A STATISTICAL TEST SUITE FOR RANDOM AND PSEUDORANDOM NUMBER GENERATORS FOR CRYPTOGRAPHIC APPLICATIONS

NIST Special Publication 800-22
(with revision dated December 2000)

Andrew Rukhin, Juan Soto, James Nechvatal,
Miles Smid, Elaine Barker, Stefan Leigh,
Mark Levenson, Mark Vangel, David Banks,
Alan Heckert, James Dray, San Vo

ABSTRACT

This paper discusses some aspects of selecting and testing random and pseudorandom number generators. The outputs of such generators may be used in many cryptographic applications, such as the generation of key material. Generators suitable for use in cryptographic applications may need to meet stronger requirements than for other applications. In particular, their outputs must be unpredictable in the absence of knowledge of the inputs. Some criteria for characterizing and selecting appropriate generators are discussed in this document. The subject of statistical testing and its relation to cryptanalysis is also discussed, and some recommended statistical tests are provided. These tests may be useful as a first step in determining whether or not a generator is suitable for a particular cryptographic application. However, no set of statistical tests can absolutely certify a generator as appropriate for usage in a particular application, i.e., statistical testing cannot serve as a substitute for cryptanalysis. The design and cryptanalysis of generators is outside the scope of this paper.

Key words: random number generator, hypothesis test, P-value

Certain commercial equipment and materials were used in the development of this test suite. Such identification does not imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the materials or equipment identified are necessarily the best available for the purpose.

TABLE OF CONTENTS

1 INTRODUCTION TO RANDOM NUMBER TESTING 1

1.1 General Discussion 1
1.1.1 Randomness 1
1.1.2 Unpredictability 2
1.1.3 Random Number Generators (RNGs) 2
1.1.4 Pseudorandom Number Generators (PRNGs) 3
1.1.5 Testing 3
1.1.6 Considerations for Randomness, Unpredictability and Testing 6

1.2 Definitions and Abbreviations 6

1.3 Mathematical Symbols 11

2 RANDOM NUMBER GENERATION TESTS 13

2.1 Frequency (Monobit) Test 14
2.1.1 Test Purpose 14
2.1.2 Function Call 14
2.1.3 Test Statistic and Reference Distribution 15
2.1.4 Test Description 15
2.1.5 Decision Rule (at the 1 % Level) 15
2.1.6 Conclusion and Interpretation of Test Results 15
2.1.7 Input Size Recommendations 16
2.1.8 Example 16

2.2 Frequency Test within a Block 16
2.2.1 Test Purpose 16
2.2.2 Function Call 16
2.2.3 Test Statistic 17
2.2.4 Test Description 17
2.2.5 Decision Rule (at the 1 % Level) 18
2.2.6 Conclusion and Interpretation of Test Results 18
2.2.7 Input Size Recommendations 18
2.2.8 Example 18

2.3 Runs Test 18
2.3.1 Test Purpose 18
2.3.2 Function Call 19
2.3.3 Test Statistic and Reference Distribution 19
2.3.4 Test Description 19
2.3.5 Decision Rule (at the 1 % Level) 20
2.3.6 Conclusion and Interpretation of Test Results 20
2.3.7 Input Size Recommendations 20
2.3.8 Example 21

2.4 Test for the Longest Run of Ones in a Block 21
2.4.1 Test Purpose 21
2.4.2 Function Call 21
2.4.3 Test Statistic and Reference Distribution 22

2.4.4 Test Description ... 22
2.4.5 Decision Rule (at the 1 % Level) ... 23
2.4.6 Conclusion and Interpretation of Test Results ... 23
2.4.7 Input Size Recommendations ... 23
2.4.8 Example ... 23

2.5 Binary Matrix Rank Test ... 24
2.5.1 Test Purpose ... 24
2.5.2 Function Call ... 24
2.5.3 Test Statistic and Reference Distribution ... 25
2.5.4 Test Description ... 25
2.5.5 Decision Rule (at the 1 % Level) ... 26
2.5.6 Conclusion and Interpretation of Test Results ... 26
2.5.7 Input Size Recommendations ... 26
2.5.8 Example ... 26

2.6 Discrete Fourier Transform (Spectral) Test ... 27
2.6.1 Test Purpose ... 27
2.6.2 Function Call ... 27
2.6.3 Test Statistic and Reference Distribution ... 27
2.6.4 Test Description ... 27
2.6.5 Decision Rule (at the 1 % Level) ... 28
2.6.6 Conclusion and Interpretation of Test Results ... 28
2.6.7 Input Size Recommendations ... 29
2.6.8 Example ... 29

2.7 Non-overlapping Template Matching Test ... 29
2.7.1 Test Purpose ... 29
2.7.2 Function Call ... 29
2.7.3 Test Statistic and Reference Distribution ... 30
2.7.4 Test Description ... 30
2.7.5 Decision Rule (at the 1 % Level) ... 31
2.7.6 Conclusion and Interpretation of Test Results ... 31
2.7.7 Input Size Recommendations ... 32
2.7.8 Example ... 32

2.8 Overlapping Template Matching Test ... 32
2.8.1 Test Purpose ... 32
2.8.2 Function Call ... 33
2.8.3 Test Statistic and Reference Distribution ... 33
2.8.4 Test Description ... 33
2.8.5 Decision Rule (at the 1 % Level) ... 35
2.8.6 Conclusion and Interpretation of Test Results ... 35
2.8.7 Input Size Recommendations ... 35
2.8.8 Example ... 36

2.9 Maurer's "Universal Statistical" Test ... 36
2.9.1 Test Purpose ... 36
2.9.2 Function Call ... 36
2.9.3 Test Statistic and Reference Distribution ... 36
2.9.4 Test Description ... 37
2.9.5 Decision Rule (at the 1 % Level) ... 39
2.9.6 Conclusion and Interpretation of Test Results ... 40
2.9.7 Input Size Recommendations ... 40
2.9.8 Example ... 40

2.10 Lempel-Ziv Compression Test 41
2.10.1 Test Purpose 41
2.10.2 Function Call 41
2.10.3 Test Statistic and Reference Distribution 41
2.10.4 Test Description 41
2.10.5 Decision Rule (at the 1 % Level) 42
2.10.6 Conclusion and Interpretation of Test Results 42
2.10.7 Input Size Recommendations 43
2.10.8 Example 43

2.11 Linear Complexity Test 43
2.11.1 Test Purpose 43
2.11.2 Function Call 43
2.11.3 Test Statistic and Reference Distribution 44
2.11.4 Test Description 44
2.11.5 Decision Rule (at the 1 % Level) 45
2.11.6 Conclusion and Interpretation of Test Results 45
2.11.7 Input Size recommendations 46
2.11.8 Example 46

2.12 Serial Test 46
2.12.1 Test Purpose 46
2.12.2 Function Call 46
2.12.3 Test Statistics and Reference Distribution 47
2.12.4 Test Description 47
2.12.5 Decision Rule (at the 1 % Level) 48
2.12.6 Conclusion and Interpretation of Test Results 48
2.12.7 Input Size Recommendations 48
2.12.8 Example 48

2.13 Approximate Entropy Test 49
2.13.1 Test Purpose 49
2.13.2 Function Call 49
2.13.3 Test Statistic and Reference Distribution 49
2.13.4 Test Description 50
2.13.5 Decision Rule (at the 1 % Level) 51
2.13.6 Conclusion and Interpretation of Test Results 51
2.13.7 Input Size Recommendations 51
2.13.8 Example 51

2.14 Cumulative Sums (Cusum) Test 52
2.14.1 Test Purpose 52
2.14.2 Function Call 52
2.14.3 Test Statistic and Reference Distribution 52
2.14.4 Test Description 52
2.14.5 Decision Rule (at the 1 % Level) 54
2.14.6 Conclusion and Interpretation of Test Results 54
2.14.7 Input Size Recommendations 54
2.14.8 Example 54

2.15 Random Excursions Test 54
2.15.1 Test Purpose 54
2.15.2 Function Call 55
2.15.3 Test Statistic and Reference Distribution 55
2.15.4 Test Description 55
2.15.5 Decision Rule (at the 1 % Level) 58

2.15.6 Conclusion and Interpretation of Test Results 59
2.15.7 Input Size Recommendations..... 59
2.15.8 Example 59

2.16 Random Excursions Variant Test..... 59
2.16.1 Test Purpose..... 59
2.16.2 Function Call..... 60
2.16.3 Test Statistic and Reference Distribution..... 60
2.16.4 Test Description..... 60
2.16.5 Decision Rule (at the 1 % Level) 62
2.16.6 Conclusion and Interpretation of Test Results 62
2.16.7 Input Size Recommendations..... 62
2.16.8 Example 62

3 TECHNICAL DESCRIPTION OF TESTS..... 64

3.1 Frequency (Monobit) Test……...64

3.2 Frequency Test within a Block.……65

3.3 Runs Test……66

3.4 Test for the Longest Run of Ones in a Block ...……...……67

3.5 Binary Matrix Rank Test……..69

3.6 Discrete Fourier Transform (Spectral) Test……71

3.7 Non-overlapping Template Matching Test……..74

3.8 Overlapping Template Matching Test……77

3.9 Maurer's "Universal Statistical" Test……..78

3.10 Lempel-Ziv Compression Test……..81

3.11 Linear Complexity Test……84

3.12 Serial Test……..…..87

3.13 Approximate Entropy Test……....89

3.14 Cumulative Sums (Cusum) Test……...91

3.15 Random Excursions Test……...93

3.16 Random Excursions Variant Test…….……96

4. TESTING STRATEGY AND RESULT INTERPRETATION 98

4.1 Strategies for the Statistical Analysis of an RNG 98

4.2 The Interpretation of Empirical Results 100

4.2.1 Proportion of Sequences Passing a Test 100
4.2.2 Uniform Distribution of *P-values* 101

4.3 General Recommendations and Guidelines 101

4.4 Application of Multiple Tests 104

5. USER'S GUIDE 106

5.1 About the Package 106

5.2 System Requirements 107

5.3 How to Get Started 107

5.4 Data Input and Output of Empirical Results 109
5.4.1 Data Input 109
5.4.2 Output of Empirical Results 109
5.4.3 Test Data Files 109

5.5 Program Layout 109
5.5.1 General Program 110
5.5.2 Implementation Details 110
5.5.3 Description of the Test Code 111

5.6 Running the Test Code 113

5.7 Interpretation of Results 115

APPENDIX A: RANK COMPUTATION FOR BINARY MATRICES 117

APPENDIX B: SOURCE CODE 121

APPENDIX C: EMPIRICAL RESULTS FOR SAMPLE DATA 124

APPENDIX D: CONSTRUCTION OF APERIODIC TEMPLATES 127

APPENDIX E: GENERATION OF THE BINARY EXPANSION OF IRRATIONAL NUMBERS 129

APPENDIX F: NUMERIC ALGORITHM ISSUES 130

APPENDIX G: HIERARCHICAL DIRECTORY STRUCTURE 132

APPENDIX H: VISUALIZATION APPROACHES 136

APPENDIX I: INSTRUCTIONS FOR INCORPORATING ADDITIONAL STATISTICAL TESTS 139

APPENDIX J: INSTRUCTIONS FOR INCORPORATING ADDITIONAL PRNGS 141

APPENDIX K: GRAPHICAL USER INTERFACE (GUI) 143

APPENDIX L: DESCRIPTION OF THE REFERENCE PSEUDO RANDOM NUMBER GENERATORS 146

APPENDIX M: REFERENCES 151

1 INTRODUCTION TO RANDOM NUMBER TESTING

The need for random and pseudorandom numbers arises in many cryptographic applications. For example, common cryptosystems employ keys that must be generated in a random fashion. Many cryptographic protocols also require random or pseudorandom inputs at various points, e.g., for auxiliary quantities used in generating digital signatures, or for generating challenges in authentication protocols.

This document discusses the randomness testing of random number and pseudorandom number generators that may be used for many purposes including cryptographic, modeling and simulation applications. The focus of this document is on those applications where randomness is required for cryptographic purposes. A set of statistical tests for randomness is described in this document. The National Institute of Standards and Technology (NIST) believes that these procedures are useful in detecting deviations of a *binary sequence* from randomness. However, a tester should note that apparent deviations from randomness may be due to either a poorly designed generator or to anomalies that appear in the binary sequence that is tested (i.e., a certain number of failures is expected in random sequences produced by a particular generator). It is up to the tester to determine the correct interpretation of the test results. Refer to Section 4 for a discussion of testing strategy and the interpretation of test results.

1.1 General Discussion

There are two basic types of generators used to produce random sequences: ***random number generators*** (RNGs - see Section 1.1.3) and ***pseudorandom number generators*** (PRNGs - see Section 1.1.4). For cryptographic applications, both of these generator types produce a stream of zeros and ones that may be divided into substreams or blocks of random numbers.

1.1.1 Randomness

A ***random*** bit sequence could be interpreted as the result of the flips of an unbiased "fair" coin with sides that are labeled "0" and "1," with each flip having a probability of exactly ½ of producing a "0" or "1." Furthermore, the flips are ***independent*** of each other: the result of any previous coin flip does *not* affect future coin flips. The unbiased "fair" coin is thus the perfect random bit stream generator, since the "0" and "1" values will be randomly distributed (and [0,1] uniformly 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.

Obviously, the use of unbiased coins for cryptographic purposes is impractical. Nonetheless, the hypothetical output of such an idealized generator of a true random sequence serves as a benchmark for the evaluation of random and pseudorandom number generators.

1.1.2 Unpredictability

Random and pseudorandom numbers generated for cryptographic applications should be unpredictable. In the case of PRNGs, if the seed is unknown, the next output number in the sequence should be unpredictable in spite of any knowledge of previous random numbers in the sequence. This property is known as forward unpredictability. It should also not be feasible to determine the seed from knowledge of any generated values (i.e., backward unpredictability is also required). No correlation between a seed and any value generated from that seed should be evident; each element of the sequence should appear to be the outcome of an independent random event whose probability is 1/2.

To ensure forward unpredictability, care must be exercised in obtaining seeds. The values produced by a PRNG are completely predictable if the seed and generation algorithm are known. Since in many cases the generation algorithm is publicly available, the seed must be kept secret and should not be derivable from the pseudorandom sequence that it produces. In addition, the seed itself must be unpredictable.

1.1.3 Random Number Generators (RNGs)

The first type of sequence generator is a random number generator (RNG). An RNG uses a non-deterministic source (i.e., the entropy source), along with some processing function (i.e., the entropy distillation process) to produce randomness. The use of a distillation process is needed to overcome any weakness in the entropy source that results in the production of non-random numbers (e.g., the occurrence of long strings of zeros or ones). The entropy source typically consists of some physical quantity, such as the noise in an electrical circuit, the timing of user processes (e.g., key strokes or mouse movements), or the quantum effects in a semiconductor. Various combinations of these inputs may be used.

The outputs of an RNG may be used directly as a random number or may be fed into a pseudorandom number generator (PRNG). To be used directly (i.e., without further processing), the output of any RNG needs to satisfy strict randomness criteria as measured by statistical tests in order to determine that the physical sources of the RNG inputs appear random. For example, a physical source such as electronic noise may contain a superposition of regular structures, such as waves or other periodic phenomena, which may appear to be random, yet are determined to be non-random using statistical tests.

For cryptographic purposes, the output of RNGs needs to be unpredictable. However, some physical sources (e.g., date/time vectors) are quite predictable. These problems may be mitigated by combining outputs from different types of sources to use as the inputs for an RNG. However, the resulting outputs from the RNG may still be deficient when evaluated by statistical tests. In addition, the production of high-quality random numbers may be too time consuming, making such production undesirable when a large quantity of random numbers is needed. To produce large quantities of random numbers, pseudorandom number generators may be preferable.

1.1.4 Pseudorandom Number Generators (PRNGs)

The second generator type is a pseudorandom number generator (PRNG). A PRNG uses one or more inputs and generates multiple "pseudorandom" numbers. Inputs to PRNGs are called ***seeds***. In contexts in which unpredictability is needed, the seed itself must be random and unpredictable. Hence, by default, a PRNG should obtain its seeds from the outputs of an RNG; i.e., a PRNG requires a RNG as a companion.

The outputs of a PRNG are typically deterministic functions of the seed; i.e., all true randomness is confined to seed generation. The deterministic nature of the process leads to the term "pseudorandom." Since each element of a pseudorandom sequence is reproducible from its seed, only the seed needs to be saved if reproduction or validation of the pseudorandom sequence is required.

Ironically, pseudorandom numbers often appear to be more random than random numbers obtained from physical sources. If a pseudorandom sequence is properly constructed, each value in the sequence is produced from the previous value via transformations which appear to introduce additional randomness. A series of such transformations can eliminate statistical auto-correlations between input and output. Thus, the outputs of a PRNG may have better statistical properties and be produced faster than an RNG.

1.1.5 Testing

Various statistical tests can be applied to a sequence to attempt to compare and evaluate the sequence to a truly random sequence. Randomness is a probabilistic property; that is, the properties of a random sequence can be characterized and described in terms of probability. The likely outcome of statistical tests, when applied to a truly random sequence, is known a priori and can be described in probabilistic terms. There are an infinite number of possible statistical tests, each assessing the presence or absence of a "pattern" which, if detected, would indicate that the sequence is nonrandom. Because there are so many tests for judging whether a sequence is random or not, no specific finite set of tests is deemed "complete." In addition, the results of statistical testing must be interpreted with some care and caution to avoid incorrect conclusions about a specific generator (see Section 4).

