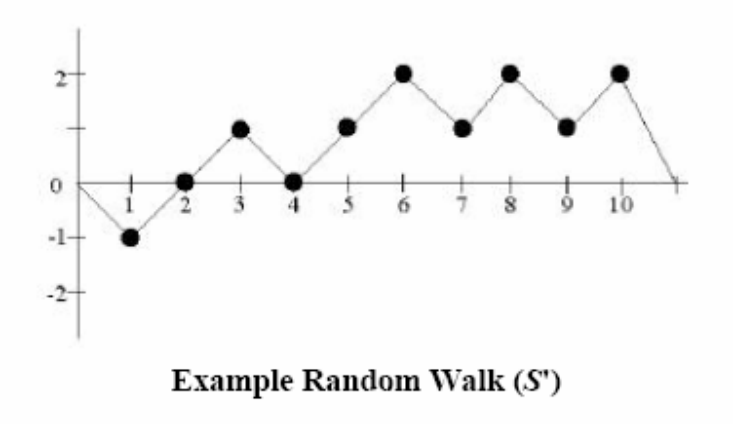
ε = 0110110101, then *n = 10* and *X = -1, 1, 1, -1, 1, 1, -1, 1, -1, 1*.

For the example in this section,

S1 = -1 *S6 = 2*
S2 = 0 *S7 = 1*

S3 = 1 *S8 = 2*
S4 = 0 *S9 = 1*
S5 = 1 *S10 = 2*

The set S = {-1, 0, 1, 0, 1, 2, 1, 2, 1,2}
For the example, *S' = 0, -1, 0, 1, 0, 1, 2, 1, 2, 1, 2, 0*. The resulting random walk is shown below.



Example Random Walk (*S'*)

$\xi(-1) = 1, \xi(1) = 4, \xi(2) = 3$ and all other $\xi(x) = 0$

when *x* = 1, *P-value* = $erfc\left(\frac{|4-3|}{\sqrt{2*3(4|1|-2}}\right) = 0.683091$

L. INPUT SIZES

Table L.1 shows the input sizes and parameter setting that were used in the suite to test the random number generators.

		Input Size (n)	Other Parameters
1	Frequency (Monobit) test	100	
2	Frequency test within a block	2000	M=20, N=100
3	Runs test	100	
4	Test for the longest run of ones in a block	6272	M=128
5	Binary matrix rank test	38912	
6	Discrete fourier transform (spectral) test	1024	
7	Non-overlapping template matching test	1048576	m=9, B=111111111
8	Overlapping template matching test	998976	m=9, M=1032, N=968
9	Maurer's universal statistical test	387840	L=6, Q=640
10	Linear Complexity test	1000000	M=500, N=1000
11	Serial test	500	m=5, n=500
12	Approximate Entropy test	500	m=5, n=500
13	Cumulative sums (cusum) test	100	
14	Random Excursions Test	1000000	
15	Random Excursions Variants Test	1000000	

L.1 Input sizes and parameter settings

M. HOW THE NUMBERS WERE GENERATED

This appendix gives an overview of how the numbers are generated by the five generators examined in this project as well as how the numbers were extracted for testing.

Random.org

A radio is tuned into a frequency where nobody is broadcasting. The atmospheric noise picked up by the receiver is fed into a Sun SPARC workstation through the microphone port where it is sampled by a program as an eight bit mono signal at a frequency of 8KHz. The upper seven bits of each sample are discarded immediately and the remaining bits are gathered and turned into a stream of bits with a high content of entropy. Skew correction is performed on the bit stream, in order to ensure that there is an approximately even distribution of 0s and 1s.

The skew correction algorithm used is based on transition mapping. Bits are read two at a time, and if there is a transition between values (the bits are 01 or 10) one of them - say the first - is passed on as random. If there is no transition (the bits are 00 or 11), the bits are discarded and the next two are read. This simple algorithm was originally due to Von Neumann and completely eliminates any bias towards 0 or 1 in the data. It is only one of several ways of performing skew correction, though, and has a number of drawbacks. First, it takes an indeterminate number of input bits. Second, it is quite inefficient, resulting in the loss of 75% of the data, even when the bit stream is already unbiased. [ref – random.org]

The Random.org numbers that were used in the tests are on the disc attached to the inside of the back cover of the report.

Excel

The Excel application contained within Windows-XP was used to generate the numbers that were subjected to the statistical tests.

The RANDBETWEEN function in Excel returns a random integer between specified numbers. To generate binary random numbers the formula RANDBETWEEN(0,1) was used. Although it was not possible to verify it is thought that the RANDBETWEEN function calls on the RAND function in a manner similar to the following:

- Call RANDBETWEEN(a,b)
- RAND()*(b+1-a)+a is calculated. This will give a random number between a and b+1 but it will not necessarily be integer. Note the RAND() returns a uniform number between 0 and 1.
- To make the number integer the fractional part is truncated.

Essentially, the random number is generated with the RAND() function and a transformation made to the RAND() output. And so, how RAND() generates numbers needs to be identified.

The inadequacy of the random number generation in Excel pre-2003 was much publicised in the literature [X, X and X]. The RAND() function in earlier versions of Excel used a pseudo-random number generation algorithm whose performance on standard tests of randomness was not sufficient. Although this is likely to affect only those users who have to make a large number of calls to RAND, such as a million or more, the pseudo-random number generation algorithm that is described below was implemented for Excel 2003. [64]
The basic idea behind this RNG is to generate three streams of random numbers (in columns headed "ix", "iy", and "iz") by a common technique and then to use the result that if you take three random numbers on [0,1] and sum them, the fractional part of the sum is itself a random number on [0,1]. The critical statements in the Fortran code listing from the original Wichman and Hill article who developed the algorithm are:

```
IX, IY, IZ SHOULD BE SET TO INTEGER VALUES BETWEEN 1 AND 30000 BEFORE FIRST
ENTRY
IX = MOD(171 * IX, 30269)
IY = MOD(172 * IY, 30307)
IZ = MOD(170 * IZ, 30323)
RANDOM = AMOD(FLOAT(IX) / 30269.0 + FLOAT(IY) / 30307.0 + FLOAT(IZ) / 30323.0, 1.0)
```

Therefore IX, IY, IZ generate integers between 0 and 30268, 0 and 30306, and 0 and 30322 respectively. These are combined in the last statement to implement the simple principle that was expressed earlier: if you take three random numbers on [0,1] and sum them, the fractional part of the sum is itself a random number on [0,1].

Because RAND produces pseudo-random numbers, if a long sequence of them is produced, eventually the sequence will repeat itself. Combining random numbers as in the Wichman-Hill procedure guarantees that more than 10^13 numbers will be generated before the repetition begins.

Minitab

Minitab also uses a pseudorandom number generator. Although there should be detailed information about the built-in generator, which should state explicitly which generator is used, there appears not to be in the Minitab manual, the Minitab help files or the Minitab website.

For this project Minitab (Release 14) was used to generate random numbers as follows:

- Calc menu – Random Data – Integer
- Minimum value 0
- Maximum value 1

Hotbits

Hotbits is similar to Random.org in that it is also a physical random number generator. The Hotbits website [11] gives a detailed and lively description of how the numbers are generated. What follows is a summary; Hotbits uses radioactive decay of Kryton-85 as a source of entropy. The numbers are generated by timing successive pairs of radioactive decays detected by a Geiger-Müller tube interfaced to a computer. There is no way to predict when a given atom of Krypton-85 will decay into Rubidium and so the interval between two consecutive decays is also random. A pair of these intervals is measured, and a zero or one emitted based on the relative length of the two intervals. If the same interval is measured for the two decays, then the measurement is discarded. In practice, to avoid any residual bias resulting from non-random systematic errors in the apparatus or measuring process consistently favouring one state, the sense of the comparison between T_1 and T_2 is reversed for consecutive bits.

HotBits output can be requested by filling out and transmitting a request form, which is sent by the users WWW browser in HTTP to Hotbits' Web server, **www.fourmilab.ch**. The request form is processed by a CGI program written in Perl which, after validating the request, forwards it in HTTP format to a dedicated HotBits server machine which is connected to the HotBits generation hardware via the **COM1** port.