A statistical test is formulated to test a specific ***null hypothesis*** (H_0). For the purpose of this document, the null hypothesis under test is that the sequence being tested is *random*. Associated with this null hypothesis is the alternative hypothesis (H_a) which, for this document, is that the sequence is *not* random. For each applied test, a decision or conclusion is derived that accepts or rejects the null hypothesis, i.e., whether the generator is (or is not) producing random values, based on the sequence that was produced.

For each test, a relevant randomness 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, say out at the 99 % point). During a test, a test statistic value is computed on the data (the sequence being tested). This test statistic value is compared to the critical value. If the test statistic value exceeds the critical value, the null hypothesis for randomness is rejected. Otherwise, the null hypothesis (the randomness hypothesis) is *not* rejected (i.e., the hypothesis is accepted).

In practice, the reason that statistical hypothesis testing works is that the reference distribution and the critical value are dependent on and generated under a tentative assumption of randomness. If the randomness assumption is, in fact, true for the data at hand, then the resulting calculated test statistic value on the data will have a very low probability (e.g., 0.01 %) of exceeding the critical value.

On the other hand, if the calculated test statistic value *does* exceed the critical value (i.e., if the low probability event does in fact occur), then from a statistical hypothesis testing point of view, the low probability event should *not* occur naturally. Therefore, when the calculated test statistic value exceeds the critical value, the conclusion is made that the original assumption of randomness is suspect or faulty. In this case, statistical hypothesis testing yields the following conclusions: reject H_0 (randomness) and accept H_a (non-randomness).

Statistical hypothesis testing is a conclusion-generation procedure that has two possible outcomes, either accept H_0 (the data is random) or accept H_a (the data is non-random). The following 2 by 2 table relates the true (unknown) status of the data at hand to the conclusion arrived at using the testing procedure.

TRUE SITUATION	CONCLUSION	
	Accept H_0	Accept H_a (reject H_0)
Data is random (H_0 is true)	No error	Type I error
Data is not random (H_a is true)	Type II error	No error

If the data is, in truth, random, then a conclusion to reject the null hypothesis (i.e., conclude that the data is non-random) will occur a small percentage of the time. This conclusion is called a Type I error. If the data is, in truth, non-random, then a conclusion to accept the null hypothesis (i.e., conclude that the data is actually random) is called a Type II error. The conclusions to accept H_0 when the data is really random, and to reject H_0 when the data is non-random, are correct.

The probability of a Type I error is often called the ***level of significance*** of the test. This probability can be set prior to a test and is denoted as a. For the test, a is the probability that the test will indicate that the sequence is not random when it really is random. That is, a sequence appears to have non-random properties even when a “good” generator produced the sequence. Common values of a in cryptography are about 0.01.

The probability of a Type II error is denoted as b. For the test, b is the probability that the test will indicate that the sequence is random when it is not; that is, a “bad” generator produced a

sequence that appears to have random properties. Unlike a, b is not a fixed value. b can take on many different values because there are an infinite number of ways that a data stream can be non-random, and each different way yields a different b. The calculation of the Type II error b is more difficult than the calculation of a because of the many possible types of non-randomness.

One of the primary goals of the following tests is to minimize the probability of a Type II error, i.e., to minimize the probability of accepting a sequence being produced by a good generator when the generator was actually bad. The probabilities a and b are related to each other and to the size n of the tested sequence in such a way that if two of them are specified, the third value is automatically determined. Practitioners usually select a sample size n and a value for a (the probability of a Type I error – the level of significance). Then a critical point for a given statistic is selected that will produce the smallest b (the probability of a Type II error). That is, a suitable sample size is selected along with an acceptable probability of deciding that a bad generator has produced the sequence when it really is random. Then the cutoff point for acceptability is chosen such that the probability of falsely accepting a sequence as random has the smallest possible value.

Each test is based on a calculated test statistic value, which is a function of the data. If the test statistic value is S and the critical value is t, then the Type I error probability is $P(S > t \,||\, H_o$ is true) = P(reject $H_o \mid H_0$ is true), and the Type II error probability is $P(S \leq t \,||\, H_0$ is false) = P(accept $H_0 \mid H_0$ is false). The test statistic is used to calculate a *P-value* that summarizes the strength of the evidence against the null hypothesis. For these tests, each *P-value* is the probability that a perfect random number generator would have produced a sequence less random than the sequence that was tested, given the kind of non-randomness assessed by the test. 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 (a) can be chosen for the tests. If *P-value* ³ a, then the null hypothesis is accepted; i.e., the sequence appears to be random. If *P-value* < a, then the null hypothesis is rejected; i.e., the sequence appears to be non-random. The parameter a denotes the probability of the Type I error. Typically, a is chosen in the range [0.001, 0.01].

- An a of 0.001 indicates that one would expect one sequence in 1000 sequences to be rejected by the test if the sequence was random. For a *P-value* ³ *0.001*, a sequence would be considered to be random with a confidence of 99.9 %. For a *P-value* < *0.001*, a sequence would be considered to be non-random with a confidence of 99.9 %.

- An a of 0.01 indicates that one would expect 1 sequence in 100 sequences to be rejected. A *P-value* ³ 0.01 would mean that the sequence would be considered to be random with a confidence of 99 %. A *P-value* < *0.01* would mean that the conclusion was that the sequence is non-random with a confidence of 99 %.

For the examples in this document, a has been chosen to be 0.01. Note that, in many cases, the parameters in the examples do not conform to the recommended values; the examples are for illustrative purposes only.

1.1.6 Considerations for Randomness, Unpredictability and Testing

The following assumptions are made with respect to random binary sequences to be tested:

1. Uniformity: At any point in the generation of a sequence of random or pseudorandom bits, the occurrence of a zero or one is equally likely, i.e., the probability of each is exactly 1/2. The expected number of zeros (or ones) is $n/2$, where n = the sequence length.

2. Scalability: Any test applicable to a sequence can also be applied to subsequences extracted at random. If a sequence is random, then any such extracted subsequence should also be random. Hence, any extracted subsequence should pass any test for randomness.

3. Consistency: The behavior of a generator must be consistent across starting values (seeds). It is inadequate to test a PRNG based on the output from a single seed, or an RNG on the basis of an output produced from a single physical output.

1.2 Definitions and Abbreviations

Term	Definition
Asymptotic Analysis	A statistical technique that derives limiting approximations for functions of interest.
Asymptotic Distribution	The limiting distribution of a test statistic arising when n approaches infinity.
Bernoulli Random Variable	A random variable that takes on the value of one with probability p and the value of zero with probability $1-p$.
Binary Sequence	A sequence of zeroes and ones.
Binomial Distribution	A random variable is binomially distributed if there is an integer n and a probability p such that the random variable is the number of successes in n Bernoulli experiments, where the probability of success in a single experiment is p. In a Bernoulli experiment, there are only two possible outcomes.
Bit String	A sequence of bits.
Block	A subset of a bit string. A block has a predetermined length.
Central Limit Theorem	For a random sample of size n from a population with mean m and variance s^2, the distribution of the sample means is

	approximately normal with mean m and variance s^2/n as the sample size increases.
Complementary Error Function	See Erfc.
Confluent Hypergeometric Function	The confluent hypergeometric function is defined as $\Phi(a;b;z) = \frac{\Gamma(b)}{\Gamma(a)\Gamma(b-a)} \int_0^1 e^{zt} t^{a-1} (1-t)^{b-a-1} dt$.
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 (i.e., a test statistic value) that, by construction, has a small probability of occurring (e.g., 5 %) when the null hypothesis of randomness is true.
Cumulative Distribution Function (CDF) $F(x)$	A function giving the probability that the random variable X is less than or equal to x, for every value x. That is, $F(x) = P(X \leq x)$.
Entropy	A measure of the disorder or randomness in a closed system. The entropy of uncertainty of a random variable X with probabilities $p_i, \ldots, p_n$ is defined to be $H(X) = -\sum_{i=1}^{n} p_i \log p_i$.
Entropy Source	A physical source of information whose output either appears to be random in itself or by applying some filtering/distillation process. This output is used as input to either a RNG or PRNG.
Erfc	The complementary error function $erfc(z)$ is defined in Section 5.5.3. This function is related to the normal cdf.
igamc	The incomplete gamma function $Q(a,x)$ is defined in Section 5.5.3.
Geometric Random Variable	A random variable that takes the value k, a non-negative integer with probability $p^k(1-p)$. The random variable x is the number of successes before a failure in an indefinite series of Bernoulli trials.
Global Structure/Global Value	A structure/value that is available by all routines in the test code.
GUI	Graphical User Interface.

Incomplete Gamma Function	See the definition for igamc.
Hypothesis (Alternative)	A statement H_a that an analyst will consider as true (e.g., H_a: the sequence is non-random) if and when the null hypothesis is determined to be false.
Hypothesis (Null)	A statement H_0 about the assumed default condition/property of the observed sequence. For the purposes of this document, the null hypothesis H_0 is that the sequence is random. If H_0 is in fact true, then the reference distribution and critical values of the test statistic may be derived.
Kolmogorov-Smirnov Test	A statistical test that may be used to determine if a set of data comes from a particular probability distribution.
Level of Significance (a)	The probability of falsely rejecting the null hypothesis, i.e., the probability of concluding that the null hypothesis is false when the hypothesis is, in fact, true. The tester usually chooses this value; typical values are 0.05, 0.01 or 0.001; occasionally, smaller values such as 0.0001 are used. The level of significance is the probability of concluding that a sequence is non-random when it is in fact random. Synonyms: Type I error, a error.
Linear Dependence	In the context of the binary rank matrix test, linear dependence refers to m-bit vectors that may be expressed as a linear combination of the linearly independent m-bit vectors.
Maple	An interactive computer algebra system that provides a complete mathematical environment for the manipulation and simplification of symbolic algebraic expressions, arbitrary extended precision mathematics, two- and three-dimensional graphics, and programming.
MATLAB	An integrated, technical computer environment that combines numeric computation, advanced graphics and visualization, and a high level programming language. MATLAB includes functions for data analysis and visualization; numeric and symbolic computation; engineering and scientific graphics; modeling, simulation and prototyping; and programming, application development and a GUI design.
Normal (Gaussian) Distribution	A continuous distribution whose density function is given by

	$f(x;m;s) = \frac{1}{\sqrt{2ps^2}} e^{-\frac{1}{2}\left(\frac{x-m}{s}\right)^2}$, where μ and σ are location and scale parameters.
P-value	The probability (under the null hypothesis of randomness) that the chosen test statistic will assume values that are equal to or worse than the observed test statistic value when considering the null hypothesis. The *P-value* is frequently called the "tail probability."
Poisson Distribution - §3.8	Poisson distributions model (some) discrete random variables. Typically, a Poisson random variable is a count of the number of rare events that occur in a certain time interval.
Probability Density Function (PDF)	A function that provides the "local" probability distribution of a test statistic. From a finite sample size *n*, a probability density function will be approximated by a histogram.
Probability Distribution	The assignment of a probability to the possible outcomes (realizations) of a random variable.
Pseudorandom Number Generator (PRNG)	A *deterministic algorithm* which, given a truly random binary sequence of length *k*, outputs a binary sequence of length $l >> k$ which appears to be random. The input to the generator is called the seed, while the output is called a pseudorandom bit sequence.
Random Number Generator (RNG)	A mechanism that purports to generate truly random data.
Random Binary Sequence	A sequence of bits for which the probability of each bit being a "0" or "1" is ½. The value of each bit is independent of any other bit in the sequence, i.e., each bit is unpredictable.
Random Variable	Random variables differ from the usual deterministic variables (of science and engineering) in that random variables allow the systematic distributional assignment of probability values to each possible outcome.
Rank (of a matrix)	Refers to the rank of a matrix in linear algebra over GF(2). Having reduced a matrix into 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.

Run	An uninterrupted sequence of like bits (i.e., either all zeroes or all ones).
Seed	The input to a pseudorandom number generator. Different seeds generate different pseudorandom sequences.
SHA-1	The Secure Hash Algorithm defined in Federal Information Processing Standard 180-1.
Standard Normal Cumulative Distribution Function	See the definition in Section 5.5.3. This is the normal function for mean = 0 and variance = 1.
Statistically Independent (Events)	Two events are independent if the occurrence of one event does not affect the chances of the occurrence of the other event. The mathematical formulation of the independence of events A and B is the probability of the occurrence of both A and B being equal to the product of the probabilities of A and B (i.e., $P(A \text{ and } B) = P(A)P(B)$).
Statistical Test (of a Hypothesis)	A function of the data (binary stream) which is computed and used to decide whether or not to reject the null hypothesis. A systematic statistical rule whose purpose is to generate a conclusion regarding whether the experimenter should accept or reject the null hypothesis H_o.
Word	A predefined substring consisting of a fixed pattern/template (e.g., 010, 0110).

Abbreviation	Definition
ANSI	American National Standards Institute
FIPS	Federal Information Processing Standard
NIST	National Institute of Standards and Technology
RNG	Random Number Generator
SHA-1	Secure Hash Algorithm

1.3 Mathematical Symbols

In general, the following notation is used throughout this document. However, the tests in this document have been designed and described by multiple authors who may have used slightly different notation. The reader is advised to consider the notation used for each test separate from that notation used in other tests.

Symbol	Meaning
ëxû	The floor function of *x*; for a given real positive *x*, *ëxû* = *x-g*, where *ëxû* is a non-negative integer, and *0 £ g <* 1.
a	The significance level.
d	The normalized difference between the observed and expected number of frequency components. See Sections 2.6 and 3.6.
Ñy²m(obs); Ñ²y²m(obs)	A measure of how well the observed values match the expected value. See Sections 2.12 and 3.12.
E[ ]	The expected value of a random variable.
e	The original input string of zero and one bits to be tested.
e_i	The ith bit in the original sequence *e*.
H_0	The null hypothesis; i.e., the statement that the sequence is random.
log(x)	The natural logarithm of *x*: $log(x) = log_e(x) = ln(x)$.
$log_2(x)$	Defined as $\frac{ln(x)}{ln(2)}$, where *ln* is the natural logarithm.
M	The number of bits in a substring (block) being tested.
N	The number of *M*-bit blocks to be tested.
n	The number of bits in the stream being tested.
f_n	The sum of the log_2 distances between matching *L*-bit templates, i.e., the sum of the number of digits in the distance between *L*-bit templates. See Sections 2.9 and 3.9.
p	3.14159… unless defined otherwise for a specific test.
p_i	The average number of ones in a string of *n* bits.

s	The standard deviation of a random variable = $\sqrt{\int (x-m)^2 f(x)dx}$.
s^2	The variance of a random variable = (standard deviation)^2.
s_{obs}	The observed value which is used as a statistic in the Frequency test.
S_n	The nth partial sum for values X_i = {-1, +1}; i.e., the sum of the first n values of X_i.
S	The summation symbol.
F	Standard Normal Cumulative Distribution Function (see Section 5.5.3).
x_j	The total number of times that a given state occurs in the identified cycles. See Section 2.16 and 3.16.
X_i	The elements of the string consisting of ±1 that is to be tested for randomness, where $X_i = 2e_i-1$.
c^2	The [theoretical] chi-square distribution; used as a test statistic; also, a test statistic that follows the c^2 distribution.
$c^2(obs)$	The chi-square statistic computed on the observed values. See Sections 2.2, 2.4, 2.5, 2.7, 2.8, 2.11, 2.13, 2.15, and the corresponding sections of Section 3.
V_n	The expected number of runs that would occur in a sequence of length n under an assumption of randomness See Sections 2.3 and 3.3.
$V_n(obs)$	The observed number of runs in a sequence of length n. See Sections 2.3 and 3.3.
W	The expected number of words in a bitstring being tested.
W_{obs}	The number of disjoint words in a sequence. See Sections 2.10 and 3.10.

2 RANDOM NUMBER GENERATION TESTS

The NIST Test Suite is a statistical package consisting of 16 tests that were developed to test the randomness of (arbitrarily long) binary sequences produced by either hardware or software based cryptographic random or pseudorandom number generators. These tests focus on a variety of different types of non-randomness that could exist in a sequence. Some tests are decomposable into a variety of subtests. The 16 tests are:

1. The Frequency (Monobit) Test,
2. Frequency Test within a Block,
3. The Runs Test,
4. Test for the Longest-Run-of-Ones in a Block,
5. The Binary Matrix Rank Test,
6. The Discrete Fourier Transform (Spectral) Test,
7. The Non-overlapping Template Matching Test,
8. The Overlapping Template Matching Test,
9. Maurer's "Universal Statistical" Test,
10. The Lempel-Ziv Compression Test,
11. The Linear Complexity Test,
12. The Serial Test,
13. The Approximate Entropy Test,
14. The Cumulative Sums (Cusums) Test,
15. The Random Excursions Test, and
16. The Random Excursions Variant Test.

This section (Section 2) consists of 16 subsections, one subsection for each test. Each subsection provides a high level description of the particular test. The corresponding subsections in Section 3 provide the technical details for each test.

Section 4 provides a discussion of testing strategy and the interpretation of test results. The order of the application of the tests in the test suite is arbitrary. However, it is recommended 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: The most time-consuming statistical test is the Linear Complexity test; see Sections 2.11 and 3.11).

Section 5 provides a user's guide for setting up and running the tests, and a discussion on program layout. The statistical package includes source code and sample data sets. The test code was developed in ANSI C. Some inputs are assumed to be global values rather than calling parameters.

A number of tests in the test suite have the ***standard normal*** and the ***chi-square (χ^2)*** as reference distributions. If the sequence under test is in fact non-random, the calculated test statistic will fall in extreme regions of the reference distribution. The standard normal

distribution (i.e., the bell-shaped curve) is used to compare the value of the test statistic obtained from the RNG with the expected value of the statistic under the assumption of randomness. The test statistic for the standard normal distribution is of the form $z = (x - m)/s$, where x is the sample test statistic value, and m and s^2 are the expected value and the variance of the test statistic. The χ^2 distribution (i.e., a left skewed curve) is used to compare the goodness-of-fit of the observed frequencies of a sample measure to the corresponding expected frequencies of the hypothesized distribution. The test statistic is of the form $\chi^2 = \sum\left(\left(o_i - e_i\right)^2 / e_i\right)$, where o_i and e_i are the observed and expected frequencies of occurrence of the measure, respectively.

For many of the tests in this test suite, the assumption has been made that the size of the sequence length, n, is large (of the order 10^3 to 10^7). 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 distributions would be inappropriate and would need to be replaced by exact distributions that would commonly be difficult to compute.

Note: For many of the examples throughout Section 2, small sample sizes are used for illustrative purposes only, e.g., $n = 10$. The normal approximation is not really applicable for these examples.

2.1 Frequency (Monobit) Test

2.1.1 Test Purpose

The focus of the 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 ½, that is, the number of ones and zeroes in a sequence should be about the same. All subsequent tests depend on the passing of this test; there is no evidence to indicate that the tested sequence is non-random.

2.1.2 Function Call

Frequency(n), where:

n The length of the bit string.

Additional input used by the function, but supplied by the testing code:

ε The sequence of bits as generated by the RNG or PRNG being tested; this exists as a global structure at the time of the function call; $e = e_1, e_2, \ldots, e_n$.

2.1.3 Test Statistic and Reference Distribution

s_{obs}: The absolute value of the sum of the X_i (where, $X_i = 2e - 1 = \pm 1$) in the sequence divided by the square root of the length of the sequence.

The reference distribution for the test statistic is half normal (for large n). (Note: If z (where $z = s_{obs} / \sqrt{2}$; see Section 3.1) is distributed as normal, then $|z|$ is distributed as half normal.) If the sequence is random, then the plus and minus ones will tend to cancel one another out so that the test statistic will be about 0. If there are too many ones or too many zeroes, then the test statistic will tend to be larger than zero.

2.1.4 Test Description

(1) Conversion to ±1: The zeros and ones of the input sequence (e) are converted to values of –1 and +1 and are added together to produce $S_n = X_1 + X_2 + \cdots + X_n$, where $X_i = 2e_i - 1$.

For example, if $e = 1011010101$, then n=10 and $S_n = 1 + (-1) + 1 + 1 + (-1) + 1 + (-1) + 1 + (-1) + 1 = 2$.

(2) Compute the test statistic $s_{obs} = \frac{|S_n|}{\sqrt{n}}$

For the example in this section, $s_{obs} = \frac{|2|}{\sqrt{10}} = .632455532$.

(3) Compute $P\text{-}value = \boldsymbol{erfc}\left(\frac{s_{obs}}{\sqrt{2}}\right)$, where ***erfc*** is the complementary error function as defined in Section 5.5.3.3.

For the example in this section, $P\text{-}value = \boldsymbol{erfc}\left(\frac{.632455532}{\sqrt{2}}\right) = 0.527089$.

2.1.5 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.

2.1.6 Conclusion and Interpretation of Test Results

Since the *P-value* obtained in step 3 of Section 2.1.4 is ≥ 0.01 (i.e., *P-value* = 0.527089), the conclusion is that the sequence is random.

Note that if 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 are indicative of too many ones, and large negative values of S_n are indicative of too many zeros.

2.1.7 Input Size Recommendations

It is recommended that each sequence to be tested consist of a minimum of 100 bits (i.e., $n \geq 100$).

2.1.8 Example

(input) e = 1100100100001111110110101010001000100001011010001100001101001100010011000110011000101000101110000

(input) $n = 100$

(processing) $S_{100} = -16$

(processing) $s_{obs} = 1.6$

(output) *P-value* = 0.109599

(conclusion) Since *P-value* ≥ 0.01, accept the sequence as random.

2.2 Frequency Test within a Block

2.2.1 Test Purpose

The focus of the 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 an assumption of randomness. For block size M=1, this test degenerates to test 1, the Frequency (Monobit) test.

2.2.2 Function Call

BlockFrequency(M,n), where:

M The length of each block.

n The length of the bit string.

Additional input used by the function, but supplied by the testing code:

ε The sequence of bits as generated by the RNG or PRNG being tested; this exists as a global structure at the time of the function call; $e = e_1, e_2, \ldots, e_n$.

2.2.3 Test Statistic

$c^2(obs)$: A measure of how well the observed proportion of ones within a given M-bit block match the expected proportion (1/2).

The reference distribution for the test statistic is a χ^2 distribution.

2.2.4 Test Description

(1) Partition the input sequence into $N = \left\lfloor \frac{n}{M} \right\rfloor$ non-overlapping blocks. Discard any unused bits.

For example, if $n = 10$, $M = 3$ and $e = 0110011010$, 3 blocks ($N = 3$) would be created, consisting of *011, 001 and 101*. The final *0* would be discarded.

(2) Determine the proportion p_i of ones in each M-bit block using the equation

$$p_i = \frac{\sum_{j=1}^{M} e_{(i-1)M+j}}{M}, \text{ for } 1 \le i \le N.$$

For the example in this section, $p_1 = 2/3$, $p_2 = 1/3$, and $p_3 = 2/3$.

(3) Compute the χ^2 statistic: $c^2(obs) = 4\,M \sum_{i=1}^{N} (p_i - ½)^2$.

For the example in this section, $\chi^2(obs) = 4 \times 3 \times \left(\left(\frac{2}{3} - \frac{1}{2}\right)^2 + \left(\frac{1}{3} - \frac{1}{2}\right)^2 + \left(\frac{2}{3} - \frac{1}{2}\right)^2 \right) = 1$.

(4) Compute *P-value* = **igamc** ($N/2$, $c^2(obs)/2$) , where **igamc** is the incomplete gamma function for Q(a,x) as defined in Section 5.5.3.3.

Note: When comparing this section against the technical description in Section 3.2, note that $Q(a,x) = 1-P(a,x)$.

For the example in this section, *P-value* = $\mathbf{igamc}\left(\frac{3}{2}, \frac{1}{2}\right) = 0.801252$.

2.2.5 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.

2.2.6 Conclusion and Interpretation of Test Results

Since the *P-value* obtained in step 4 of Section 2.2.4 is ≥ 0.01 (i.e., *P-value* = 0.801252), the conclusion is that the sequence is random.

Note that 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.

2.2.7 Input Size Recommendations

It is recommended that each sequence to be tested consist of a minimum of 100 bits (i.e., $n \geq 100$). Note that $n \geq MN$. The block size M should be selected such that $M \geq 20$, $M > .01n$ and $N < 100$.

2.2.8 Example

(input) e = 1100100100001111110110101010001000100001011010001100001101001100010011000110011000101000101110000

(input) $n = 100$

(input) $M = 10$

(processing) $N = 10$

(processing) $c^2 = 7.2$

(output) *P-value* = 0.706438

(conclusion) Since *P-value* ³ 0.0, accept the sequence as random.

2.3 Runs Test

2.3.1 Test Purpose

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

2.3.2 Function Call

Runs(*n*), where:

n The length of the bit string.

Additional inputs for the function, but supplied by the testing code:

ε The sequence of bits as generated by the RNG or PRNG being tested; this exists as a global structure at the time of the function call; $e = e_1, e_2, \ldots, e_n$.

2.3.3 Test Statistic and Reference Distribution

$V_n(obs)$: The total number of runs (i.e., the total number of zero runs + the total number of one-runs) across all *n* bits.

The reference distribution for the test statistic is a χ^2 distribution.

2.3.4 Test Description

Note: The Runs test carries out a Frequency test as a prerequisite.

(1) Compute the pre-test proportion *p* of ones in the input sequence: $p = \frac{\sum_j e_j}{n}$.

For example, if $e = 1001101011$, then n=10 and $p = 6/10 = 3/5$.

(2) Determine if the prerequisite Frequency test is passed: If it can be shown that $\left| p - \frac{1}{2} \right| \geq t$, then the Runs test need not be performed (i.e., the test should not have been run because of a failure to pass test 1, the Frequency (Monobit) test). If the test is not applicable, then the *P-value* is set to 0.0000. Note that for this test, $t = \frac{2}{\sqrt{n}}$ has been pre-defined in the test code.

For the example in this section, since $t = 2/\sqrt{10} \approx 0.63246$, then $|p - 1/2| = | 3/5 - 1/2 | = 0.1 < t$, and the test is not run.

Since the observed value p is within the selected bounds, the runs test is applicable.

(3) Compute the test statistic $V_n(obs) = \sum_{k=1}^{n-1} r(k) + 1$, where $r(k)=0$ if $e_k=e_{k+1}$, and $r(k)=1$ otherwise.

Since e = 1 00 11 0 1 0 11, then

$$V_{10}(obs)=(1+0+1+0+1+1+1+1+0)+1=7.$$

(4) Compute $P\text{-}value = \mathbf{erfc}\left(\frac{|V_n(obs) - 2np(1-p)|}{2\sqrt{2np}(1-p)}\right)$.

For the example, $P\text{-}value = \mathbf{erfc}\left(\frac{7 - \left(2 \bullet 10 \bullet \frac{3}{5}\left(1 - \frac{3}{5}\right)\right)}{2 \bullet \sqrt{2 \bullet 10} \bullet \frac{3}{5} \bullet \left(1 - \frac{3}{5}\right)}\right) = 0.147232.$

2.3.5 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.

2.3.6 Conclusion and Interpretation of Test Results

Since the *P-value* obtained in step 4 of Section 2.3.4 is ≥ 0.01 (i.e., *P-value* = 0.147232), the conclusion is that the sequence is random.

Note that a large value for $V_n(obs)$ would have indicated an oscillation in the string which is too fast; a small value would have indicated that the oscillation is too slow. (An oscillation is considered to be a change from a one to a zero or vice versa.) A fast oscillation occurs when there are a lot of changes, e.g., 010101010 oscillates with every bit. A stream with a slow oscillation has fewer runs than would be expected in a random sequence, e.g., 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.

2.3.7 Input Size Recommendations

It is recommended that each sequence to be tested consist of a minimum of 100 bits (i.e., $n \geq 100$).

2.3.8 Example

(input) e = 1100100100001111110110101010001000100001011010001100001101001100010011000110011000101000101110000

(input) $n = 100$

(input) $t = 0.02$

(processing) $p = 0.42$

(processing) $V_n(obs) = 52$

(output) *P-value* = 0.500798

(conclusion) Since *P-value* ³ *0.01*, accept the sequence as random.

2.4 Test for the Longest Run of Ones in a Block

2.4.1 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 in 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. See Section 4.4.

2.4.2 Function Call

LongestRunOfOnes(n), where:

n The length of the bit string.

Additional input for the function supplied by the testing code:

ε The sequence of bits as generated by the RNG or PRNG being tested; this exists as a global structure at the time of the function call; $e = e_1, e_2, \ldots, e_n$.

M The length of each block. The test code has been pre-set 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
750,000	10^4

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

2.4.3 Test Statistic and Reference Distribution

$c^2(obs)$: A measure of how well the observed longest run length within *M*-bit blocks matches the expected longest length within *M*-bit blocks.

The reference distribution for the test statistic is a χ^2 distribution.

2.4.4 Test Description

(1) Divide the sequence into *M*-bit blocks.

(2) Tabulate the frequencies n_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 the values of *M* supported by the test code, the v_i cells will hold the following counts:

v_i	M = 8	M = 128	M = 10^4
v_0	≤ 1	≤ 4	≤ 10
v_1	2	5	11
v_2	3	6	12
v_3	≥ 4	7	13
v_4		8	14
v_5		≥ 9	15
v_6			≥ 16

For an example, see Section 2.4.8.

(3) Compute $c^2(obs) = \sum_{i=0}^{K} \frac{(v_i - Np_i)^2}{Np_i}$, where the values for p_i are provided in 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
10^4	6	75

For the example of 2.4.8,

$$c^2(obs) = \frac{(4-16(.2148))^2}{16(.2148)} + \frac{(9-16(.3672))^2}{16(.3672)} + \frac{(3-16(.2305))^2}{16(.2305} + \frac{(0-16(.1875))^2}{16(.1875)} = 4.882605$$

(4) Compute *P-value* = **igamc** $\left(\frac{K}{2}, \frac{c^2(obs)}{2}\right)$.

For the example, *P-value* = **igamc** $\left(\frac{3}{2}, \frac{4.882605}{2}\right) =$ 0.180598.

2.4.5 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.

2.4.6 Conclusion and Interpretation of Test Results

For the example in Section 2.4.8, since the P-value ≥ 0.01 (*P-value = 0.180609)*, the conclusion is that the sequence is random. Note that large values of $c^2(obs)$ indicate that the tested sequence has clusters of ones.

2.4.7 Input Size Recommendations

It is recommended that each sequence to be tested consist of a minimum of bits as specified in the table in Section 2.4.2.

2.4.8 Example

For the case where K = 3 and M = 8:

(input) ε = 11001100000101010110110001001100111000000000001001
0011010101000100010011110101101000000011010111110 0
11001110011011011000101100 10

(input) n = 128

(processing)

Subblock	Max-Run	Subblock	Max-Run
11001100	(2)	00010101	(1)
01101100	(2)	01001100	(2)
11100000	(3)	00000010	(1)
01001101	(2)	01010001	(1)
00010011	(2)	11010110	(2)
10000000	(1)	11010111	(3)
11001100	(2)	11100110	(3)
11011000	(2)	10110010	(2)

(processing) $n_0 = 4;\ n_1 = 9;\ n_2 = 3;\ n_4 = 0;\ c^2 = 4.882457$

(output) *P-value = 0.180609*

(conclusion) Since the *P-value* is ≥ 0.01, accept the sequence as random.

2.5 Binary Matrix Rank Test

2.5.1 Test Purpose

The focus of the test is the rank of disjoint sub-matrices of the entire sequence. The purpose of this test is to check for linear dependence among fixed length substrings of the original sequence. Note that this test also appears in the DIEHARD battery of tests [7].

2.5.2 Function Call

Rank(n), where:

n The length of the bit string.

Additional input used by the function supplied by the testing code:

ε The sequence of bits as generated by the RNG or PRNG being tested; this exists as a global structure at the time of the function call; $e = e_1, e_2, \ldots, e_n$.

M The number of rows in each matrix. For the test suite, M has been set to 32. If other values of M are used, new approximations need to be computed.

Q The number of columns in each matrix. For the test suite, Q has been set to 32. If other values of Q are used, new approximations need to be computed.

2.5.3 Test Statistic and Reference Distribution

$c^2(obs)$: A measure of how well the observed number of ranks of various orders match the expected number of ranks under an assumption of randomness.

The reference distribution for the test statistic is a χ^2 distribution.

2.5.4 Test Description

(1) Sequentially divide the sequence into $M \bullet Q$-bit disjoint blocks; there will exist $N = \left\lfloor \frac{n}{MQ} \right\rfloor$ such blocks. Discarded bits will be reported as not being used in the computation within each block. Collect the $M \bullet Q$ bit segments into M by Q matrices. Each row of the matrix is filled with successive Q-bit blocks of the original sequence e.

For example, if $n = 20$, $M = Q = 3$, and $e = 01011001001010101101$, then partition the stream into $N = \left\lfloor \frac{n}{3 \bullet 3} \right\rfloor = 2$ matrices of cardinality $M \bullet Q$ ($3 \bullet 3 = 9$). Note that the last two bits (0 and 1) will be discarded. The two matrices are $\begin{vmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \end{vmatrix}$ and $\begin{vmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{vmatrix}$. 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 three bits in row 3. The second matrix is similarly constructed using the next nine bits in the sequence.

(2) Determine the binary rank (R_ℓ) of each matrix, where $\ell = 1, \ldots, N$. The method for determining the rank is described in Appendix A.

For the example in this section, the rank of the first matrix is 2 ($R_1 = 2$), and the rank of the second matrix is 3 ($R_2 = 3$).

(3) Let F_M = the number of matrices with $R_\ell = M$ (full rank),
F_{M-1} = the number of matrices with $R_\ell = M-1$ (full rank - 1),
$N - F_M - F_{M-1}$ = the number of matrices remaining.

For the example in this section, $F_M = F_3 = 1$ (R_2 has the full rank of 3), $F_{M-1} = F_2 = 1$ (R_1 has rank 2), and no matrix has any lower rank.

(4) Compute

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

For the example in this section,

$$c^2(obs) = \frac{(1-0.2888\bullet 2)^2}{0.2888\bullet 2} + \frac{(1-0.5776\bullet 2)^2}{0.5776\bullet 2} + \frac{(2-1-1-0.1336\bullet 2)^2}{0.1336\bullet 2} = 0.596953.$$

(5) Compute $P-value = e^{-c^2(obs)/2}$. Since M = 3 in the example, the *P-value* is equal to $igamc\left(\frac{3}{2}, \frac{c^2(obs)}{2}\right)$.

For the example in this section, *P-value* = $e^{0.596953/2}$ = 0.741948.

2.5.5 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.

2.5.6 Conclusion and Interpretation of Test Results

Since the *P-value* obtained in step 5 of Section 2.5.4 is ≥ 0.01 (*P-value* = 0.741948), the conclusion is that the sequence is random.

Note that large values of $c^2(obs)$ (and hence, small *P-values*) would have indicated a deviation of the rank distribution from that corresponding to a random sequence.

2.5.7 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 to be tested should consist of a minimum of 38,912 bits.

2.5.8 Example

(input) e = the first 100,000 binary digits in the expansion of e

(input) n = 100000, $M = Q$ = 32 (NOTE: 672 BITS WERE DISCARDED.)