To provide better response, the dedicated HotBits server machine maintains an inventory of two million (256 kilobytes) random bits, and services requests from this inventory whenever possible. The server rebuilds inventory in the background, between user requests for HotBits. Random.org has a similar inventory procedure.

There were restrictions on the amount of numbers that could be downloaded from Hotbits. The maximum that can be downloaded is 2048 bytes and the number of downloads per day per computer is limited to five. The suite needs 5485924 bits. To download enough numbers to run the tests 100 times would take over 18 years using one computer! For this reason, the tests were ran only once on HotBits output. Several computers were used to download the numbers over the course of a few days. Note that the "binary download to a file" option was used to download the numbers.

Randomnumbers.info

Like Random.org and Hotbits, Randomnumbers.info [27] is also a physical random number generator. Randomnumbers.info gets its entropy from a quantum source. Exactly how the numbers are generated is not clear from the information posted on the website. It does not use radioactive decay. There is some suggestion perhaps that it uses a photon source.

The numbers can be downloaded easily from the Randomnumbers,info website. Notice that there are again restrictions on the amount of numbers that can be downloaded.

N. RESULTS

This appendix records the results of testing Random.org and the chosen comparatives.

Table N.1 below shows the pass rates for Random.org and the two PRNG comparatives. All three RNGs have acceptable pass rates of the individual tests, where acceptable is deemed to be a pass rate of 88.95% (see Section 4.2.4.1).

While the pass rates of the individual tests are all quite high this does not imply a high pass rate of the suite. Remember that to pass the suite the generator must pass all of the tests in one run – that is the resulting output must be 41 p-values greater than 0.01. Based on 100 run-throughs of the entire suite Random.org passed 68% of the time, Excel passed 76% of the time while Minitab passed 71% of the time. These percentages are somewhat related to the idea of the expected 86% that was discussed in Section 4.2.4 but as noted there less than the 86% would be expected because of the dependence between some of the p-values.

Table N.2 shows the p-value results of the two true random number generators – Randomnumbers.info and Hotbits. Both were subjected to suite of tests once. Hotbits passed all the tests in the suite (all p-values are greater than 0.01). Randomnumbers.info failed both the Non-overlapping and Overlapping Template Matching test. The generator should not be deemed non-random because of these failures. Rather, these failures are evidence of non-randomness. Further testing should be done before any definitive conclusion is made.

Pass Rates				
		Random.org	Excel	Minitab
1	Frequency (monobit)	98	99	96
2	Frequency (block)	97	98	100
3	Runs test	99	97	100
4	Longest run of ones in a block	100	100	100
5	Binary matrix rank	100	99	99
6	Discrete fourier transform (spectral)	100	100	96
7	Non-overlapping template matching	100	100	98
8	Overlapping template matching	98	100	99
9	Maurer's universal statistical	99	100	100
10	Linear complexity	99	100	100
11	Serial (1)	96	99	98
	Serial (2)	99	98	99
12	Approximate entropy	100	99	99
13	Cumulative sums (mode=0)	99	100	99
	Cumulative sums (mode=1)	99	99	99
14	Random excursions (1)	97	100	99
	Random excursions (2)	98	97	100
	Random excursions (3)	99	99	99
	Random excursions (4)	98	99	98
	Random excursions (5)	99	98	99
	Random excursions (6)	96	100	99
	Random excursions (7)	98	99	97
	Random excursions (8)	96	99	100
15	Random excursions variants (1)	99	99	99
	Random excursions variants (2)	99	100	99
	Random excursions variants (3)	99	100	99
	Random excursions variants (4)	99	100	99
	Random excursions variants (5)	100	100	99
	Random excursions variants (6)	100	100	99
	Random excursions variants (7)	99	100	100
	Random excursions variants (8)	99	99	100
	Random excursions variants (9)	99	99	99
	Random excursions variants (10)	99	98	99
	Random excursions variants (11)	99	100	99
	Random excursions variants (12)	100	100	99
	Random excursions variants (13)	100	100	100
	Random excursions variants (14)	99	99	99
	Random excursions variants (15)	98	99	99
	Random excursions variants (16)	98	100	99
	Random excursions variants (17)	98	100	97
	Random excursions variants (18)	99	100	98

Table N.1 Pass Rates

Note: These pass rates are based on 100 tests.

P-values			
		RN.info	Hotbits
1	Frequency (monobit)	1.0000	0.8415
2	Frequency (block)	0.3970	0.0914
3	Runs test	0.1096	0.6920
4	Longest run of ones in a block	0.9436	0.1654
5	Binary matrix rank	0.0732	0.2866
6	Discrete fourier transform (spectral)	0.9750	0.3178
7	Non-overlapping template matching	0.0000	0.2860
8	Overlapping template matching	0.0000	0.0868
9	Maurer's universal statistical	0.6498	0.8207
10	Linear complexity	0.8347	0.1624
11	Serial (1)	0.7678	0.8010
	Serial (2)	0.7854	0.6526
12	Approximate entropy	0.3818	0.9310
13	Cumulative sums (mode=0)	0.2192	0.3230
	Cumulative sums (mode=1)	0.1783	0.3230
14	Random excursions (1)	0.9920	0.3264
	Random excursions (2)	0.7558	0.6843
	Random excursions (3)	0.8741	0.3741
	Random excursions (4)	0.9499	0.8120
	Random excursions (5)	0.6241	0.9421
	Random excursions (6)	0.8031	0.3060
	Random excursions (7)	0.6597	0.3289
	Random excursions (8)	0.8423	0.2430
15	Random excursions variants (1)	0.2716	0.6480
	Random excursions variants (2)	0.3838	0.4274
	Random excursions variants (3)	0.8378	0.3334
	Random excursions variants (4)	0.8988	0.5219
	Random excursions variants (5)	0.5044	0.6452
	Random excursions variants (6)	0.4491	0.6810
	Random excursions variants (7)	0.3458	0.5216
	Random excursions variants (8)	0.2476	0.2019
	Random excursions variants (9)	0.2918	0.2138
	Random excursions variants (10)	0.8330	0.9587
	Random excursions variants (11)	0.1616	0.7197
	Random excursions variants (12)	0.1090	0.9262
	Random excursions variants (13)	0.2168	0.8550
	Random excursions variants (14)	0.3797	0.6247
	Random excursions variants (15)	0.2799	0.3932
	Random excursions variants (16)	0.1360	0.2877
	Random excursions variants (17)	0.2763	0.2806
	Random excursions variants (18)	0.5227	0.4611

Table N.2 P-values for TRNG comparative

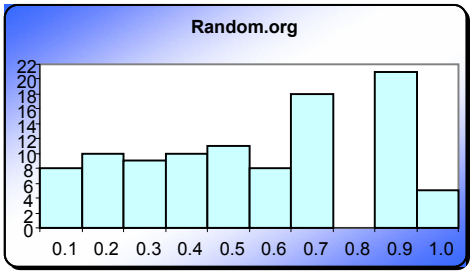
Having looked at the pass rates NIST then recommends examining the uniformity of the p-values. Table N.3 shows the results of a chi-square test on the p-values as described in Section 4.2.4.2. Highlighted in red are the chi-square values that exceed the critical value of 33.725. It can be seen that Excel results lack uniformity for both the overlapping and non-overlapping template matching tests. Random.org perhaps lacks uniformity in the cumulative sums test (forward mode). Its chi-square value marginally exceeds the critical value. Because it is such a small departure from the critical value and all other uniformity checks are well below the critical value, the overall uniformity of p-values is concluded to be satisfactory.

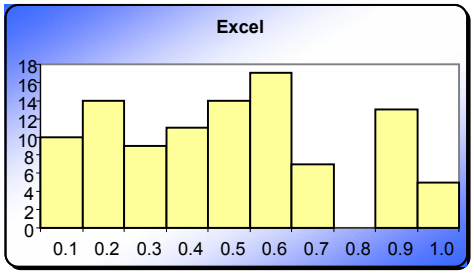
Uniformity Check				
		Random.org	Excel	Minitab
1	Frequency (monobit)	32	22.6	17
2	Frequency (block)	9.8	16.4	5.8
3	Runs test	24.2	4.6	24
4	Longest run of ones in a block	7.6	11.6	6.2
5	Binary matrix rank	10.4	9.8	14.8
6	Discrete fourier transform (spectral)	6.4	1.8	3.6
7	Non-overlapping template matching	14.6	94.4	4
8	Overlapping template matching	12.6	176.8	3.6
9	Maurer's universal statistical	12.4	7.6	10
10	Linear complexity	3.2	22.8	
11	Serial (1)	5.4	9	4
	Serial (2)	3.4	2.6	13.2
12	Approximate entropy	7.8	6.4	10.4
13	Cumulative sums (mode=0)	34.2	9.4	19.2
	Cumulative sums (mode=1)	22	12.8	23.7
14	Random excursions (1)	12.2	14.8	7
	Random excursions (2)	12	10.2	22.4
	Random excursions (3)	10.8	2.4	12.6
	Random excursions (4)	3.2	10.8	16
	Random excursions (5)	14.6	6	12
	Random excursions (6)	8.2	8.6	12.2
	Random excursions (7)	7.4	7.6	8.6
	Random excursions (8)	11.4	10.6	19.2
15	Random excursions variants (1)	1	19.4	11.4
	Random excursions variants (2)	2.6	27.2	13.8
	Random excursions variants (3)	5.2	12.2	8.2
	Random excursions variants (4)	11.2	21	17
	Random excursions variants (5)	14	10.6	14.8
	Random excursions variants (6)	10	15.2	3
	Random excursions variants (7)	2.4	16.2	14.4
	Random excursions variants (8)	5.4	6.2	4.2
	Random excursions variants (9)	11.4	4.6	6.4
	Random excursions variants (10)	8.4	13.2	3.6
	Random excursions variants (11)	10.2	5.4	8
	Random excursions variants (12)	8.2	7.8	8
	Random excursions variants (13)	17.8	8.6	3.2
	Random excursions variants (14)	10.4	12.8	9.4
	Random excursions variants (15)	13.8	2.2	17
	Random excursions variants (16)	11.4	9.8	12.6
	Random excursions variants (17)	11.6	5	8.2
	Random excursions variants (18)	9.4	2.8	14.2

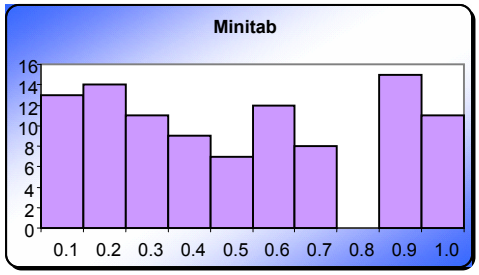
Table N.3 Uniformity Check on p-values

What follows are the histograms of each of the generators for each test. Essentially Table N.3 is a summary of these.