(processing) N = 97

(processing) F_M = 23, F_{M-1} = 60, $N - F_M - F_{M-1}$ = 14

(processing) $c^2 = 1.2619656$

(output) *P-value* = 0.532069

(conclusion) Since *P-value* ³ *0.01*, accept the sequence as random.

2.6 Discrete Fourier Transform (Spectral) Test

2.6.1 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 %.

2.6.2 Function Call

DiscreteFourierTransform(n), where:

n The length of the bit string.

Additional input used by the function, but supplied by the testing code:

ε The sequence of bits as generated by the RNG or PRNG being tested; this exists as a global structure at the time of the function call; $e = e_1, e_2, \ldots, e_n$.

2.6.3 Test Statistic and Reference Distribution

d: The normalized difference between the observed and the expected number of frequency components that are beyond the 95 % threshold.

The reference distribution for the test statistic is the normal distribution.

2.6.4 Test Description

(1) The zeros and ones of the input sequence (e) are converted to values of –1 and +1 to create the sequence $X = x_1, x_2, \ldots, x_n$, where $x_i = 2e_i - 1$.

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

(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 (see Section 3.6 for a sample diagram of a DFT result).

(3) Calculate $M = modulus(S') \equiv |S'|$, 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 $T = \sqrt{3n}$ = 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.

For the example in this section, $N_0 = 4.75$.

(6) Compute N_1 = the actual observed number of peaks in M that are less than T.

For the example in this section, $N_1 = 4$.

(7) Compute $d = \dfrac{(N_1 - N_0)}{\sqrt{n(.95)(.05)/2}}$.

For the example in this section, $d = \dfrac{(4 - 4.75)}{\sqrt{10(.95)(.05)/2}}$ = -1.538968.

(8) Compute $P\text{-}value = erfc\left(\dfrac{|d|}{\sqrt{2}}\right)$.

For the example in this section, $P\text{-}value = erfc\left(\dfrac{1.538968}{\sqrt{2}}\right) = 0.123812$.

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 ≥ 0.01 (P-value = 0.123812), the conclusion is that the sequence is random.

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.

2.6.7 Input Size Recommendations

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

2.6.8 Example

(input) e = 1100100100001111110110101010001000100001011010001100001000110100110001001100011001100010100010111000

(input) n = 100

(processing) N_1 = 46

(processing) N_0 = 47.5

(processing) d = -0.973329

(output) *P-value* = 0.330390

(conclusion) Since *P-value* $\geq$ *0.01*, accept the sequence as random.

2.7 Non-overlapping Template Matching Test

2.7.1 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 for the Overlapping Template Matching test of Section 2.8, 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. If the pattern is found, the window is reset to the bit after the found pattern, and the search resumes.

2.7.2 Function Call

NonOverlappingTemplateMatching(m,n)

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

n The length of the entire bit string under test.

Additional input used by the function, but supplied by the testing code:

ε The sequence of bits as generated by the RNG or PRNG being tested; this exists as a global structure at the time of the function call; $e = e_1, e_2, \ldots, e_n$.

B The *m*-bit template to be matched; *B* is a string of ones and zeros (of length *m*) which is defined in a template library of non-periodic patterns contained within the test code.

M The length in bits of the substring of *e* to be tested. *M* has been set to 131,072 (i.e., 2^{17}) in the test code.

N The number of independent blocks. *N* has been fixed at 8 in the test code.

2.7.3 Test Statistic and Reference Distribution

$c^2(obs)$: A measure of how well the observed number of template "hits" matches the expected number of template "hits" (under an assumption of randomness).

The reference distribution for the test statistic is the c^2 distribution.

2.7.4 Test Description

(1) Partition the sequence into *N* independent blocks of length *M*.

For example, if $e = 10100100101110010110$, then $n = 20$. If $N = 2$ and $M = 10$, then the two blocks would be 1010010010 and 1110010110.

(2) Let W_j ($j = 1, \ldots, N$) be the number of times that *B* (the template) occurs within the block j. Note that $j = 1,\ldots,N$. The search for matches proceeds by creating an *m*-bit window on the sequence, comparing the bits within that window against the template. If there is <u>no</u> 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 <u>is</u> 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.

For the above example, if $m = 3$ and the template $B = 001$, then the examination proceeds as follows:

	Block 1		**Block 2**	
Bit Positions	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 $W_2 = 1$.

(3) Under an assumption of randomness, compute the theoretical mean m and variance s^2:

$$m = (M-m+1)/2^m \qquad s^2 = M\left(\frac{1}{2^m} - \frac{2m-1}{2^{2m}}\right).$$

For the example in this section, $m = (10-3+1)/2^3 = 1$, and

$$s^2 = 10 \bullet \left(\frac{1}{2^3} - \frac{2 \bullet 3 - 1}{2^{2 \bullet 3}}\right) = 0.46875 .$$

(4) Compute $c^2(obs) = \sum_{j=1}^{N} \frac{(W_j - m)^2}{s^2}$.

For the example in this section, $c^2(obs) = \frac{(2-1)^2 + (1-1)^2}{0.46875} = \frac{1+0}{0.46875} = 2.133333$.

(5) Compute *P-value* = **igamc** $\left(\frac{N}{2}, \frac{c^2(obs)}{2}\right)$. 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; for $m = 10$, up to 284 *P-values* may be computed.

For the example in this section, *P-value* = **igamc**$\left(\frac{2}{2}, \frac{2.133333}{2}\right) = 0.344154$.

2.7.5 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.

2.7.6 Conclusion and Interpretation of Test Results

Since the *P-value* obtained in step 5 of Section 2.7.4 is ≥ 0.01 (P-value = 0.344154), the conclusion is that the sequence is random.

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

2.7.7 Input Size Recommendations

The test code has been written to provide templates for $m = 2, 3, \ldots, 10$. It is recommended that $m = 9$ or $m = 10$ be specified to obtain meaningful results. Although $N = 8$ has been specified in the test code, the code may be altered to other sizes. However, N should be chosen such that N £ 100 to be assured that the *P-values* are valid. The test code has been written to assume a sequence length of $n = 10^6$ (entered via a calling parameter) and $M = 131072$ (hard coded). If values other than these are desired, be sure that $M > 0.01 \cdot n$ and $N =$ ën/Mû..

2.7.8 Example

For a template B = 000000001 whose size is $m = 9$:

(input) $e = 2^{20}$ bits produced by the G-SHA-1 generator[1]

(input) $n = 2^{20}$, $B = 000000001$

(processing) $m = 255.984375$ and $s^2 = 247.499999$

(processing) $W_1 = 259$; $W_2 = 229$; $W_3 = 271$; $W_4 = 245$; $W_5 = 272$; $W_6 = 262$; $W_7 = 259$; and $W_8 = 246$

(processing) $c^2(obs) = 5.999377$

(output) *P-value* = 0.647302

(conclusion) Since the *P-value* ³ 0.01, accept the sequence as random.

2.8 Overlapping Template Matching Test

2.8.1 Test Purpose

The focus of the Overlapping Template Matching test is the number of occurrences of pre-specified target strings. Both this test and the Non-overlapping Template Matching test of Section 2.7 use an *m*-bit window to search for a specific *m*-bit pattern. As with the test in Section 2.7, if the pattern is *not* found, the window slides one bit position. The difference between this test and the test in Section 2.7 is that when the pattern *is* found, the window slides only one bit before resuming the search.

[1] Defined in Federal Information Processing Standard (FIPS) 186-2.

2.8.2 Function Call

OverlappingTemplateMatching(m,n)

- m The length in bits of the template – in this case, the length of the run of ones.
- n The length of the bit string.

Additional input used by the function, but supplied by the testing code:

- ε The sequence of bits as generated by the RNG or PRNG being tested; this exists as a global structure at the time of the function call; $e = e_1, e_2, \ldots, e_n$.
- B The m-bit template to be matched.
- K The number of degrees of freedom. K has been fixed at 5 in the test code.
- M The length in bits of a substring of e to be tested. M has been set to 1032 in the test code.
- N The number of independent blocks of n. N has been set to 968 in the test code.

2.8.3 Test Statistic and Reference Distribution

$c^2(obs)$: A measure of how well the observed number of template "hits" matches the expected number of template "hits" (under an assumption of randomness).

The reference distribution for the test statistic is the c^2 distribution.

2.8.4 Test Description

(1) Partition the sequence into N independent blocks of length M.

For example, if e = *1011101111**0010110100**0111001011**1011111000**0101101001*, then n = 50. If $K = 2$, $M = 10$ and $N = 5$, then the five blocks are *1011101111, 0010110100, 0111001011, 1011111000,* and *0101101001* .

(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 $i = 0,\ldots 5$), such that v_0 is incremented when there are no occurrences of B in a substring, v_1 is incremented for one occurrence of B,…and v_5 is incremented for 5 or more occurrences of B.

For the above example, if $m = 2$ and $B = 11$, then the examination of the first block (*1011101111)* proceeds as follows:

Bit Positions	**Bits**	**No. of occurrences of *B* = 11**
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 block 1, 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 4, there are 4 occurrences of 11; v_4 is incremented. In block 5, there is one occurrence of 11; v_1 is incremented.

Therefore, $v_0 = 0$, $v_1 = 1$, $v_2 = 1$, $v_3 = 1$, $v_4 = 1$, $v_5 = 1$ after all blocks have been examined.

(3) Compute values for l and h that will be used to compute the theoretical probabilities p_i corresponding to the classes of v_0:

$$l = (M-m+1)/2^m \qquad h = l/2.$$

For the example in this section, $l = (10-2+1)/2^2 = 2.25$, and $h = l/2=1.125$.

(4) Compute $c^2(obs) = \sum_{i=0}^{5} \frac{(v_i - Np_i)^2}{Np_i}$, where $p_0 = 0.367879$, $p_1 = 0.183940$, $p_2 = 0.137955$, $p_3 = 0.099634$, $p_4 = 0.069935$ and $p_5 = 0.140657$ as computed by the equations specified in Section 3.8.

For the example in this section, the values of p_i were recomputed, since the example doesn't fit the requirements stated in Section 3.8.5. The example is intended only for illustration. The values of p_i are: $p_0 = 0.324652$, $p_1 = 0.182617$, $p_2 = 0.142670$, $p_3 = 0.106645$, $p_4 = 0.077147$, and $p_5 = 0.166269$.

$$c^2(obs) = \frac{(0-5\bullet 0.324652)^2}{5\bullet 0.324652} + \frac{(1-5\bullet 0.182617)^2}{5\bullet 0.182617} + \frac{(1-5\bullet 0.142670)^2}{5\bullet 0.142670} +$$
$$\frac{(1-5\bullet 0.106645)^2}{5\bullet 0.106645} + \frac{(1-5\bullet 0.077147)^2}{5\bullet 0.077147} + \frac{(1-5\bullet 0.166269)^2}{5\bullet 0.166269} = 3.167729.$$

(5) Compute *P-value* = **igamc**$\left(\frac{5}{2}, \frac{c^2(obs)}{2}\right)$.

For the example in this section, *P-value* = **igamc**$\left(\frac{5}{2}, \frac{3.167729}{2}\right)$ = 0.274932.

2.8.5 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.

2.8.6 Conclusion and Interpretation of Test Results

Since the *P-value* obtained in step 4 of Section 2.8.4 is ≥ 0.01 (*P-value* = 0.274932), the conclusion is that the sequence is random.

Note that for the 2-bit template (B = 11), if the entire sequence had too many 2-bit runs of ones, then: 1) n_5 would have been too large, 2) the test statistic would be too large, 3) the *P-value* would have been small (< 0.01) and 4) a conclusion of non-randomness would have resulted.

2.8.7 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 10^6 bits (i.e., $n \geq 10^6$). Various values of m may be selected, but for the time being, NIST recommends $m = 9$ or $m = 10$. If other values are desired, please choose these values as follows:

- $n \geq MN$.
- N should be chosen so that $N \cdot (\min p_i) > 5$.
- $l = (M-m+1)/2^m \gg 2$
- m should be chosen so that $m \gg \log_2 M$
- Choose K so that $K \gg 2l$. Note that the p_i values would need to be recalculated for values of K other than 5.

2.8.8 Example

(input) e = the binary expansion of e up to 1,000,000 bits

(input) $n = 1000000$, $B = 111111111$

(processing) $n_0 = 329$; $n_1 = 164$; $n_2 = 150$; $n_3 = 111$; $n_4 = 78$; and $n_5 = 136$

(processing) $c^2(obs) = 8.965859$

(output) $P\text{-}value = 0.110434$

(conclusion) Since the $P\text{-}value$ ³ 0.01, accept the sequence as random.

2.9 Maurer's "Universal Statistical" Test

2.9.1 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.

2.9.2 Function Call

Universal(L, Q, n), where

- L The length of each block. Note: the use of L as the block size is not consistent with the block size notation (M) used for the other tests. However, the use of L as the block size was specified in the original source of Maurer's test.
- Q The number of blocks in the initialization sequence.
- n The length of the bit string.

Additional input used by the function, but supplied by the testing code:

- ε The sequence of bits as generated by the RNG or PRNG being tested; this exists as a global structure at the time of the function call; $e = e_1, e_2, \ldots, e_n$.

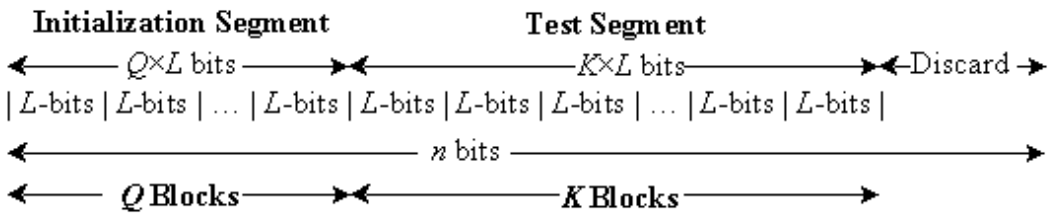
2.9.3 Test Statistic and Reference Distribution

f_n : The sum of the log_2 distances between matching L-bit templates, i.e., the sum of the number of digits in the distance between L-bit templates.

The reference distribution for the test statistic is the half-normal distribution (a one-sided variant of the normal distribution) as is also the case for the Frequency test in Section 2.1.

2.9.4 Test Description

(1) The n-bit sequence (e) is partitioned into two segments: an initialization 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 initialize the test. The remaining K blocks are the test blocks (K = ën/Lû - Q).

For example, if e = *01011010011101010111*, then $n = 20$. If $L = 2$ and $Q = 4$, then K = ën/Lû - Q = ë20/2û - 4 = 6 . The initialization segment is 0101101001; the test segment is 1101010111. The L-bit blocks are shown in the following table:

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

(2) Using the initialization 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, $T_j = i$, where j is the decimal representation of the contents of the ith L-bit block).

For the example in this section, the following table is created using the 4 initialization 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)**
Initialization	0	2	4	0

(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., $sum = sum + log_2(i - T_j)$).

For the example in this section, the table and the cumulative sum are developed as follows:

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 $sum = 1.584962501 + log_2(6-0) = 1.584962501 + 2.584962501 = 4.169925002$.
For block 7: 7 is placed in the "01" row of the table (i.e., T_1), and $sum = 4.169925002 + log_2(7-5) = 4.169925002 + 1 = 5.169925002$.
For block 8: 8 is placed in the "01" row of the table (i.e., T_1), and $sum = 5.169925002 + log_2(8-7) = 5.169925002 + 0 = 5.169925002$.
For block 9: 9 is placed in the "01" row of the table (i.e., T_1), and $sum = 5.169925002 + log_2(9-8) = 5.169925002 + 0 = 5.169925002$.
For block 10: 10 is placed 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

(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.

For the example in this section, $f_n = \frac{7.169925002}{6} = 1.1949875.$

(5) Compute *P-value* = ***erfc***$\left(\left|\frac{f_n - expectedValue(L)}{\sqrt{2}s}\right|\right)$, where ***erfc*** is defined in Section 5.5.3.3, and *expectedValue(L)* and s are taken from a table of precomputed values[2] (see the table below). Under an assumption of randomness, the sample mean, *expectedValue(L)*, is the theoretical expected value of the computed statistic for the given L-bit length. The theoretical standard deviation is given by $s = c\sqrt{\frac{variance(L)}{K}}$, where $c = 0.7 - \frac{0.8}{L} + \left(4 + \frac{32}{L}\right)\frac{K^{-3/L}}{15}$.

L	***expectedValue***	***variance***
6	5.2177052	2.954
7	6.1962507	3.125
8	7.1836656	3.238
9	8.1764248	3.311
10	9.1723243	3.356
11	10.170032	3.384

L	***expectedValue***	***variance***
12	11.168765	3.401
13	12.168070	3.410
14	13.167693	3.416
15	14.167488	3.419
16	15.167379	3.421

For the example in this section, *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 are not provided in the above table, since a block of length two is not recommended for testing. However, this value for L is easy to use in an example. The value for the expected value and variance for the case where L = 2, although not shown in the above table, were taken from the indicated reference[3].

2.9.5 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] From the "*Handbook of Applied Cryptography.*"
[3] From the "*Handbook of Applied Cryptography.*"

2.9.6 Conclusion and Interpretation of Test Results

Since the *P-value* obtained in step 5 of Section 2.9.4 is ≥ 0.01 (P-value = 0.767189), the conclusion is that the sequence is random.

Theoretical expected values for j have been computed as shown in the table in step (5) of Section 2.9.4. If f_n differs significantly from *expectedValue(L)*, then the sequence is significantly compressible.