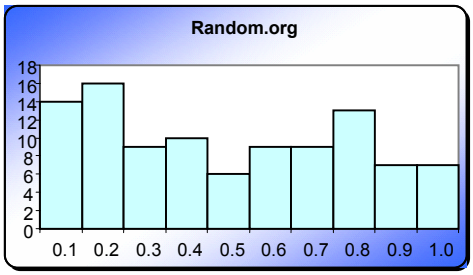
Frequency Test

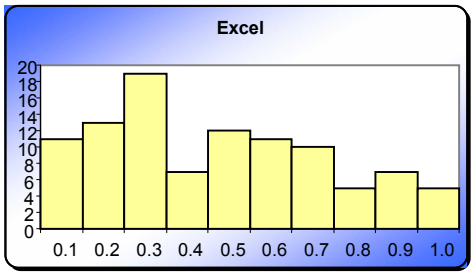


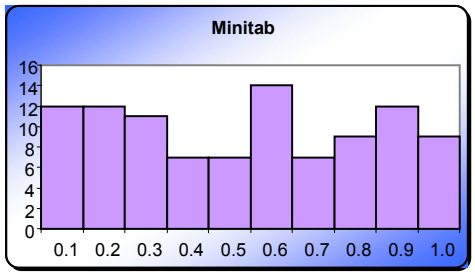




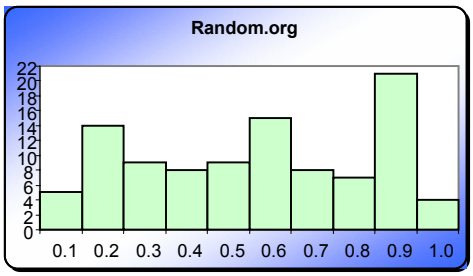
Frequency Block Test

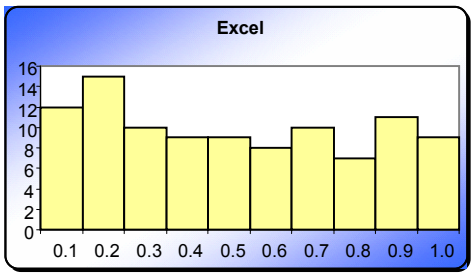


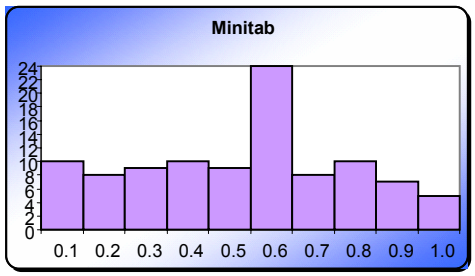




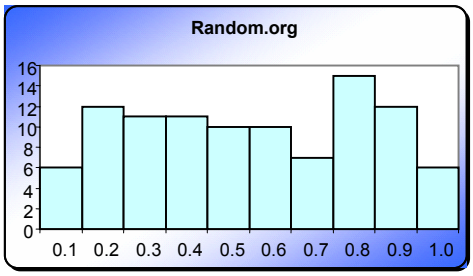
Runs Test

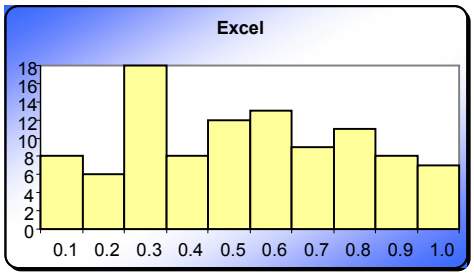


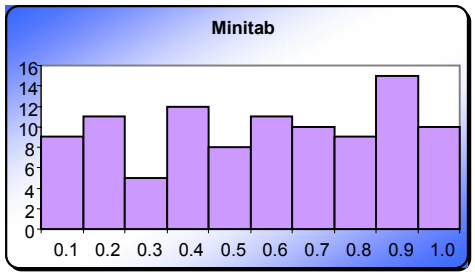




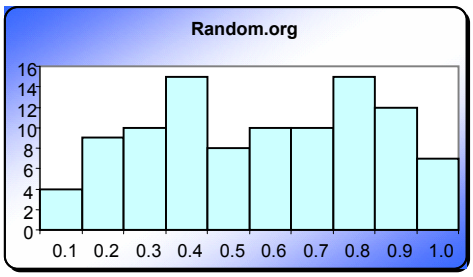
Longest Run in Block Test

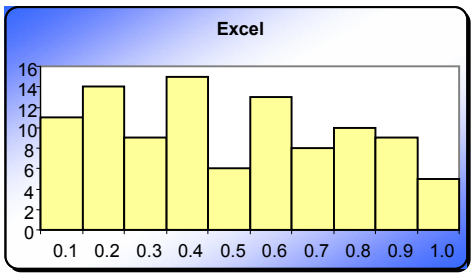


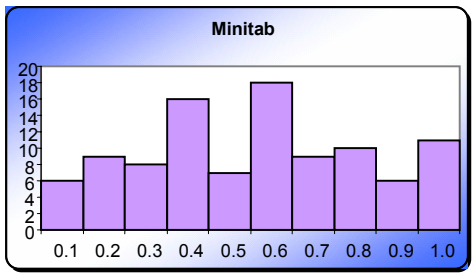




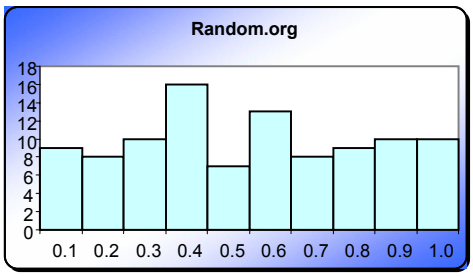
Binary Matrix Test

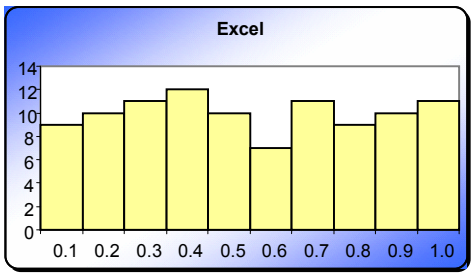


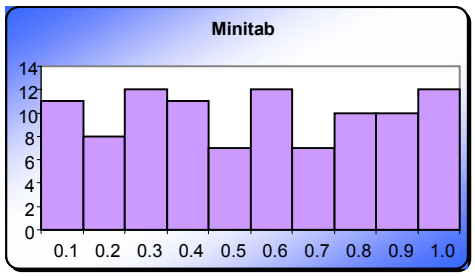




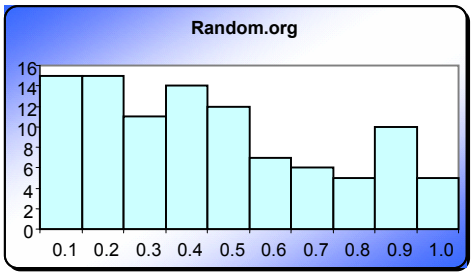
Discrete Fourier (Spectral) Test

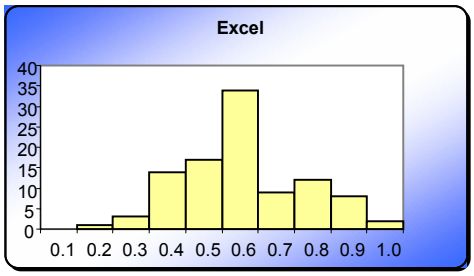


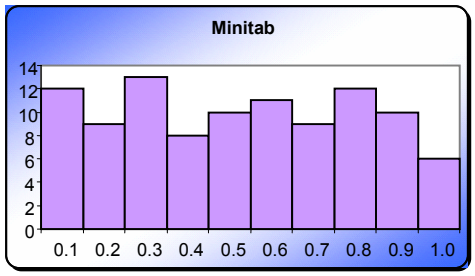




Non-Overlapping Template Matching Test

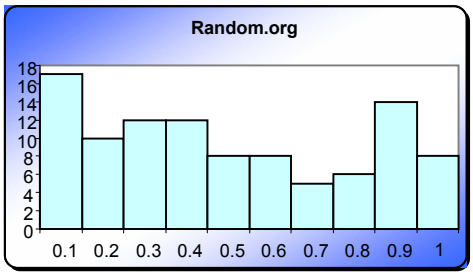


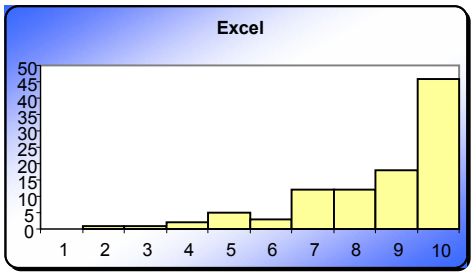


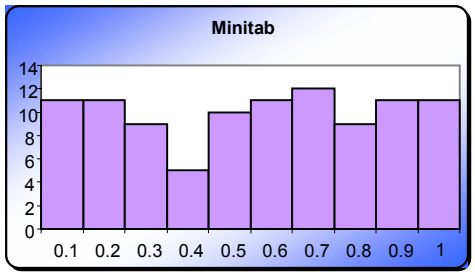


It is for this test that Excel lacks uniformity. Visually, without any formal test, this is evident.

Overlapping Template Matching Test

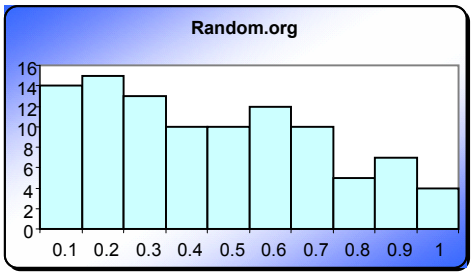


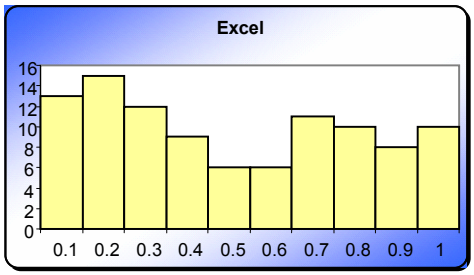


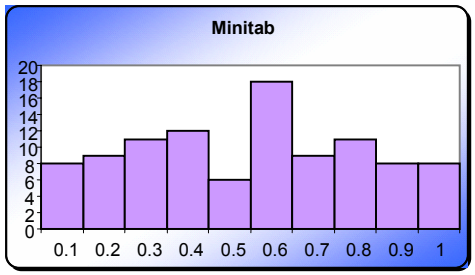


It is for this test also that Excel lack uniformity. Again the bias towards certain bins of p-values is again seen.

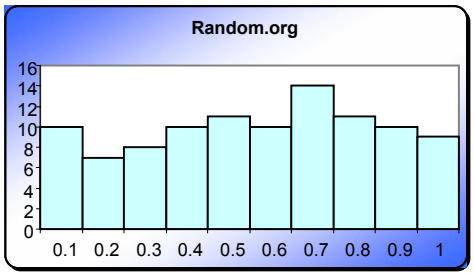
Maurer's "Universal Statistical" Test

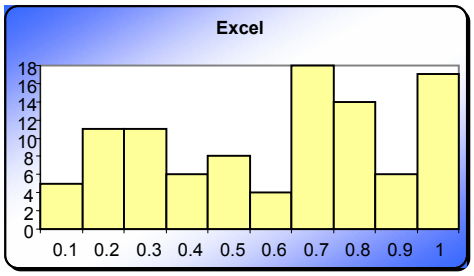






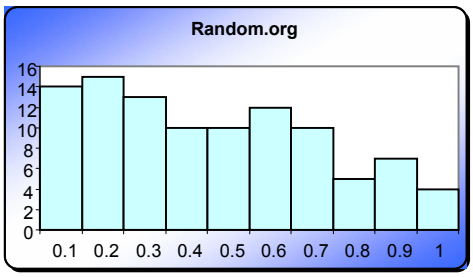
Linear Complexity Test

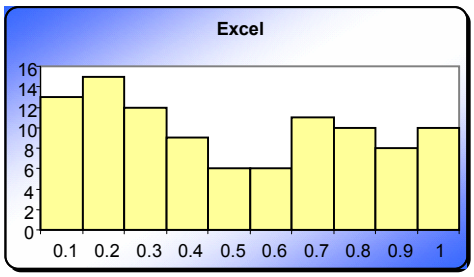


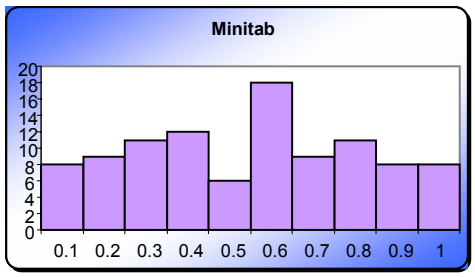


A sufficient amount of p-values for Minitab were not collected for this test to construct a reasonable graph.

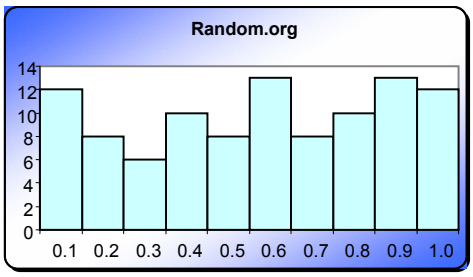
Maurer's "Universal Statistical" Test

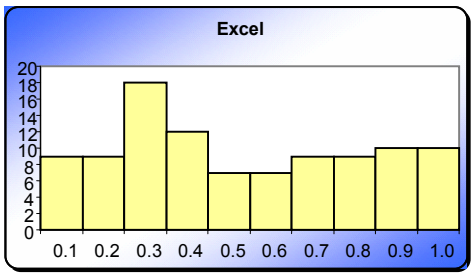


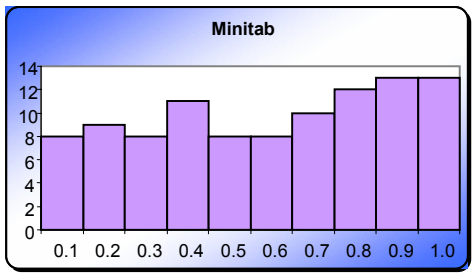




Serial Test

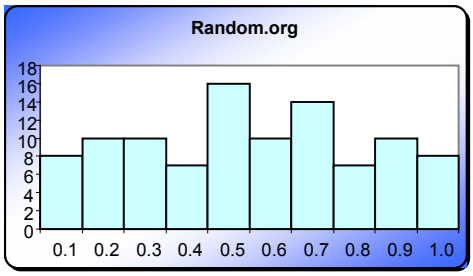


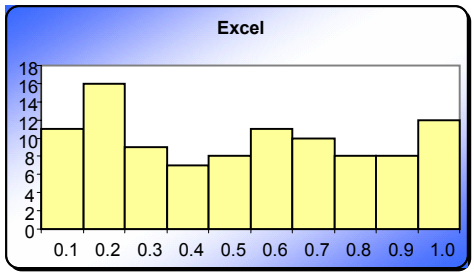


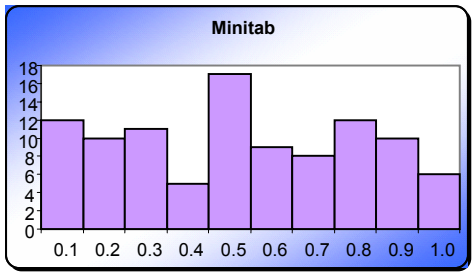


Note: The Serial Test returns two p-values. These histograms are of what NIST call P-value 1.

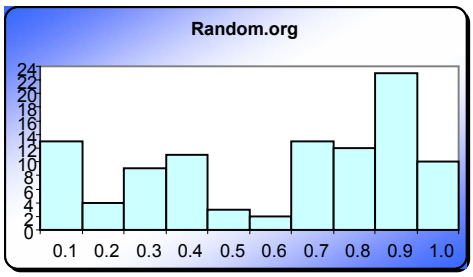
Approximate Entropy Test

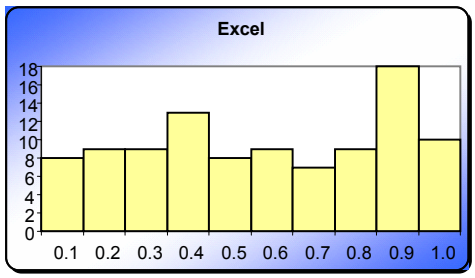


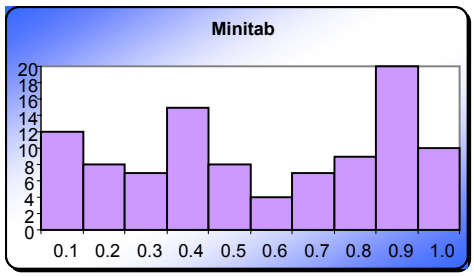




Cumulative Sums Test (mode=0)

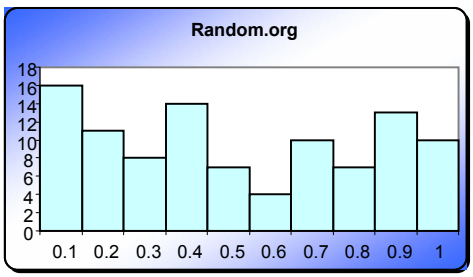


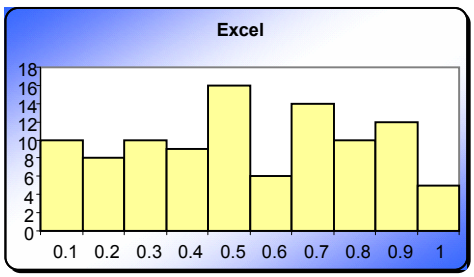


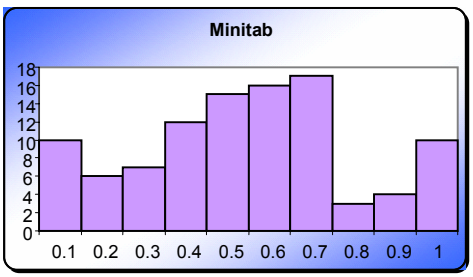


Note: The Cumulative Sums Test results in 2 p-values – one for the forward cumulative sum (mode=0) and one for the backward cumulative sum (mode=1). The former is shown here. It is for this test that Random.org fails the test for uniformity.

Excursions Test

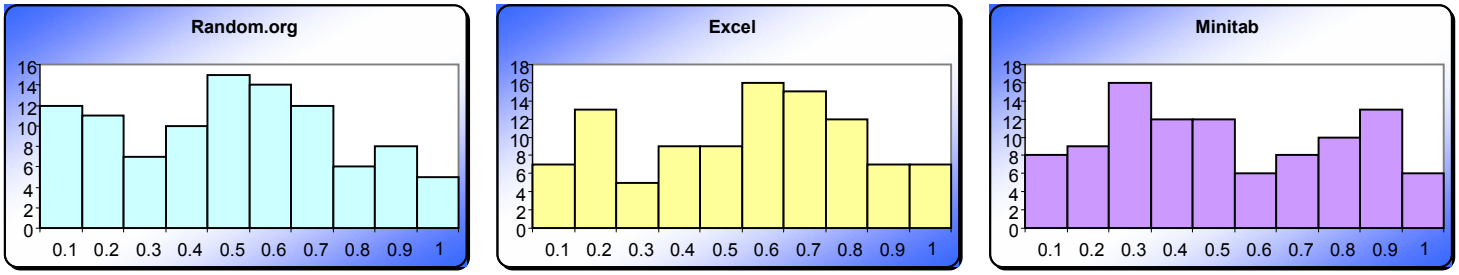






Note: The Excursions Test results in 8 p-values, one for each state. The histograms here show state x=-3.

Excursions Variant Test



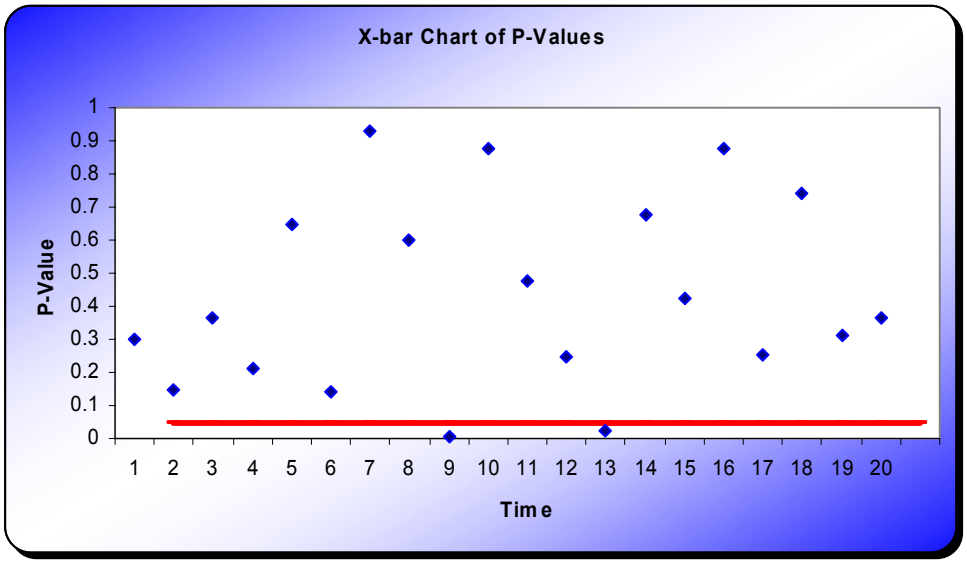
Note: The Excursions Variant Test results in 18 p-values, one for each of the states defined in the test (see Appendix K). The histograms here show state x=5.

O. GRAPHICS

As is mentioned in Section 3.4 graphics are a good way to explore the data. They are also particularly useful for those that do not have a background in statistics. The client also expressed an interest in graphics recommendations as not only would they aid interpretation but would look attractive on the website. This appendix gives an example of what kinds of charts could be constructed.