2.9.7 Input Size Recommendations

This test requires a long sequence of bits (n ³ $(Q + K)L$) which are divided into two segments of L-bit blocks, where L should be chosen so that 6 £ L £ 16. The first segment consists of Q initialization blocks, where Q should be chosen so that $Q = 10 \cdot 2^L$. The second segment consists of K test blocks, where K = *én/Lù* - Q » $1000 \cdot 2^L$. The values of L, Q and n should be chosen as follows:

n	***L***	***Q = 10 · 2^L***
³ *387,840*	*6*	*640*
³ *904,960*	*7*	*1280*
³ *2,068,480*	*8*	*2560*
³ *4,654,080*	*9*	*5120*
³ *1,342,400*	*10*	*10240*
³ *22,753,280*	*11*	*20480*
³ *49,643,520*	*12*	*40960*
³ *107,560,960*	*13*	*81920*
³ *231,669,760*	*14*	*163840*
³ *496,435,200*	*15*	*327680*
³ *1,059,061,760*	*16*	*655360*

2.9.8 Example

(input) e = A binary string constructed using G-SHA-1[4]

(input) n = 1048576, L = 7, Q = 1280

(note) Note: 4 bits are discarded.

(processing) $c = 0.591311$, $s = 0.002703$, $K = 148516$, $sum = 919924.038020$

[4] Defined in FIPS 186-2.

(processing) $f_n = 6.194107$, $expectedValue = 6.196251$, $s = 3.125$

(output) *P-value* = 0.427733

(conclusion) Since *P-value* ³ *0.01*, accept the sequence as random.

2.10 Lempel-Ziv Compression Test

2.10.1 Test Purpose

The focus of this test is the number of cumulatively distinct patterns (words) in the sequence. The purpose of the test is to determine how far the tested sequence can be compressed. The sequence is considered to be non-random if it can be significantly compressed. A random sequence will have a characteristic number of distinct patterns.

2.10.2 Function Call

LempelZivCompression(n), where:

n The length of the bit string.

Additional input used by the function, but supplied by the testing code:

e The sequence of bits as generated by the RNG or PRNG being tested; this exists as a global structure at the time of the function call; $e = e_1, e_2, \ldots, e_n$.

2.10.3 Test Statistic and Reference Distribution

W_{obs}: The number of disjoint and cumulatively distinct words in the sequence.

The reference distribution for the test statistic is the normal distribution.

2.10.4 Test Description

(1) Parse the sequence into consecutive, disjoint and distinct words that will form a "dictionary" of words in the sequence. This is accomplished by creating substrings from consecutive bits of the sequence until a substring is created that has not been found previously in the sequence. The resulting substring is a new word in the dictionary.

Let W_{obs} = the number of cumulatively distinct words.

For example, if $e = 010110010$, then the examination proceeds as follows:

Bit Position	Bit	New Word?	The Word is:
1	0	Yes	0 (Bit 1)
2	1	Yes	1 (Bit 2)
3	0	No	
4	1	Yes	01 (Bits 3-4)
5	1	No	
6	0	Yes	10 (Bits (5-6)
7	0	No	
8	1	No	
9	0	Yes	010 (Bits 7-9)

There are five words in the "dictionary": 0, 1, 01, 10, 010. Hence, $W_{obs} = 5$.

(2) Compute *P-value* = ½ ***erfc***$\left(\frac{m - W_{obs}}{\sqrt{2s^2}}\right)$, where $m = 69586.25$ and $s = \sqrt{70.448718}$ when $n = 10^6$. For other values of n, the values of m and s would need to be calculated. Note that since no known theory is available to determine the exact values of m and s, these values were computed (under an assumption of randomness) using SHA-1. The Blum-Blum-Shub generator will give similar results for m and s^2.

Because the example in this section is much shorter than the recommended length, the values for m and s^2 are not valid. Instead, suppose that the test was conducted on a sequence of a million bits, and the value $W_{obs} = 69600$ was obtained, then

P-value = ½ ***erfc***$\left(\frac{69586.25 - 69600}{\sqrt{2 \bullet 70.448718}}\right) = 0.949310.$

2.10.5 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.

2.10.6 Conclusion and Interpretation of Test Results

Since the *P-value* obtained in step 2 of Section 2.10.4 is ≥ 0.01 (*P-value* = 0.949310), the conclusion is that the sequence is random.

Note that for $n = 106$, if W_{obs} had fallen below 69,561, then the conclusion would have been that the sequence is significantly compressible and, therefore, not random.

2.10.7 Input Size Recommendations

It is recommended that each sequence to be tested consist of a minimum of 1,000,000 bits (i.e., $n \geq 10^6$).

2.10.8 Example

(input) e = the first 1,000,000 digits in the binary expansion of e

(input) n = 1,000,000

(processing) $W_{obs} = 69559$

(output) $P\text{-}value = 0.000584$

(conclusion) Since $P\text{-}value < 0.01$, reject the sequence as being random.

2.11 Linear Complexity Test

2.11.1 Test Purpose

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

2.11.2 Function Call

LinearComplexity(M, n), where:

M The length in bits of a block.

n The length of the bit string.

Additional input used by the function, but supplied by the testing code:

e The sequence of bits as generated by the RNG or PRNG being tested; this exists as a global structure at the time of the function call; $e = e_1, e_2, \ldots, e_n$.

K The number of degrees of freedom; $K = 6$ has been hard coded into the test.

2.11.3 Test Statistic and Reference Distribution

$c^2(obs)$: A measure of how well the observed number of occurrences of fixed length LFSRs matches the expected number of occurrences under an assumption of randomness.

The reference distribution for the test statistic is the c^2 distribution.

2.11.4 Test Description

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

(2) Using the Berlekamp-Massey algorithm[5], determine the linear complexity L_i of each of the N blocks ($i = 1,\ldots,N$). L_i is the length of the shortest linear feedback shift register sequence that generates all bits in the block i. Within any L_i-bit sequence, some combination of the bits, when added together modulo 2, produces the next bit in the sequence (bit $L_i + 1$).

For example, if $M = 13$ and the block to be tested is *1101011110001*, then $L_i = 4$, and the sequence is produced by adding the 1st and 2nd bits within a 4-bit subsequence 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

For this block, the trial feedback algorithm works. If this were not the case, other feedback algorithms would be attempted for the block (e.g., adding bits 1 and 3 to produce bit 5, or adding bits 1, 2 and 3 to produce bit 6, etc.).

(3) Under an assumption of randomness, calculate the theoretical mean m:

$$m = \frac{M}{2} + \frac{\left(9 + (-1)^{M+1}\right)}{36} - \frac{\left(M/3 + 2/9\right)}{2^M}.$$

[5] Defined in *The Handbook of Applied Cryptography*; A. Menezes, P. Van Oorschot and S. Vanstone; CRC Press, 1997.

For the example in this section, $m = \frac{13}{2} + \frac{\left(9 + (-1)^{13+1}\right)}{36} - \frac{\left(13/3 + 2/9\right)}{2^{13}}$ = 6.777222.

(4) For each substring, calculate a value of T_i, where $T_i = (-1)^M \bullet (L_i - m) + 2/9$.

For the example, $T_i = (-1)^{13}(4 - 6.777222) + 2/9 =$ 2.999444.

(5) Record the T_i values in $v_0, \ldots, v_6$ as follows:

If:	T_i £ -2.5	Increment v_o by one
	$-2.5 < T_i$ £ -1.5	Increment v_1 by one
	$-1.5 < T_i$ £ -0.5	Increment v_2 by one
	$-0.5 < T_i$ £ 0.5	Increment v_3 by one
	$0.5 < T_i$ £ 1.5	Increment v_4 by one
	$1.5 < T_i$ £ 2.5	Increment v_5 by one
	$T_i > 2.5$	Increment v_6 by one

(6) Compute $c^2(obs) = \sum_{i=0}^{K} \frac{(v_i - Np_i)^2}{Np_i}$, where $p_0 = 0.01047$, $p_1 = 0.03125$, $p_2 = 0.125$, $p_3 = 0.5$, $p_4 = 0.25$, $p_5 = 0.0625$, $p_6 = 0.02078$ are the probabilities computed by the equations in Section 3.11.

(7) Compute *P-value* = **igamc** $\left(\frac{K}{2}, \frac{c^2(obs)}{2}\right)$.

2.11.5 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.

2.11.6 Conclusion and Interpretation of Test Results

Since the *P-value* obtained in step 7 of Section 2.10.4 is ≥ 0.01 (P-value = 0.949310), the conclusion is that the sequence is random.

Note that if the *P-value* were < *0.01*, this would have indicated that the observed frequency counts of T_i stored in the n_I bins varied from the expected values; it is expected that the distribution of the frequency of the T_i (in the n_I bins) should be proportional to the computed p_i as shown in step (6) of Section 2.11.5.

2.11.7 Input Size recommendations

Choose n ³ 10^6. The value of M must be in the range 500£ M £ 5000, and N ³ 200 for the c^2 result to be valid (see Section 3.11 for a discussion).

2.11.8 Example

(input) e = "the first 1,000,000 binary digits in the expansion of e"

(input) $n = 1000000 = 10^6$, $M = 1000$

(processing) $v_0 = 11$; $v_1 = 31$; $v_2 = 116$; $v_3 = 501$; $v_4 = 258$; $v_5 = 57$; $v_6 = 26$

(processing) $c^2(obs) = 2.700348$

(output) $P\text{-}value = 0.845406$

(conclusion) Since the $P\text{-}value$ ³ 0.01, accept the sequence as random.

2.12 Serial Test

2.12.1 Test Purpose

The focus of this test is the frequency of all possible overlapping m-bit patterns across the entire sequence. The purpose of this test is to determine whether the number of occurrences of the 2^m m-bit overlapping patterns is 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 of Section 2.1.

2.12.2 Function Call

Serial*(m,n)*, where:

- m The length in bits of each block.
- n The length in bits of the bit string.

Additional input used by the function, but supplied by the testing code:

- ε The sequence of bits as generated by the RNG or PRNG being tested; this exists as a global structure at the time of the function call; $e = e_1, e_2, \ldots, e_n$.

2.12.3 Test Statistics and Reference Distribution

$\tilde{N}y^2_m(obs)$ and $\tilde{N}^2 y^2_m(obs)$: A measure of how well the observed frequencies of *m*-bit patterns match the expected frequencies of the *m*-bit patterns.

The reference distribution for the test statistic is the c^2 distribution.

2.12.4 Test Description

(1) Form an augmented sequence *e′*: Extend the sequence by appending the first *m-1* bits to the end of the sequence for distinct values of *n*.

For example, given $n = 10$ and $e = 0011011101$. If $m = 3$, then $e' = 001101110100$. If $m = 2$, then $e' = 00110111010$. If $m = 1$, then e' = the original sequence 0011011101.

(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 $v_{i_1 \ldots i_m}$ denote the frequency of the *m*-bit pattern $i_1 \ldots i_m$; let $v_{i_1 \ldots i_{m-1}}$ denote the frequency of the (*m-1*)-bit pattern $i_1 \ldots i_{m-1}$; and let $v_{i_1 \ldots i_{m-2}}$ denote the frequency of the (*m-2*)-bit pattern $i_1 \ldots i_{m-2}$.

For the example in this section, when $m = 3$, then $(m-1) = 2$, and $(m-2) = 1$. The frequency of all 3-bit blocks is: $v_{000} = 0$, $v_{001} = 1$, $v_{010} = 1$, $v_{011} = 2$, $v_{100} = 1$, $v_{101} = 2$, $v_{110} = 2$, $v_{111} = 0$. The frequency of all possible (*m-1*)-bit blocks is: $v_{00} = 1$, $v_{01} = 3$, $v_{10} = 3$, $v_{11} = 3$. The frequency of all (*m-2*)-bit blocks is: $v_0 = 4$, $v_1 = 6$.

(3) Compute:

$$y^2_m = \frac{2^m}{n} \sum_{i_1 \ldots i_m} \left(v_{i_1 \ldots i_m} - \frac{n}{2^m} \right)^2 = \frac{2^m}{n} \sum_{i_1 \ldots i_m} v^2_{i_1 \ldots i_m} - n$$

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

$$y^2_{m-2} = \frac{2^{m-2}}{n} \sum_{i_1 \ldots i_{m-2}} \left(v_{i_1 \ldots i_{m-2}} - \frac{n}{2^{m-2}} \right)^2 = \frac{2^{m-2}}{n} \sum_{i_1 \ldots i_{m-2}} v^2_{i_1 \ldots i_{m-2}} - n$$

For the example in this section,

$$y^2_3 = \frac{2^3}{10}(0+1+1+4+1+4+4+1) - 10 = 12.8 - 10 = 2.8$$

$$y^2_2 = \frac{2^2}{10}(1+9+9+9) - 10 = 11.2 - 10 = 1.2$$

$$y^2_1 = \frac{2}{10}(16+36) - 10 = 10.4 - 10 = 0.4$$

(4) Compute: $\nabla y_m^2 = y_m^2 - y_{m-1}^2$, and

$$\nabla^2 y_m^2 = y_m^2 - 2y_{m-1}^2 + y_{m-2}^2 .$$

For the example in this section,

$$\nabla y_m^2 = y_m^2 - y_{m-1}^2 = \Psi_3^2 - \Psi_2^2 = 2.8 - 1.2 = 1.6$$

$$\nabla^2 y_m^2 = y_m^2 - 2y_{m-1}^2 + y_{m-2}^2 = \Psi_3^2 - 2\Psi_2^2 + \Psi_1^2 = 2.8 - 2(1.2) + 0.4 = 0.8$$

(5) Compute: *P-value1* = **igamc** $\left(2^{m-2}, \nabla y_m^2\right)$ and

P-value2 = **igamc** $\left(2^{m-3}, \nabla^2 y_m^2\right)$.

For the example in this section,

P-value1 = **igamc** $(2, 1.6) = 0.808792$

P-value2 = **igamc** $(1, 0.8) = 0.670320$.

2.12.5 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.

2.12.6 Conclusion and Interpretation of Test Results

Since the *P-value* obtained in step 5 of Section 2.12.4 is ≥ 0.01 (P-value1 = 0.808792 and P-value2 = 0.670320), the conclusion is that the sequence is random.

Note that if $\tilde{N}^2 y^2_m$ or $\tilde{N} y^2_m$ had been large, then non-uniformity of the *m*-bit blocks is implied.

2.12.7 Input Size Recommendations

Choose *m* and *n* such that $m < \ddot{e}\log_2 n\hat{u} - 2$.

2.12.8 Example

(input) *e* = 1,000,000 bits from the binary expansion of *e*

(input) $m = 2;\ n = 1000000 = 10^6$

(processing) *#0s = 499971; #1s = 500029*

#00s = 250116; #01s = #10s = 249855; #11s = 250174

(processing) $y^2_2 = 0.343128;\ y^2_1 = 0.003364;\ y^2_0 = 0.000000$

(processing) $\tilde{N}y^2_2 = 0.339764;\ \tilde{N}^2 y^2_2 = 0.336400$

(output) $P\text{-}value_1 = 0.843764;\ P\text{-}value_2 = 0.561915$

(conclusion) Since both *P-value1* and *P-value2* were $^3\ 0.01$, accept the sequences as random for both tests.

2.13 Approximate Entropy Test

2.13.1 Test Purpose

As with the Serial test of Section 2.12, the focus of this test is the frequency of all possible overlapping m-bit patterns across the entire sequence. The purpose of the test is to compare the frequency of overlapping blocks of two consecutive/adjacent lengths (*m* and *m+1*) against the expected result for a random sequence.

2.13.2 Function Call

ApproximateEntropy(*m*,*n*), where:

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 of the entire bit sequence.

Additional input used by the function, but supplied by the testing code:

ε The sequence of bits as generated by the RNG or PRNG being tested; this exists as a global structure at the time of the function call; $e = e_1, e_2, \ldots, e_n$.

2.13.3 Test Statistic and Reference Distribution

$c^2(obs)$: A measure of how well the observed value of *ApEn*(*m*) (see step 6 in Section 2.13.4) matches the expected value.

The reference distribution for the test statistic is the c^2 distribution.

2.13.4 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.

For example, if $e = 0100110101$ and $m = 3$, then $n = 10$. Append the 0 and 1 at the beginning of the sequence to the end of the sequence. The sequence to be tested becomes *010011010101*. (Note: This is done for each value of m.)

(2) A frequency count is made of the n overlapping blocks (e.g., if a block containing e_j to e_{j+m-1} is examined at time j, then the block containing e_{j+1} to e_{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.

For the example in this section, 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*, #011 = *1*, #100 = *1*, #101 = *3*, #110 = *1*, #111 = *0*

(3) Compute $C_i^m = \frac{\# i}{n}$ for each value of i.
For example in this section, $C^3_{000} = 0$, $C^3_{001} = 0.1$, $C^3_{010} = 0.3$, $C^3_{011} = 0.1$, $C^3_{100} = 0.1$, $C^3_{101} = 0.3$, $C^3_{110} = 0.1$, $C^3_{111} = 0$.

(4) Compute $j^{(m)} = \sum_{i=0}^{2^m-1} p_i \log p_i$, where $p_i = C^3_j$, and $j = \log_2 i$.

For the example in this section, $j^{(3)} = 0(\log 0) + 0.1(\log 0.1) + 0.3(\log 0.3) + 0.1(\log 0.1) + 0.1(\log 0.1) + 0.3(\log 0.3) + 0.1(\log 0.1) + 0(\log 0) = -1.64341772$.

(5) Repeat steps 1-4, replacing m by $m+1$.

Step 1: For the example in this section, m is now 4, the sequence to be tested becomes *0100110101010*.

Step 2: The overlapping blocks become 0100, 1001, 0011, 0110, 1101, 1010, 0101, 1010, 0101, 1010. The calculated values are: #0011 = 1, #0100 = 1, #0101 = 2, #0110 = 1, #1001 = 1, #1010 = 3, #1101 = 1, and all other patterns are zero.