X-bar Chart of P-values

It is recommended that an X-bar chart of p-values for each test be constructed similar to Figure O.1 below. This will allow identification of possible trends in the p-values of a particular test over time and also identifies the p-values below a certain threshold (red line).



O.1 X-bar chart of p-values

NIST describe three "visualization approaches", or graphics, in the manual. This graphics relate to three of the tests in the test suite – the Discrete Fourier Transform (Spectral) Test, the Approximate Entropy Test and the Linear Complexity Test.

Discrete Fourier Transform (Spectral) Plot

Figure O.2 depicts the spectral components (i.e. the modulus of the DFT) obtained via the application of the Fast Fourier Transform on a sample random binary sequence (consisting of 5000 bits). To demonstrate how the spectral test can detect periodic features in the binary sequence, every 10th bit was changed to a one. To pass this test, no more than 5 % of the peaks should surpass the 95 % cutoff, (determined to be $\sqrt{3*5000} \approx 122.47$. Clearly, greater than 5 % of the peaks exceed the cutoff point in the figure. Thus, the binary sequence fails this test.

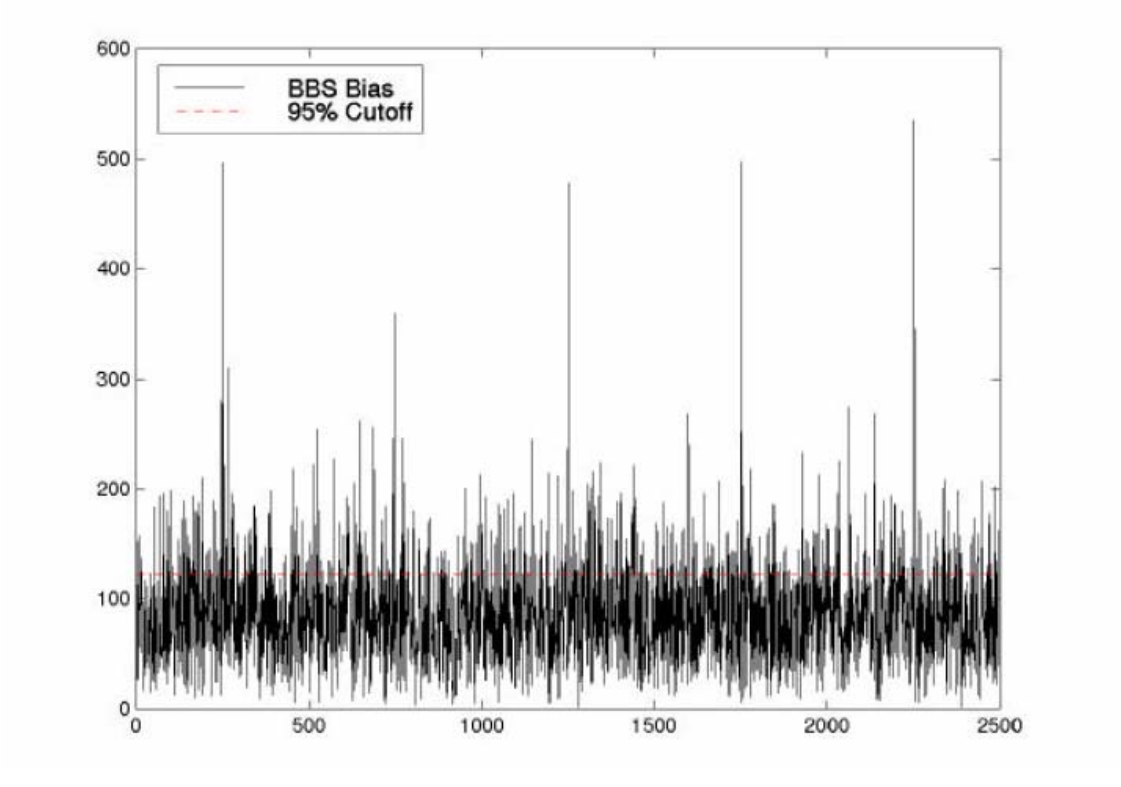


Figure O.2: Discrete Fourier Transform Plot

Source: NIST Manual

Approximate Entropy (ApEn) Graph

Figure O.3 depicts the approximate entropy values (for block length = 2) for three binary sequences, the binary expansion of e and p, and a binary sequence taken from a pseudo-random number generator called SHA-1. In theory, for an n-bit sequence, the maximum entropy value that can be attained is $\ln(2) \approx 0.693147$. The x-axis reflects the number of bits considered in the sequence. The y-axis reflects the deficit from maximal irregularity, that is, the difference between the ln (2) and the observed approximate entropy value. Thus, for a fixed sequence length, one can determine which sequence appears to be more random. For a sequence of 1,000,000 bits, e appears more random than both p and the SHA-119 sequence. However, for larger block sizes, this is not the case.

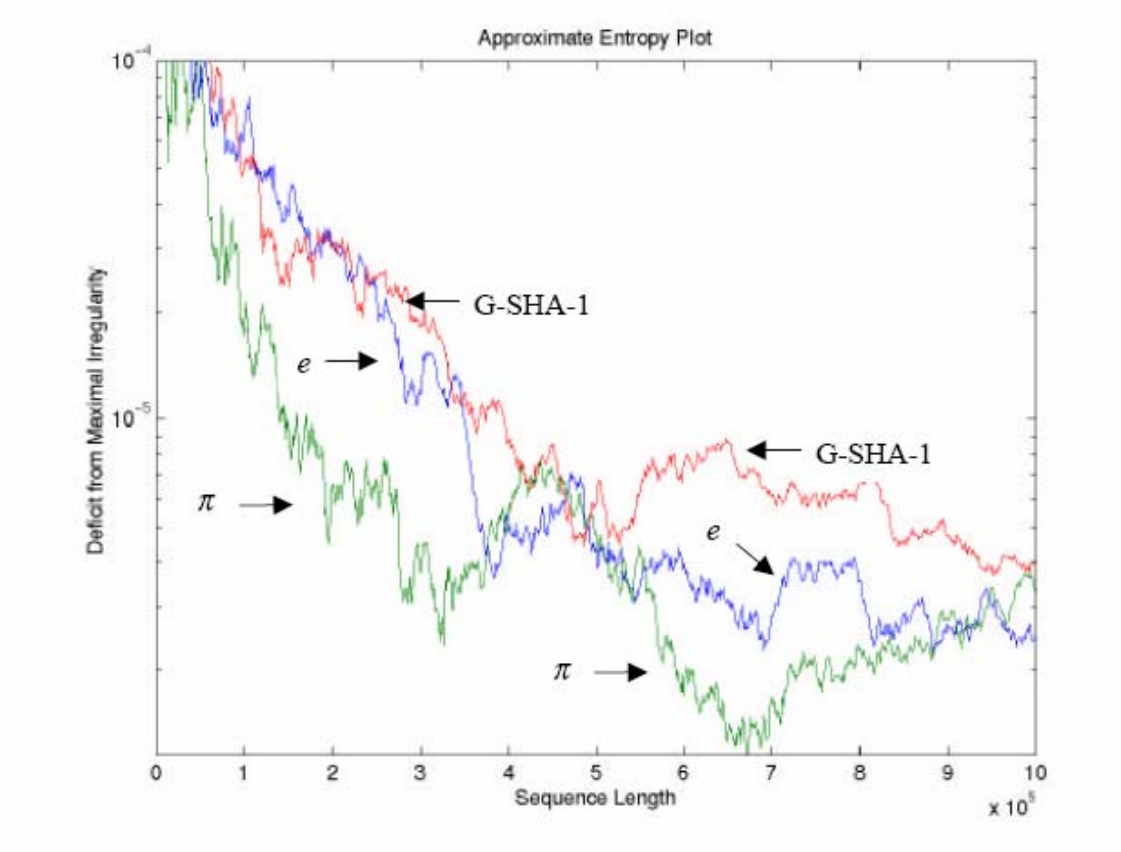


Figure O.3: Approximate Entropy Graph

Source: NIST Manual

Linear Complexity Profile

Figure O.4 depicts the linear complexity profile for a pseudo-random number generator that is strictly based on the XOR (exclusive-or) operator. The generator is defined as follows: given a random binary seed, $x_1, x_2, x_3, ..., x_{127}$, subsequent bits in the sequence are generated according to the following rule: $x_i = x_{i-1} \oplus x_{i-127}$ for $i \geq 128$.

The Berlekamp-Massey algorithm computes the connection polynomial that, for some seed value, reconstructs the finite sequence. The degree of this polynomial corresponds to the length of the shortest Linear Feedback Shift Register (LFSR) that represents the polynomial. The linear complexity profile depicts the degree, which for a random finite length (n-bit) sequence is about n/2. Thus, the x-axis reflects the number of bits observed in the sequence thus far. The y-axis depicts the degree of the connection polynomial. At n = 254, observe that the degree of the polynomial ceases to increase and remains constant at 127. This value precisely corresponds to the number of bits in the seed used to construct the sequence.

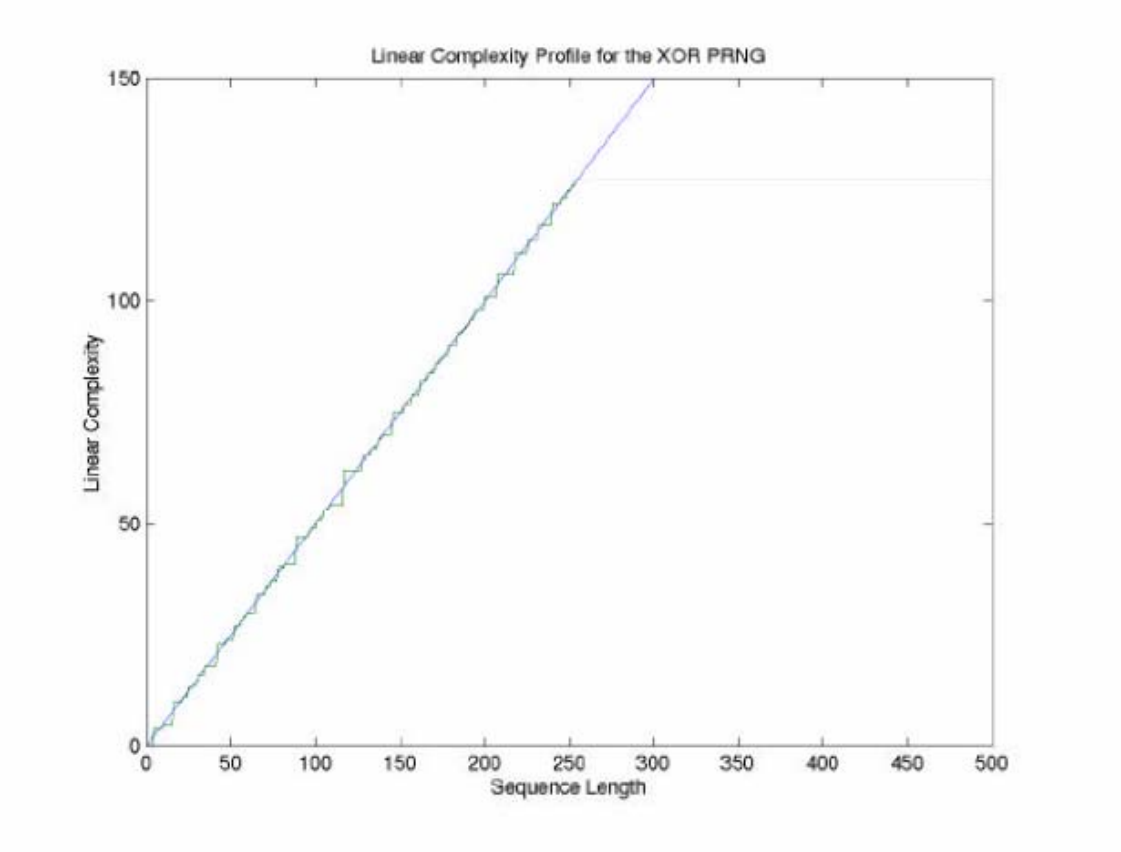


Figure O.4: Linear Complexity Profile

Source: NIST Manual

P. FURTHER READING

This appendix lists some suggested reading for those who are interested to learn more about generating and testing random number generators and for the person who may develop upon this project. These are documents which are not referred to directly in the text but were background reading. The list here is in addition to the references in Appendix X.

1. Pincus and Kalman. *Not all (possibly) "random" sequences are created equal*. Proc. Natl. Acad. USA. Vol 94, pp 3513-1528, April 1997.
2. Pincus and Singer. *Randomness and degrees of irregularity*. Proc. Natl. Acad. USA. Vol 93, pp 2038-2088, March 1996.
3. Szczepanski et al. *Biometric Random Number Generators*, Computers and Security (2004) 23, 77-84. Elsevier.
4. *Holiday Photos Test Galaxy Theory*, News in Science – 15/09/2004
5. Deng and Lin, *Random Number Generation for the New Century*. The American Statistician, May 2000; 54, 2.
6. L'Ecuyer, Pierre. *Uniform Random Number Generators: A Review*. Proceedings of the 1997 Winter Simulation Conference.
7. Lagarias, Jeffrey C., *Pseudorandom Numbers*. Statistical Science 1993, Vol 8, No. 1, 31-39.
8. Kahn, David. *The Code Breakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet.* 1967.
9. Ritter, Terry, Randomness Tests: A Literature Survey. http://www.ciphersbyritter.com/RES/RANDTEST/HTM
10. Bennett, D. J. Randomness. Cambridge, MA: Harvard University Press, 1998.
11. Marsaglia & Zaman, *Monkey Tests for Random Number Generators*, Computer Mathematics Applications, Vol 26, No. 9, pp1-10, 1993.
12. Schindler & Killman, Evaluation Criteria for True (Physical) Random Number Generators Used in Crytographic Applications, Springer-Verlag Berlin Heidelberg 2003.
13. Hayes, Brian, Randomness as a Resource, American Scientist Volume 89, Number 4, July-August 2001 (pages 300-304)
14. L'Ecuyer, Pierre. Random Numbers for Simulation. Communications of the ACM Volume 33, Issue 10 (October 1990), pages 85-87.
15. Park & Miller, Random Number Generators: Good Ones are Hard To Find, Communications of the ACM, Computing Practices, October 1988, Volume 31, Number 10.
16. Modianos et al., Testing Intrinsic Random-Number Generators, Byte Programming Insight, January 1987
17. Kronmal, Richard, Evaluation of a Pseudorandom Normal Number Generator, Journal of the Association for Computing Machinery, Vol 11, No.3 (July 1964) pp.357-363.
18. Marsaglia, George, A Current View of Random Number Generators, Keynote Address, Computer Science and Statistics: 16th Symposium on the Interface, 1984.
19. How We Learned to Cheat at Online Poker: A Study in Software Security By Brad Arkin Frank Hill Scott Marks Matt Schmid and Thomas John Walls http://www.developer.com/tech/article.php/616221

20. True random number generators http://www.robertnz.net/true_rng.html
21. Maclaren, Nick. Cryptographic Pseudo-random Numbers in Simulation. Fast software encryption : Cambridge Security Workshop, Cambridge, U.K., December 9-11, 1993 : proceedings
22. Tsang et al., Tuning the Collision Test for Stringency, HKU CSIS Tech Report, May 2000.
23. L'Ecuyer, Pierre. Software for Uniform Random Number Generation: Distinguishing the Good and the Bad.
24. The Evaluation of RPG100 by Using NIST and DIEHARD tests, FDK Corporation, Dec 2003. http://www.fdk.co.jp/cyber-e/pdf/HM-RAE104.pdf
25. Murphy, Sean. The Power of NIST's Statistical Testing of AES Candidates, Information Security Group, University of London, March 2000. (but NIST 2001)
26. Schindler and Killmann, *Evaluation Criteria for True (Physical) Random Number Generators Used in Cryptographic Applications,* Revised Papers from the 4th International Workshop on Cryptographic Hardware and Embedded Systems, 2002

Q. GLOSSARY

This glossary is provided for the convenience of the reader.