Step 3: $C^4_{0011} = C^4_{0100} = C^4_{0110} = C^4_{1001} = C^4_{1101} = 0.1$, $C^4_{0101} = 0.2$, $C^4_{1010} = 0.3$, and all other values are zero.

Step 4: $j^{(4)} = 0 + 0 + 0 + 0.1(\log 0.01) + 0.1(\log 0.01) + 0.2(\log 0.02) + 0.1(\log 0.01) + 0 + 0 + 0.1(\log 0.01) + 0.3(\log 0.03) + 0 + 0 + 0.1(\log 0.01) + 0 + 0) = -1.83437197$.

(6) Compute the test statistic: $c^2 = 2n[\log 2 - ApEn(m)]$, where $ApEn(m) = j^{(m)} - j^{(m+1)}$.

For the example in this section,

$ApEn(3) = -1.643418 - (-1.834372) = 0.190954$

$c^2 = 2 \cdot 10(0.693147-0.190954) = 0.502193$

(7) Compute *P-value* = **igamc**$(2^{m-1}, \frac{c^2}{2})$.

For the example in this section, *P-value* = **igamc**$\left(2^2, \frac{0.502193}{2}\right)$ = 0.261961.

2.13.5 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.

2.13.6 Conclusion and Interpretation of Test Results

Since the *P-value* obtained in step 7 of Section 2.13.4 is ≥ 0.01 (P-value = 0.261961), the conclusion is that the sequence is random.

Note that small values of *ApEn(m)* would imply strong regularity (see step 6 of Section 2.13.4). Large values would imply substantial fluctuation or irregularity.

2.13.7 Input Size Recommendations

Choose *m* and *n* such that $m < \ë\log_2 n\hat{u} - 2$.

2.13.8 Example

(input) *e* = 1100100100001111110110101010001000100001011010001100000100011010011000100110001100110001010001011100 0

(input) *m* = 2; *n* = 100

(processing) *ApEn(m)* = 0.665393

(processing) $c^2(obs) = 5.550792$

(output) *P-value* = 0.235301

(conclusion) Since *P-value* ³ *0.01*, accept the sequence as random.

2.14 Cumulative Sums (Cusum) Test

2.14.1 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 behavior 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.

2.14.2 Function Call

CumulativeSums(*mode*,*n*), where:

n The length of the bit string.

Additional input for the function, but supplied by the testing code:

ε The sequence of bits as generated by the RNG or PRNG being tested; this exists as a global structure at the time of the function call; $e = e_1, e_2, \ldots, e_n$.

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

2.14.3 Test Statistic and Reference Distribution

z: The largest excursion from the origin of the cumulative sums in the corresponding (-1, +1) sequence.

The reference distribution for the test statistic is the normal distribution.

2.14.4 Test Description

(1) Form a normalized sequence: The zeros and ones of the input sequence (*e*) are converted to values X_i of –1 and +1 using $X_i = 2e_i - 1$.

For example, if $e = 1011010111$, then $X = 1, (-1), 1, 1, (-1), 1, (-1), 1, 1, 1$.

(2) Compute 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)
$S_1 = X_1$ $S_2 = X_1 + X_2$ $S_3 = X_1 + X_2 + X_3$ . . $S_k = X_1 + X_2 + X_3 + \ldots + X_k$ . . $S_n = X_1 + X_2 + X_3 + \ldots + X_k + \ldots + X_n$	$S_1 = X_n$ $S_2 = X_n + X_{n-1}$ $S_3 = X_n + X_{n-1} + X_{n-2}$ . . $S_k = X_n + X_{n-1} + X_{n-2} + \ldots + X_{n-k+1}$ . . $S_n = X_n + X_{n-1} + X_{n-2} + \ldots + X_{k-1} + \ldots + X_1$

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.

For the example in this section, when *mode = 0* and *X = 1, (-1), 1, 1, (-1), 1, (-1), 1, 1, 1*, then:

$S_1 = 1$
$S_2 = 1 + (-1) = 0$
$S_3 = 1 + (-1) + 1 = 1$
$S_4 = 1 + (-1) + 1 + 1 = 2$
$S_5 = 1 + (-1) + 1 + 1 + (-1) = 1$
$S_6 = 1 + (-1) + 1 + 1 + (-1) + 1 = 2$
$S_7 = 1 + (-1) + 1 + 1 + (-1) + 1 + (-1) = 1$
$S_8 = 1 + (-1) + 1 + 1 + (-1) + 1 + (-1) + 1 = 2$
$S_9 = 1 + (-1) + 1 + 1 + (-1) + 1 + (-1) + 1 + 1 = 3$
$S_{10} = 1 + (-1) + 1 + 1 + (-1) + 1 + (-1) + 1 + 1 + 1 = 4$

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

For the example in this section, the largest value of S_k is 4, so $z = 4$.

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

where F is the Standard Normal Cumulative Probability Distribution Function as defined in Section 5.5.3.3.

For the example in this section, *P-value* = 0.4116588.

2.14.5 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.

2.14.6 Conclusion and Interpretation of Test Results

Since the *P-value* obtained in step 4 of Section 2.14.4 is ≥ 0.01 (*P-value* = 0.411658), the conclusion is that the sequence is random.

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

2.14.7 Input Size Recommendations

It is recommended that each sequence to be tested consist of a minimum of 100 bits (i.e., $n \geq 100$).

2.14.8 Example

(input) e = 11001001000011111101101010100010001000010110100011
00001000110100110001001100011001100010100010111000

(input) n = 100

(input) *mode* = 0 (forward) || *mode* = 1 (reverse)

(processing) z = 1.6 (forward) || z = 1.9 (reverse)

(output) *P-value* = 0.219194 (forward) || *P-value* = 0.114866 (reverse)

(conclusion) Since *P-value* > *0.01*, accept the sequence as random.

2.15 Random Excursions Test

2.15.1 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 of 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 conclusions), one test and conclusion for each of the states: -4, -3, -2, -1 and +1, +2, +3, +4.

2.15.2 Function Call

RandomExcursions(n), where:

n The length of the bit string.

Additional input used by the function, but supplied by the testing code:

ε The sequence of bits as generated by the RNG or PRNG being tested; this exists as a global structure at the time of the function call; $e = e_1, e_2, \ldots, e_n$.

2.15.3 Test Statistic and Reference Distribution

$c^2(obs)$: For a given state x, a measure of how well the observed number of state visits within a cycle match the expected number of state visits within a cycle, under an assumption of randomness.

The reference distribution for the test statistic is the c^2 distribution.

2.15.4 Test Description

(1) Form a normalized (-1, +1) sequence X: The zeros and ones of the input sequence (e) are changed to values of –1 and +1 via $X_i = 2e_i - 1$.

For example, if e = 0110110101, then $n = 10$ and $X = -1, 1, 1, -1, 1, 1, -1, 1, -1, 1$.

(2) Compute the partial sums S_i of successively larger subsequences, each starting with X_1. Form the set $S = \{S_i\}$.

$$S_1 = X_1$$
$$S_2 = X_1 + X_2$$
$$S_3 = X_1 + X_2 + X_3$$
.
.

$S_k = X_1 + X_2 + X_3 + \ldots + X_k$

.

.

$S_n = X_1 + X_2 + X_3 + \ldots + X_k + \ldots + X_n$

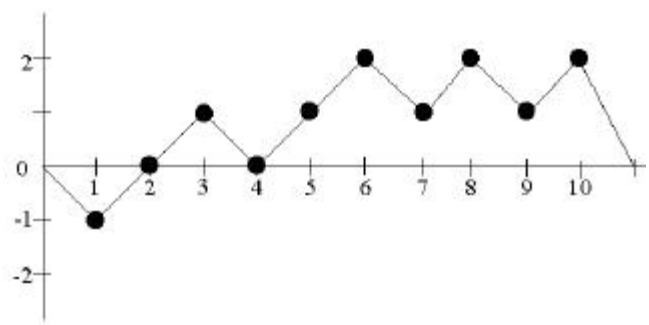
For the example in this section,

$S_1 = -1$	$S_6 = 2$
$S_2 = 0$	$S_7 = 1$
$S_3 = 1$	$S_8 = 2$
$S_4 = 0$	$S_9 = 1$
$S_5 = 1$	$S_{10} = 2$

The set S = {-1, 0, 1, 0, 1, 2, 1, 2, 1, 2}.

(3) Form a new sequence S' by attaching zeros before and after the set S. That is, $S' = 0, s_1, s_2, \ldots, s_n, 0$.

For the example in this section, $S' = 0, -1, 0, 1, 0, 1, 2, 1, 2, 1, 2, 0$. The resulting random walk is shown below.



Example Random Walk (S')

(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 of S' 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[6].

For the example in this section, if S' = {0, –1, 0 1, 0, 1, 2, 1, 2, 1, 2, 0}, then J = 3 (there are zeros in positions 3, 5 and 12 of S'). The zero crossings are easily observed in the above plot. Since J = 3, there are 3 cycles, consisting of {0, -1, 0}, {0, 1, 0} and {0, 1, 2, 1, 2, 1, 2, 0}.

[6] J times the minimum of the probabilities found in the table in Section 3.15 must be ≥ 5 in order to satisfy the empirical rule for Chi-square computations.

(5) For each cycle and for each non-zero state value x having values -4 £ x £ -1 and 1 £ x £ 4, compute the frequency of each x within each cycle.

For the example in this section, in step 3, the first cycle has one occurrence of –1, the second cycle has one occurrence of 1, and the third cycle has three occurrences each of 1 and 2. This can be visualized using the following table.

State	**Cycles**		
x	**Cycle 1 (0, -1, 0)**	**Cycle 2 (0, 1, 0)**	**Cycle 3 (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

(6) For each of the eight states of x, compute $n_k(x)$ = the total number of cycles in which state x occurs exactly k times among all cycles, for $k = 0, 1, \ldots, 5$ (for k = 5, all frequencies ≥ 5 are stored in $n_5(x)$). Note that $\sum_{k=0}^{5} n_k(x) = J$.

For the example in this section,

- $n_0(-1) = 2$ (the –1 state occurs exactly 0 times in two cycles),
  $n_1(-1) = 1$ (the –1 state occurs only once in 1 cycle), and
  $n_2(-1) = n_3(-1) = n_4(-1) = n_5(-1) = 0$ (the –1 state occurs exactly {2, 3, 4, ≥5} times in 0 cycles).

- $n_0(1) = 1$ (the 1 state occurs exactly 0 times in 1 cycle),
  $n_1(1) = 1$ (the 1 state occurs only once in 1 cycle),
  $n_3(1) = 1$ (the 1 state occurs exactly three times in 1 cycle), and
  $n_2(1) = n_4(1) = n_5(1) = 0$ (the 1 state occurs exactly {2, 4, ≥5} times in 0 cycles).

- $n_0(2) = 2$ (the 2 state occurs exactly 0 times in 2 cycles),
  $n_3(2) = 1$ (the 2 state occurs exactly three times in 1 cycle), and
  $n_1(2) = n_2(2) = n_4(2) = n_5(2) = 0$ (the 1 state occurs exactly {1, 2, 4, ≥5} times in 0 cycles).

- $n_0(-4) = 3$ (the -4 state occurs exactly 0 times in 3 cycles), and

$n_1(-4) = n_2(-4) = n_3(-4) = n_4(-4) = n_5(-4) = 0$ (the -4 state occurs exactly {1, 2, 3, 4, ≥5} times in 0 cycles).

And so on….

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	1	0	0	0	0
3	3	0	0	0	0	0
4	3	0	0	0	0	0

(7) For each of the eight states of x, compute the test statistic $c^2(obs) = \sum_{k=0}^{5} \frac{(n_k(x) - Jp_k(x))^2}{Jp_k(x)}$, where $p_k(x)$ is the probability that the state x occurs k times in a random distribution (see Section 3.15 for a table of p_k values). The values for $p_k(x)$ and their method of calculation are provided in Section 3.15. Note that eight χ^2 statistics will be produced (i.e., for $x = -4, -3, -2, -1, 1, 2, 3, 4$).

For example in this section, when $x = 1$,

$$c^2 = \frac{(1-3(0.5))^2}{3(0.5)} + \frac{(1-3(0.25))^2}{3(0.25)} + \frac{(0-3(0.125))^2}{3(0.125)} + \frac{(1-3(0.0625))^2}{3(0.0625)} + \frac{(0-3(0.0312))^2}{3(0.0312)} + \frac{(0-3(0.0312))^2}{3(0.0312)}$$

= 4.333033

(8) For each state of x, compute *P-value* = **igamc**$(5/2,\ c^2(obs)/2)$. Eight *P-values* will be produced.

For the example when $x = 1$, *P-value* = **igamc**$\left(\frac{5}{2}, \frac{4.333033}{2}\right)$ = 0.502529.

2.15.5 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.

2.15.6 Conclusion and Interpretation of Test Results

Since the *P-value* obtained in step 8 of Section 2.15.4 is ≥ 0.01 (*P-value* = 0.502529), the conclusion is that the sequence is random.

Note that if $c^2(obs)$ were too large, then the sequence would have displayed a deviation from the theoretical distribution for a given state across all cycles.

2.15.7 Input Size Recommendations

It is recommended that each sequence to be tested consist of a minimum of 1,000,000 bits (i.e., $n \geq 10^6$).

2.15.8 Example

(input) e = "the binary expansion of e up to 1,000,000 bits"

(input) $n = 1000000 = 10^6$

(processing) $J = 1490$

State=x	c^2	*P-value*	Conclusion
-4	3.835698	0.573306	Random
-3	7.318707	0.197996	Random
-2	7.861927	0.164011	Random
-1	15.692617	0.007779	Non-random
+1	2.485906	0.778616	Random
+2	5.429381	0.365752	Random
+3	2.404171	0.790853	Random
+4	2.393928	0.792378	Random

(conclusion) For seven of the states of x, the *P-value* is ≥ 0.01, and the conclusion would be that the sequence was random. However, for one state of x (x = -1), the *P-value* is < 0.01, so the conclusion would be that the sequence is non-random. When contradictions arise, further sequences should be examined to determine whether or not this behavior is typical of the generator.

2.16 Random Excursions Variant Test

2.16.1 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.

2.16.2 Function Call

RandomExcursionsVariant(n), where:

n The length of the bit string; available as a parameter during the function call.

Additional input used by the function, but supplied by the testing code:

ε The sequence of bits as generated by the RNG or PRNG being tested; this exists as a global structure at the time of the function call; $e = e_1, e_2, \ldots, e_n$.

2.16.3 Test Statistic and Reference Distribution

x: For a given state x, the total number of times that the given state is visited during the entire random walk as determined in step 4 of Section 2.13.4.

The reference distribution for the test statistic is the half normal (for large n). (Note: If x is distributed as normal, then $|x|$ is distributed as half normal.) If the sequence is random, then the test statistic will be about 0. If there are too many ones or too many zeroes, then the test statistic will be large.

2.16.4 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 $X = X_1, X_2, \ldots, X_n$, where $X_i = 2e_i - 1$.

For example, if e = 0110110101, then $n = 10$ and $X = -1, 1, 1, -1, 1, 1, -1, 1, -1, 1$.

(2) Compute partial sums S_i of successively larger subsequences, each starting with x_1. Form the set $S = \{S_i\}$.

$S_1 = X_1$
$S_2 = X_1 + X_2$
$S_3 = X_1 + X_2 + X_3$
.

.

$S_k = X_1 + X_2 + X_3 + \ldots + X_k$

.

.

$S_n = X_1 + X_2 + X_3 + \ldots + X_k + \ldots + X_n$

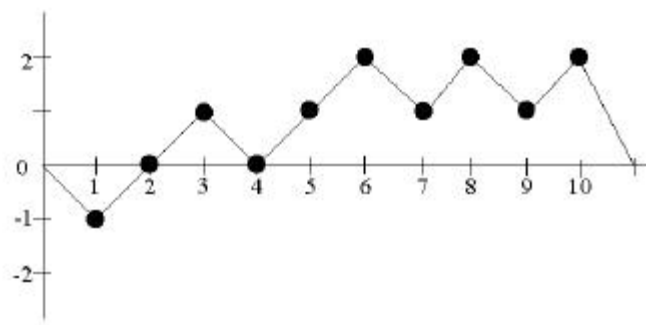
For the example in this section,

$S_1 = -1$	$S_6 = 2$
$S_2 = 0$	$S_7 = 1$
$S_3 = 1$	$S_8 = 2$
$S_4 = 0$	$S_9 = 1$
$S_5 = 1$	$S_{10} = 2$

The set S = {-1, 0, 1, 0, 1, 2, 1, 2, 1, 2}.

(3) Form a new sequence *S*' by attaching zeros before and after the set *S*. That is, $S' = 0, s_1, s_2, \ldots, s_n, 0$.

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*')

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

For the example in this section, $x(-1) = 1$, $x(1) = 4$, $x(2) = 3$, and all other $x(x) = 0$.

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

See Section 5.5.3.3 for the definition of ***erfc***.

For the example in this section, when $x = 1$, *P-value* = $\boldsymbol{erfc}\left(\frac{|4-3|}{\sqrt{2 \bullet 3(4|4|-2)}}\right) =$ 0.877371.

2.16.5 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.

2.16.6 Conclusion and Interpretation of Test Results

Since the *P-value* obtained in step 7 of Section 2.16.4 is ≥ 0.01 for the state $x = 1$(P-value = 0.877371), the conclusion is that the sequence is random.

2.16.7 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 10^6).

2.16.8 Example

(input) e = "the binary expansion of e up to 1,000,000 bits"

(input) $n = 1000000 = 10^6$

(processing) $J = 1490$

State(x)	Counts	*P-value*	Conclusion
-9	1450	0.858946	Random
-8	1435	0.794755	Random
-7	1380	0.576249	Random
-6	1366	0.493417	Random
-5	1412	0.633873	Random
-4	1475	0.917283	Random
-3	1480	0.934708	Random
-2	1468	0.816012	Random
-1	1502	0.826009	Random
+1	1409	0.137861	Random
+2	1369	0.200642	Random
+3	1396	0.441254	Random

+4	1479	0.939291	Random
+5	1599	0.505683	Random
+6	1628	0.445935	Random
+7	1619	0.512207	Random
+8	1620	0.538635	Random
+9	1610	0.593930	Random

(conclusion) Since the *P-value* ³ *0.01* for each of the eighteen states of x, accept the sequence as random.

3 TECHNICAL DESCRIPTION OF TESTS

This section contains the mathematical backgound for the tests in the NIST test suite. Each subsection corresponds to the appropriate subsection in Section 2. The relevant references for each subsection are provided at the end of that subsection.

3.1 Frequency (Monobit) Test

The most basic test is that of the null hypothesis: in a sequence of independent identically distributed Bernoulli random variables (X's or ϵ's, where X = 2ϵ-1, and so $S_n = X_1 + \ldots + X_n = 2(\epsilon_1 + \ldots + \epsilon_n) - n$), the probability of ones is $\frac{1}{2}$. By the classic De Moivre-Laplace theorem, for a sufficiently large number of trials, the distribution of the binomial sum, normalized by $\sqrt{n}$, is closely approximated by a standard normal distribution. This test makes use of that approximation to assess the closeness of the fraction of 1's to $\frac{1}{2}$. All subsequent tests are conditioned on having passed this first basic test.

The test is derived from the well-known limit Central Limit Theorem for the random walk, $S_n = X_1 + \cdots + X_n$. According to the Central Limit Theorem,

$$\lim_{n\to\infty} P\left(\frac{S_n}{\sqrt{n}} \leq z\right) = \Phi(z) \equiv \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{z} e^{-u^2/2}\, du. \tag{1}$$

This classical result serves as the basis of the simplest test for randomness. It implies that, for positive z,

$$P\left(\frac{|S_n|}{\sqrt{n}} \leq z\right) = 2\Phi(z) - 1.$$

According to the test based on the statistic $s = |S_n|/\sqrt{n}$, evaluate the observed value $|s(obs)| = |X_1 + \ldots + X_n|/\sqrt{n}$, and then calculate the corresponding $P - value$, which is $2[1 - \Phi(|s(obs)|)] = erfc(|s(obs)|/\sqrt{2})$. Here, *erfc* is the (complementary) error function

$$erfc(z) = \frac{2}{\sqrt{\pi}} \int_{z}^{\infty} e^{-u^2}\, du.$$

References for Test

[1] Kai Lai Chung, Elementary Probability Theory with Stochastic Processes. New York: Springer-Verlag, 1979 (especially pp. 210-217).

[2] Jim Pitman, Probability. New York: Springer-Verlag, 1993 (especially pp. 93-108).

3.2 Frequency Test within a Block

The test seeks to detect localized deviations from the ideal 50 % frequency of 1's by decomposing the test sequence into a number of nonoverlapping subsequences and applying a chi-square test for a homogeneous match of empirical frequencies to the ideal $\frac{1}{2}$. Small $P - values$ indicate large deviations from the equal proportion of ones and zeros in at least one of the substrings. The string of 0's and 1's (or equivalent -1's and 1's) is partitioned into a number of disjoint substrings. For each substring, the proportion of ones is computed. A chi-square statistic compares these substring proportions to the ideal $\frac{1}{2}$. The statistic is referred to a chi-squared distribution with the degrees of freedom equal to the number of substrings.

The parameters of this test are M and N, so that $n = MN$, i.e., the original string is partitioned into N substrings, each of length M. For each of these substrings, the probability of ones is estimated by the observed relative frequency of 1's, $\pi_i, i = 1, \ldots, N$. The sum

$$\chi^2(obs) = 4M \sum_{1}^{N} \left[\pi_i - \frac{1}{2}\right]^2$$

under the randomness hypothesis has the χ^2-distribution with N degrees of freedom. The reported $P - value$ is

$$\frac{\int_{\chi^2(obs)}^{\infty} e^{-u/2} u^{N/2-1}\, du}{\Gamma(N/2)\, 2^{N/2}} = \frac{\int_{\chi^2(obs)/2}^{\infty} e^{-u} u^{N/2-1}\, du}{\Gamma(N/2)}$$

$$= \mathbf{igamc}\left(\frac{N}{2}, \frac{\chi^2(obs)}{2}\right),$$

.

References for Test

[1] Nick Maclaren, "Cryptographic Pseudo-random Numbers in Simulation," Cambridge Security Workshop on Fast Software Encryption. Dec. 1993. Cambridge, U.K.: R. Anderson, pp. 185-190.

[2] Donald E. Knuth, The Art of Computer Programming. Vol 2: Seminumerical Algorithms. 3rd ed. Reading, Mass: Addison-Wesley, 1998 (especially pp. 42-47).

[3] Milton Abramowitz and Irene Stegun, Handbook of Mathematical Functions: NBS Applied Mathematics Series 55. Washington, D.C.: U.S. Government Printing Office, 1967.

3.3 Runs Test

This variant of a classic nonparametric test looks at "runs" defined as substrings of consecutive 1's and consecutive 0's, and considers whether the oscillation among such homogeneous substrings is too fast or too slow.

The specific test used here is based on the distribution of the total number of runs, V_n. For the fixed proportion $\pi = \sum_j \epsilon_j / n$ (which by the Frequency test of Section 3.1 must have been established to be close to 0.5: $|\pi - \frac{1}{2}| \leq \frac{2}{\sqrt{n}}$).

$$\lim_{n\to\infty} P\left(\frac{V_n - 2n\pi(1-\pi)}{2\sqrt{n}\pi(1-\pi)} \leq z\right) = \Phi(z). \tag{2}$$

To evaluate V_n, define for $k = 1, \ldots, n-1$, $r(k) = 0$ if $\epsilon_k = \epsilon_{k+1}$ and $r(k) = 1$ if $\epsilon_k \neq \epsilon_{k+1}$. Then $V_n = \sum_{k=1}^{n-1} r(k) + 1$. The $P - value$ reported is

$$erfc\left(\frac{|V_n(obs) - 2n\pi(1-\pi)|}{2\sqrt{2n}\pi(1-\pi)}\right).$$

Large values of $V_n(obs)$ indicate oscillation in the string of ϵ's which is too fast; small values indicate oscillation which is too slow.

References for Test

[1] Jean D. Gibbons, Nonparametric Statistical Inference, 2nd ed. New York: Marcel Dekker, 1985 (especially pp. 50-58).

[2] Anant P. Godbole and Stavros G. Papastavridis, (ed), Runs and patterns in probability: Selected papers. Dordrecht: Kluwer Academic, 1994.

3.4 Test for the Longest Run of Ones in a Block

The length of the longest consecutive subsequence (run) of ones is another characteristic that can be used for testing randomness. A string of length n, such that $n = MN$, must be partitioned into N substrings, each of length M. For the test based on the length of the longest run of ones ν_j within the j-th substring of size M, $K+1$ classes are chosen (depending on M). For each of these substrings, one evaluates the frequencies $\nu_0, \nu_1, \ldots, \nu_K$ ($\nu_0 + \nu_1 + \ldots + \nu_K = N$, i.e., the computed values of the longest run of ones within each of these substrings belonging to any of the $K+1$ chosen classes). If there are r ones and $M-r$ zeroes in the m-bit block, then the conditional probability that the longest string of ones ν in this block is less than or equal to m has the following form with $U = \min\left(M-r+1, \left[\frac{r}{m+1}\right]\right)$ (see David and Barton (1962)):

$$P(\nu \leq m|r) = \frac{1}{\binom{M}{r}} \sum_{j=0}^{U} (-1)^j \binom{M-r+1}{j} \binom{M-j(m+1)}{M-r},$$

so that

$$P(\nu \leq m) = \sum_{r=0}^{M} \binom{M}{r} P(\nu \leq m|r) \frac{1}{2^M}. \tag{3}$$

The theoretical probabilities $\pi_0, \pi_1, \ldots, \pi_K$ of these classes are determined from (3).

The empirical frequencies $\nu_i, i = 0, \ldots, K$ are conjoined by the χ^2-statistic

$$\chi^2 = \sum_{0}^{K} \frac{(\nu_i - N\pi_i)^2}{N\pi_i}.$$

which, under the randomness hypothesis, has an approximate χ^2-distribution with K degrees of freedom. The reported $P - value$ is

$$\frac{\int_{\chi^2(obs)}^{\infty} e^{-u/2} u^{K/2-1}\, du}{\Gamma\left(K/2\right) 2^{K/2}} = \mathbf{igamc}\left(\frac{K}{2}, \frac{\chi^2(obs)}{2}\right),$$

with $P(a, x)$ denoting the incomplete gamma function as expressed in Section 3.2.

The following table contains selected values of K and M with the corresponding probabilities obtained from (3). Cases $K = 3, M = 8$; $K = 5, M = 128$; and $K = 6, M = 10000$ are currently embedded in the test suite code.

$K = 3, M = 8$

classes	$\{\nu \leq 1\}$	$\{\nu = 2\}$	$\{\nu = 3\}$	$\{\nu \geq 4\}$
probabilities	$\pi_0 = 0.2148$	$\pi_1 = 0.3672$	$\pi_2 = 0.2305$	$\pi_3 = 0.1875$

$K = 5, M = 128$

classes	$\{\nu \leq 4\}$	$\{\nu = 5\}$	$\{\nu = 6\}$	$\{\nu = 7\}$
probabilities	$\pi_0 = 0.1174$	$\pi_1 = 0.2430$	$\pi_2 = 0.2493$	$\pi_3 = 0.1752$

$\{\nu = 8\}$	$\{\nu \geq 9\}$
$\pi_4 = 0.1027$	$\pi_5 = 0.1124$

$K = 5, M = 512$

classes	$\{\nu \leq 6\}$	$\{\nu = 7\}$	$\{\nu = 8\}$	$\{\nu = 9\}$
probabilities	$\pi_0 = 0.1170$	$\pi_1 = 0.2460$	$\pi_2 = 0.2523$	$\pi_3 = 0.1755$

$\{\nu = 10\}$	$\{\nu \geq 11\}$
$\pi_4 = 0.1015$	$\pi_5 = 0.1077$

$K = 5, M = 1000$

classes	$\{\nu \leq 7\}$	$\{\nu = 8\}$	$\{\nu = 9\}$	$\{\nu = 10\}$
probabilities	$\pi_0 = 0.1307$	$\pi_1 = 0.2437$	$\pi_2 = 0.2452$	$\pi_3 = 0.1714$

$\{\nu = 11\}$	$\{\nu \geq 12\}$
$\pi_4 = 0.1002$	$\pi_5 = 0.1088$

$K = 6, M = 10000$

classes	$\{\nu \leq 10\}$	$\{\nu = 11\}$	$\{\nu = 12\}$	$\{\nu = 13\}$
probabilities	$\pi_0 = 0.0882$	$\pi_1 = 0.2092$	$\pi_2 = 0.2483$	$\pi_3 = 0.1933$

$\{\nu = 14\}$	$\{\nu = 15\}$	$\{\nu \geq 16\}$
$\pi_4 = 0.1208$	$\pi_5 = 0.0675$	$\pi_6 = 0.0727$

Large values of χ^2 indicate that the sequence has clusters of ones; the generation of "random" sequences by humans tends to lead to small values of ν_n (see Revesz, 1990, p. 55).

References for Test

[1] F. N. David and D. E. Barton, Combinatorial Chance. New York: Hafner Publishing Co., 1962, p. 230.

[2] Anant P. Godbole and Stavros G. Papastavridis (ed), Runs and Patterns in Probability: Selected Papers. Dordrecht: Kluwer Academic, 1994.

[3] Pal Revesz, Random Walk in Random and Non-Random Environments. Singapore: World Scientific, 1990.

3.5 Binary Matrix Rank Test

Another approach to testing for randomness is to check for linear dependence among fixed-length substrings of the original sequence: construct matrices of successive zeroes and ones from the sequence, and check for linear dependence among the rows or columns of the constructed matrices. The deviation of the rank - or rank deficiency - of the matrices from a theoretically expected value gives the statistic of interest.

This test is a specification of one of the tests coming from the DIEHARD

[1] battery of tests. It is based on the result of Kovalenko (1972) and also formulated in Marsaglia and Tsay (1985). The result states that the rank R of the $M \times Q$ random binary matrix takes values $r = 0, 1, 2, \ldots, m$ where $m \equiv min(M, Q)$ with probabilities

$$p_r = 2^{r(Q+M-r)-MQ} \prod_{i=0}^{r-1} \frac{(1-2^{i-Q})(1-2^{i-M})}{1-2^{i-r}}.$$

The probability values are fixed in the test suite code for $M = Q = 32$. The number M is then a parameter of this test, so that ideally $n = M^2 N$, where N is the new "sample size." In practice, values for M and N are chosen so that the discarded part of the string, $n - NM^2$, is fairly small.

The rationale for this choice is that

$$p_M \approx \prod_{j=1}^{\infty} \left[1 - \frac{1}{2^j}\right] = 0.2888..,$$

$$p_{M-1} \approx 2p_M \approx 0.5776..,$$

$$p_{M-2} \approx \frac{4p_M}{9} \approx 0.1284..$$

and all other probabilities are very small (≤ 0.005) when $M \geq 10$.

For the N square matrices obtained, their ranks $R_\ell, \ell = 1, \ldots, N$ are evaluated, and the frequencies F_M, F_{M-1} and $N - F_M - F_{M-1}$ of the values M, $M-1$ and of ranks not exceeding $M-2$ are determined:

$$F_M = \#\{R_\ell = M\},$$

$$F_{M-1} = \#\{R_\ell = M-1\}.$$

To apply the χ^2-test, use the classical statistic

$$\chi^2 = \frac{(F_M - 0.2888N)^2}{0.2888N} + \frac{(F_{M-1} - 0.5776N)^2}{0.5776N}$$
$$+ \frac{(N - F_M - F_{M-1} - 0.1336N)^2}{0.13336N},$$

which, under the null (randomness) hypothesis, has an approximate χ^2-distribution with 2 degrees of freedom. The reported $P{-}value$ is $\exp\{-\chi^2(obs)/2\}$.

Interpretation of this test: large values of $\chi^2(obs)$ indicate that the deviation of the rank distribution from that corresponding to a random sequence is significant. For example, pseudo random matrices produced by a shift-register generator formed by less than M successive vectors systematically have rank $R_\ell \equiv M$, while for truly random data, the proportion of such occurrences should be only about 0.29.

References for Test

[1] George Marsaglia, DIEHARD: a battery of tests of randomness. http://stat.fsu.edu/~geo/diehard.html.

[2] I. N. Kovalenko (1972), "Distribution of the linear rank of a random matrix," Theory of Probability and its Applications. 17, pp. 342-346.

[3] G. Marsaglia and L. H. Tsay (1985), "Matrices and the structure of random number sequences," Linear Algebra and its Applications. Vol. 67, pp. 147-156.

3.6 Discrete Fourier Transform (Spectral) Test

The test described here is based on the discrete Fourier transform. It is a member of a class of procedures known as spectral methods. The Fourier test detects periodic features in the bit series that would indicate a deviation from the assumption of randomness.

Let x_k be the kth bit, where $k = 1,...,n$. Assume that the bits are coded -1 and $+1$. Let

$$f_j = \sum_{k=1}^{n} x_k \exp\left(2\pi_i(k-1)j/n\right),$$

where $\exp\left(2\pi ikj/n\right) = \cos\left(2\pi kj/n\right) + i\sin\left(2\pi kj/n\right)$, $j = 0, \ldots, n-1$, and $i \equiv \sqrt{-1}$. Because of the symmetry of the real to complex-value transform, only the values from 0 to $(n/2-1)$ are considered. Let mod_j be the modulus of the complex number f_j. Under the assumption of the randomness of the series x_i, a confidence interval can be placed on the values of mod_j. More specifically, 95 percent of the values of mod_j should be less than $h = \sqrt{3n}$.

A $P - value$ based on this threshold comes from the binomial distribution. Let N_1 be the number of peaks less than h. Only the first $n/2$ peaks are considered. Let $N_0 = .95N/2$ and $d = (N_1 - N_0)/\sqrt{n(.95)(.05)/2}$. The $P - value$ is

$$2(1 - \phi(|d|)) = \text{erfc}\left(\frac{|d|}{\sqrt{2}}\right)$$

where $\Phi(x)$ is the cumulative probability function of the standard normal distribution.

Other $P - values$ based on the series f_j or mod_j that are sensitive to departures from randomness are possible. However, the primary value of the transform comes from a plot of the series mod_j. In the accompanying figure, the top plot shows the series of mod_j for 4096 bits generated from a satisfactory generator. The line through the plot is the 95 % confidence boundary. The $P - value$ for this series is 0.8077. The bottom plot shows a corresponding plot for a generator that produces bits that are statistically dependent in a periodic pattern. In the bottom plot, significantly greater than 5 % of the magnitudes are beyond the confidence boundary. In addition, there is a clear structure in the magnitudes that is not present in the top plot. The $P - value$ for this series is 0.0001.

References for Test

[1] R. N. Bracewell, The Fourier Transform and Its Applications. New York: McGraw-Hill, 1986.

3.7 Non-overlapping Template Matching Test

This test rejects sequences exhibiting too many or too few occurrences of a given aperiodic pattern.

Let $B = (\epsilon_1^0, \cdots, \epsilon_m^0)$ be a given word (template or pattern, i.e., a fixed sequence of zeros and ones) of length m. This pattern is to be chosen as if it were a parameter of the test. We consider a test based on patterns for fixed length m. A table of selected aperiodic words out of such patterns for $m = 2, \ldots, 8$ is provided at the end of this section.