Term	Definition
Entropy	A measure of the disorder or randomness in a closed system.
Alternative hypothesis	The alternative to the null hypothesis. In this case it is any non-random characteristic.
Binary	0 or 1
Binary Sequence	Sequence of zeroes and ones
Bit string	A sequence of bits
Block	A subset of a bit string. A block has a predetermined length
Compressibility	Refers to the existence of a sub-sequence that represents the entire sequence.
Confidence interval	An interval which is believed, with a pre-assigned degree of confidence, to include the particular value of some parameter being estimated
Critical Value	The value that is exceeded by the test statistic with a small probability (significance level). A “look up” or calculated value of a test statistic that, by construction, has a small probability of occurring when the null hypothesis is true.
Cryptography	The art or science of turning meaningful sequences into apparently random noise in such a way that a key-holder can recover the original data
Deterministic	Given the same initial seed, the generator will always produce the same output sequence.
Entropy source	A physical source of information whose output either appears to be random in itself or by applying some filtering/distillation process.
Hypothesis Test	is a procedure for determining if an assertion about a characteristic of a population is reasonable.
Linear congruential method	A popular algorithm for generating random numbers X(n+1)=(aX(n)+c)mod m, n>=0 (note that it always gets into a loop)
Loop	A cycle of numbers that is repeated endlessly.
Matlab	An integrated, technical computer environment that combines numeric computation, advanced graphics and visualization, and a high level programming language. http://www.mathworks.com/
Mixed congruential method	Linear Congruential Method when c=!0
Monte Carlo methods	A general term used for any algorithm that employs random

	numbers.
Multiplicative congruential method	Linear Congruential Method when c=0
NIST	National Institute of Standards and Technology
Null hypothesis	The stated hypothesis. In this case, the null hypothesis is that a binary sequence is random from a statistical viewpoint.
Oscillation	Refers to abrupt changes between runs of zeroes or runs of ones
Period	The repeating cycle. A useful sequence will have a relatively long period.
Periodicity	Refers to sub-sequences that repeat.
PRNG	Pseudorandom Number Generator
Pseudo-random numbers	A sequence of pseudo-random numbers is a deterministic sequence of numbers having the same statistical properties as a sequence of random numbers.
p-value	A measure of the strength of the evidence provided by the data against the hypothesis
Random bit generator	Is a device or algorithm which output a sequence of statistically independent and unbiased binary digits.
Random walk	A sequence of steps, each of whose characteristics is determined by chance
Rank (of a matrix)	Refers to the rank of a matrix in linear algebra. Having reduced a matrix to row-echelon form via elementary row operations, the number of nonzero rows, if any, are counted in order to determine the number of linearly independent rows or columns in the matrix
RNG	Random Number Generator
Run	An uninterrupted sequence of like bits (i.e., either all zeroes or all ones) A run of 0's is called a gap, while a run of 1's is called a block.
Seed	The input to a pseudorandom number generator. Different seeds generate different pseudorandom sequences.
Significance level	Usually denoted as, alpha (α), it is the least upper bound of the probability of an error of type I for all distributions consistent with the null hypothesis.
Simulation	This is the re-creation, albeit in a simplified manner, of a complex phenomena, environment, or experience, providing the user with the opportunity for some new level of understanding.
Test statistic	A statistic upon which a test of a hypothesis is based.
TRNG	True Random Number Generator
Type I error	The likelihood that a test rejects a binary sequence that

	was, in fact, produced by an acceptable random number generator.
Type II error	The likelihood that a test accepts a binary sequence that was, in fact, produced by an unacceptable random number generator.

R. REFERENCES

1. The Distributed Computing Group, Trinity College Dublin. http://www.dsg.cs.tcd.ie/
2. Foley, Louise. Analysis of an On-Line Random Number Generator, MSISS Project Report, April 2001.
3. Walker, John, ENT Program http://www.fourmilab.ch/random/
4. Brief Investigation into Random Number Generation http://people.bath.ac.uk/tdj20/gee.html#about
5. Knuth, Donald E., *The Art of Computer Programming - Seminumerical Algorithm.* Vol 2 Chapter 3 *Random Numbers* pg1-184. 1997.
6. NIST (National Institute of Standards and Technology), *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*. 2001
7. Randomness Recommendations for Security http://www.cse.ohio-state.edu/cgi-bin/rfc/rfc1750.html
8. Menezes, van Oorschot and Vanstone, *Handbook of Applied Cryptography,* CRC Press 1997.
9. Prichett, James. Introduction to the Music of John Cage http://www.music.princeton.edu/~jwp/texts/bookintro.html
10. Technician http://technician.org
11. Hotbits, http://www.fourmilab.ch/hotbits/
12. Lavarand http://lavarand.sgi.com
13. Vattulainen et al., *A Comparative Study of Some Pseudorandom Number Generators*, Department of Electrical Engineering, August 1993.
14. Klimasauskas, C. *Not Knowing Your Random Number Generator Could be Costly: Random Generators – Why Are They Important*. PCAI (PC Artificial Intelligence) Issue 16.3 pg 50-56
15. L'Ecuyer, Pierre. *Uniform Random Number Generators.* Proceedings of the 1998 Winter Simulation Conference.
16. http://sprng.cs.fdu.edu/Version1.0/paper/node16.html
17. L'Ecuyer and Hellekalek, *Random Number Generators: Selection Criteria and Testing*, Lecture Notes in Statistics, New York Springer.
18. P. L'Ecuyer, *Random Numbers*, in the International Encyclopedia of the Social and Behavioral Sciences, N. J. Smelser and Paul B. Baltes Eds., Pergamon, Oxford, 2002, 12735-12738.
19. Diehard Battery of Statistical Tests http://stat.fsu.edu/~geo/diehard.html
20. Brief Description of the Crypt-X tests http://www.isrc.qut.edu.au/resource/cryptx/tests.php
21. Peterson, Ivars, *The Bias of Random Number Generators* http://www.maa.org/mathland/mathtrek_09_29_03.html
22. Rutti, Mario, *A Random Number Generator Test Suite for the C++ Standard* (Diploma Thesis), Institute for Theoretical Physics, Zurich, 2004. http://www.comp-phys.org:16080/rngts/doc/main.pdf
23. Song-Ju Kim et al. *Corrections of the NIST Statistical Test Suite for Randomness*, http://eprint.iacr.org/2004/018.pdf, 2004
24. Confidence Intervals for the Binomial Distribution www.graphpad.com

25. Agresti & Coull, Approximate is better than “Exact” for interval estimation of binomial proportions, The American Statistician, 52:119-126, 1998.
26. Minitab http://www.minitab.com
27. http://www.randomnumbers.info
28. Soto, Juan. *Statistical Testing of Random Number Generators*. National Institute of Standards and Technology.
29. Kim, Unemno & Hasegawa, *Corrections of the NIST Statistical Test Suite for Randomness*, January 2004. Available at: http://eprint.iacr.org/2004/018.pdf.
30. Description of the Berlekamp Massey Algorithm http://planetmath.org/encyclopedia/BerlekampMasseyAlgorithm.html
31. Lenore Blum, Manuel Blum, and Michael Shub. "A Simple Unpredictable Pseudo-Random Number Generator", *SIAM Journal on Computing*, volume 15, pages 364–383, May 1986.
32. Pascal Junod, "Cryptographic Secure Pseudo-Random Bits Generation: The Blum-Blum-Shub Generator", August 1999. (*http://crypto.junod.info/bbs.pdf*)
33. Mersenne Twister Homepage http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html
34. What are pseudorandom number generators? http://www.answers.com/topic/pseudorandom-number-generator
35. Ripley, Chapter 2 *Pseudo-Random Numbers* pg14-47
36. McCullough & Wilson, *On the accuracy of statistical procedures in Microsoft Excel 97*, Computational Statistics and Data Analysis 31 (1999) 27-37.
37. McCullough & Wilson, On the accuracy of statistical procedures in Microsoft Excel 2000 and Excel XP, Computational Statistics and Data Analysis 40 (2002) 713-721.
38. Bassham, Larry, Validation Testing and NIST Statistical Test Suite (workshop presentation), July 2004. http://csrc.nist.gov/CryptoToolkit/RNG/Workshop/ValTestandSTS.pdf
39. Van Lambalgen, M. Von Mises’ Definition of Random Sequences Reconsidered. The Journal of Symbolic Logic Colume 52 pg 725-, 1987 http://staff.science.uva.nl/~michiell/docs/JSL87.pdf
40. Schindler, Werner, Efficient Online Tests for True Random Number Generators, Springer-Verlag Berlin Heidelberg 2001.

S. INDEX