The set of periods of B

$$\mathcal{B} = \{j, 1 \leq j \leq m-1, \epsilon_{j+k}^0 = \epsilon_k^0, k = 1, \ldots, m-j\},$$

plays an important role. For example, when B corresponds to a run of m ones, $\mathcal{B} = \{1, \ldots, m-1\}$. For the B above, $\mathcal{B} = \emptyset$, and B is an aperiodic pattern (i.e., it cannot be written as $CC \ldots CC'$ for a pattern C shorter than B with C' denoting a prefix of C). In this situation, occurrences of B in the string are non-overlapping.

In general, let $W = W(m, M)$ be the number of occurrences of the given pattern B in the string. Note that the statistic W is defined also for patterns B with $\mathcal{B} \neq \emptyset$. The best way to calculate W is as the sum,

$$W = \sum_{i=1}^{n-m+1} I(\epsilon_{i+k-1} = \epsilon_k^0, k = 1, \cdots, m).$$

The random variables $I(\epsilon_{i+k-1} = \epsilon_k^0, k = 1, \cdots, m)$ are m-dependent, so that the Central Limit Theorem holds for W. The mean and variance of the approximating normal distribution have the following form,

$$\mu = \frac{n-m+1}{2^m},$$

$$\sigma^2 = n\left[\frac{1}{2^m} - \frac{2m-1}{2^{2m}}\right].$$

For the test suite code, M and N are chosen so that $n = MN$ and $N = 8$.

Partition the original string into N blocks of length M. Let $W_j = W_j(m, M)$ be the number of occurrences of the pattern B in the block j, for $j = 1, \ldots, N$.

Let $\mu = EW_j = (M - m + 1)2^{-m}$. Then, for large M, W_j has a normal distribution with mean μ and variance σ^2, so that the statistic

$$\chi^2(obs) = \sum_{j=1}^{N} \frac{(W_j - \mu)^2}{\sigma^2} \tag{4}$$

has an approximate χ^2-distribution with N degrees of freedom. Report the $P - value$ as $1 - \mathbf{P}\left(\frac{N}{2}, \frac{\chi^2(obs)}{2}\right)$.

The test can be interpreted as rejecting sequences exhibiting irregular occurrences of a given non-periodic pattern.

References for Test

[1] A. D. Barbour, L. Holst, and S. Janson, Poisson Approximation (1992). Oxford: Clarendon Press (especially Section 8.4 and Section 10.4).

Aperiodic Templates for small values of $2 \leq m \leq 5$

$m = 2$		$m = 3$			$m = 4$				$m = 5$				
0	1	0	0	1	0	0	0	1	0	0	0	0	1
1	0	0	1	1	0	0	1	1	0	0	0	1	1
		1	0	0	0	1	1	1	0	0	1	0	1
		1	1	0	1	0	0	0	0	1	0	1	1
					1	1	0	0	0	0	1	1	1
					1	1	1	0	0	1	1	1	1
									1	1	1	0	0
									1	1	0	1	0
									1	0	1	0	0
									1	1	0	0	0
									1	0	0	0	0
									1	1	1	1	0

Aperiodic Templates for small values of $6 \leq m \leq 8$

$m = 6$						$m = 7$							$m = 8$							
0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	0	0	0	0	0	1	1	0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0	0	0	1	0	1	0	0	0	0	0	1	0	1
0	0	0	1	1	1	0	0	0	0	1	1	1	0	0	0	0	0	1	1	1
0	0	1	0	1	1	0	0	0	1	0	0	1	0	0	0	0	1	0	0	1
0	0	1	1	0	1	0	0	0	1	0	1	1	0	0	0	0	1	0	1	1
0	0	1	1	1	1	0	0	0	1	1	0	1	0	0	0	0	1	1	0	1
0	1	0	0	1	1	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
0	1	0	1	1	1	0	0	1	0	0	1	1	0	0	0	1	0	0	1	1
0	1	1	1	1	1	0	0	1	0	1	0	1	0	0	0	1	0	1	0	1
1	0	0	0	0	0	0	0	1	0	1	1	1	0	0	0	1	0	1	1	1
1	0	1	0	0	0	0	0	1	1	0	1	1	0	0	0	1	1	0	0	1
1	0	1	1	0	0	0	0	1	1	1	0	1	0	0	0	1	1	0	1	1
1	1	0	0	0	0	0	0	1	1	1	1	1	0	0	0	1	1	1	1	1
1	1	0	0	1	0	0	1	0	0	0	1	1	0	0	1	0	0	0	1	1
1	1	0	1	0	0	0	1	0	0	1	1	1	0	0	1	0	0	1	0	1
1	1	1	0	0	0	0	1	0	1	0	1	1	0	0	1	0	0	1	1	1
1	1	1	0	1	0	0	1	0	1	1	1	1	0	0	1	0	0	1	1	1
1	1	1	1	0	0	0	1	1	0	1	1	1	0	0	1	0	1	0	1	1
1	1	1	1	1	0	0	1	1	1	1	1	1	0	0	1	0	1	1	0	1
						1	0	0	0	0	0	0	0	0	1	0	1	1	1	1
						1	0	0	1	0	0	0	0	0	1	1	0	1	0	1
						1	0	1	0	0	0	0	0	0	1	1	0	1	1	1
						1	0	1	0	1	0	0	0	0	1	1	1	0	1	1
						1	0	1	1	0	0	0	0	0	1	1	1	1	0	1
						1	0	1	1	1	0	0	0	0	1	1	1	1	1	1
						1	1	0	0	0	0	0	0	1	0	0	0	0	1	1
						1	1	0	0	0	1	0	0	1	0	0	0	1	1	1
						1	1	0	0	1	0	0	0	1	0	0	1	0	1	1
						1	1	0	1	0	0	0	0	1	0	0	1	1	1	1
						1	1	0	1	0	1	0	0	1	0	1	0	0	1	1
						1	1	0	1	1	0	0	0	1	0	1	0	1	1	1
						1	1	1	0	0	0	0	0	1	0	1	1	0	1	1
						1	1	1	0	0	1	0	0	1	0	1	1	1	1	1
						1	1	1	0	1	0	0	0	1	1	0	0	0	1	1
						1	1	1	0	1	1	0	0	1	1	0	1	1	1	1
						1	1	1	1	0	0	0	0	1	1	1	0	0	1	1
						1	1	1	1	0	1	0	0	1	1	1	1	1	1	1
						1	1	1	1	1	0	0	1	0	0	0	0	0	0	0
						1	1	1	1	1	1	0	1	0	0	1	0	0	0	0
													1	0	0	1	1	0	0	0
													1	0	1	0	0	0	0	0
													1	0	1	0	0	1	0	0
													1	0	1	0	1	0	0	0
													1	0	1	0	1	1	0	0
													1	0	1	1	0	0	0	0
													1	0	1	1	0	1	0	0
													1	0	1	1	1	0	0	0
													1	0	1	1	1	1	0	0
													1	1	0	0	0	0	0	0
													1	1	0	0	0	0	1	0
													1	1	0	0	0	1	0	0
													1	1	0	0	1	0	0	0
													1	1	0	0	1	0	1	0
													1	1	0	1	0	0	0	0
													1	1	0	1	0	0	1	0
													1	1	0	1	0	1	0	0
													1	1	0	1	1	0	0	0
													1	1	0	1	1	0	1	0
													1	1	0	1	1	1	0	0
													1	1	1	0	0	0	0	0
													1	1	1	0	0	0	1	0
													1	1	1	0	0	1	0	0
													1	1	1	0	0	1	1	0
													1	1	1	0	1	0	0	0
													1	1	1	0	1	0	1	0
													1	1	1	0	1	1	0	0
													1	1	1	1	0	0	0	0
													1	1	1	1	0	0	1	0
													1	1	1	1	0	1	0	0
													1	1	1	1	0	1	1	0
													1	1	1	1	1	0	0	0
													1	1	1	1	1	0	1	0
													1	1	1	1	1	1	0	0
													1	1	1	1	1	1	1	0

3.8 Overlapping Template Matching Test

This test rejects sequences which show too many or too few occurrences of m-runs of ones, but can be easily modified to detect irregular occurrences of any periodic pattern B.

To implement this test, parameters M and N are determined so that $n = MN$, i.e., the original string is partitioned into N blocks, each of length M.

Let $\tilde{W}_j = \tilde{W}_j(m, n)$ be the number of (possibly overlapping) runs of ones of length m in the jth block. The asymptotic distribution of $\tilde{W}_j$ is the compound Poisson distribution (the so-called Pòlya-Aeppli law, see Chrysaphinou and Papastavridis, 1988):

$$E \exp\{t\tilde{W}_j\} \to \exp\left\{\frac{\lambda(e^t - 1)}{2 - e^t}\right\} \tag{5}$$

when $(M - m + 1)2^{-m} \to \lambda > 0$ (t is a real variable).

The corresponding probabilities can be expressed in terms of the confluent hypergeometric function $\Phi =_1 F_1$. If U denotes a random variable with the compound Poisson asymptotic distribution, then for $u \geq 1$ with $\eta = \lambda/2$

$$P(U = u) = \frac{e^{-\eta}}{2^u} \sum_{\ell=1}^{u} \begin{pmatrix} u-1 \\ \ell-1 \end{pmatrix} \frac{\eta^\ell}{\ell!} = \frac{\eta e^{-2\eta}}{2^u} \Phi(u+1, 2, \eta).$$

For example,

$$P(U = 0) = e^{-\eta},$$

$$P(U = 1) = \frac{\eta}{2} e^{-\eta},$$

$$P(U = 2) = \frac{\eta e^{-\eta}}{8} [\eta + 2],$$

$$P(U = 3) = \frac{\eta e^{-\eta}}{8} \left[\frac{\eta^2}{6} + \eta + 1\right],$$

$$P(U = 4) = \frac{\eta e^{-\eta}}{16} \left[\frac{\eta^3}{24} + \frac{\eta^2}{2} + \frac{3\eta}{2} + 1\right].$$

The complement to the distribution function of this random variable has the form

$$L(u) = P(U > u) = e^{-\eta} \sum_{\ell=u+1}^{\infty} \frac{\eta^\ell}{\ell} \Delta(\ell, u)$$

with

$$\Delta(\ell, u) = \sum_{k=\ell}^{u} \frac{1}{2^k} \begin{pmatrix} k-1 \\ \ell-1 \end{pmatrix}.$$

Choose $K+1$ classes or cells for U, i.e., $\{U = 0\}$, $\{U = 1\}, \cdots, \{U = K-1\}$, $\{U \geq K\}$. The theoretical probabilities $\pi_0, \pi_1, \ldots, \pi_{K+1}$ of these cells are found from the above formulas. A reasonable choice could be $K = 5, \lambda = 2, \eta = 1$.

After $U_1, \ldots, U_N$ are found, evaluate the frequencies $\nu_0, \nu_1, \ldots, \nu_K$ of each cell, $\nu_0 + \nu_1 + \ldots + \nu_K = N$, and calculate the value of the chi-square statistic

$$\chi^2 = \sum_{0}^{K} \frac{(\nu_i - N\pi_i)^2}{N\pi_i}.$$

The expression for the $P-value$ is the same as that used in Section 3.7. The interpretation is that for very small $P-values$, the sequence shows irregular occurrences of m-runs of ones.

References for Test

[1] O. Chrysaphinou and S. Papastavridis, "A Limit Theorem on the Number of Overlapping Appearances of a Pattern in a Sequence of Independent Trials." Probability Theory and Related Fields, Vol. 79 (1988), pp. 129-143.

[2] N.J. Johnson, S. Kotz, and A. Kemp, Discrete Distributions. John Wiley, 2nd ed. New York, 1996 (especially pp. 378-379).

3.9 Maurer's "Universal Statistical" Test

This test was introduced in 1992 by Ueli Maurer of the Department of Computer Science at Princeton University. Maurer's test statistic relates closely to the *per-bit entropy* of the stream, which its author asserts is "the correct quality measure for a secret-key source in a cryptographic application." As

such, the test is claimed to measure the actual cryptographic significance of a defect because it is "related to the running time of [an] enemy's optimal key-search strategy," or the effective key size of a cipher system.

The test is not designed to detect a very specific pattern or type of statistical defect. However, the test is designed "to be able to detect any one of the very general class of statistical defects that can be modeled by an ergodic stationary source with finite memory." Because of this, Maurer claims that the test subsumes a number of the standard statistical tests.

The test is a compression-type test "based on the idea of Ziv that a universal statistical test can be based on a universal source coding algorithm. A generator should pass the test if and only if its output sequence cannot be compressed significantly." According to Maurer, the source-coding algorithm due to Lempel-Ziv "seems to be less suited for application as a statistical test" because it seems to be difficult to define a test statistic whose distribution can be determined or approximated.

The test requires a long (on the order of $10 \cdot 2^L + 1000 \cdot 2^L$ with $6 \leq L \leq 16$) sequence of bits which are divided into two stretches of L-bit blocks ($6 \leq L \leq 16$), Q ($\geq 10 \cdot 2^L$) initialization blocks and K ($\approx 1000 \cdot 2^L$) test blocks. We take $K = \text{ceiling}(n/L) - Q$ to maximize its value. The order of magnitude of Q should be specifically chosen to ensure that all possible L-bit binary patterns do in fact occur within the initialization blocks. The test is not suited for very large values of L because the initialization takes time exponential in L.

The test looks backs through the entire sequence while walking through the test segment of L-bit blocks, checking for the nearest previous exact L-bit template match and recording the distance - in number of blocks - to that previous match. The algorithm computes the $\log_2$ of all such distances for all the L-bit templates in the test segment (giving, effectively, the number of digits in the binary expansion of each distance). Then it averages over all the expansion lengths by the number of test blocks.

$$f_n = \frac{1}{K}[\sum_{i=Q+1}^{Q+K} \log_2(\#\text{indices since previous occurrence of ith template})]$$

The algorithm achieves this efficiently by subscripting a dynamic look-up table making use of the integer representation of the binary bits constituting the template blocks. A standardized version of the statistic - the standardization being prescribed by the test - is compared to an acceptable range based on a standard normal (Gaussian) density, making use of the test statistic's mean which is given by formula (16) in Maurer (1992),

$$Ef_n = 2^{-L} \sum_{i=1}^{\infty} (1 - 2^{-L})^{i-1} \log_2 i.$$

The expected value of the test statistic f_n is that of the random variable $\log_2 G$ where $G = G_L$ is a geometric random variable with the parameter $1 - 2^{-L}$.

There are several versions of approximate empirical formulas for the variance of the form

$$Var(f_n) = c(L, K) Var(\log_2 G)/K.$$

Here, $c(L, K)$ represents the factor that takes into account the dependent nature of the occurrences of templates. The latest of the approximations (Coron and Naccache (1998): not embedded in the test suite code) has the form

$$c(L, K) = 0.7 - \frac{0.8}{L} + \left(1.6 + \frac{12.8}{L}\right) K^{-4/L}.$$

However, Coron and Naccache (1998) report that "the inaccuracy due to [this approximation] can make the test 2.67 times more permissive than what is theoretically admitted." In other words, the ratio of the standard deviation of f_n obtained from the approximation above to the true standard deviation deviates considerably from one. In view of this fact and also since all approximations are based on the "admissible" assumption that $Q \to \infty$, the randomness hypothesis may be tested by verifying normality of the observed values f_n, assuming that the variance is unknown. This can be done using a t-test.

The original sequence must be partitioned into r ($r \leq 20$) substrings, on each of which the value of the universal test statistic is evaluated (for the same value of parameters K, L and Q). The sample variance is evaluated,

and the $P - value$ is

$$erfc\left(\left|\frac{f_n - E(L)}{\sqrt{var(f_n)}}\right|\right)$$

References for Test

[1] Ueli M. Maurer, "A Universal Statistical Test for Random Bit Generators," Journal of Cryptology. Vol. 5, No. 2, 1992, pp. 89-105.

[2] J-S Coron and D. Naccache, "An Accurate Evaluation of Maurer's Universal Test," Proceedings of SAC '98 (Lecture Notes in Computer Science). Berlin: Springer-Verlag, 1998.

[3] H. Gustafson, E. Dawson, L. Nielsen, W. Caelli, "A computer package for measuring the strength of encryption algorithms," Computers & Security. 13 (1994), pp. 687-697.

[4] A. J. Menezes, P. C. van Oorschot, S. A. Vanstone, Handbook of Applied Cryptography. Boca Raton: CRC Press, 1997.

[5] J. Ziv, "Compression, tests for randomness and estimating the statistical model of an individual sequence," Sequences (ed. R.M. Capocelli). Berlin: Springer-Verlag, 1990.

[6] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," IEEE Transactions on Information Theory. Vol. 23, pp. 337-343.

3.10 Lempel-Ziv Compression Test

This test compresses the candidate random sequence using the (1977) Lempel-Ziv algorithm. If the reduction is statistically significant when compared to a theoretically expected result, the sequence is declared to be non-random. To test a generator, many sequences are tested in this way. Significance probabilities are calculated for each sequence, and the hypothesis that the significance probabilities are uniformly distributed are tested, for example, by the Kolmogorov-Smirnov test.

The Lempel-Ziv test is thought to subsume the frequency, runs, other compression, and possibly spectral tests, but it may intersect the random binary matrix rank test. The test is similar to the entropy test and even more similar to Maurer's Universal Statistical test. However, the Lempel-Ziv test directly incorporates the compression heuristic that defines modern information theory.

There are several variations on the Lempel-Ziv algorithm (1977). The test used here assumes that $\{X_i\}_{i=1}^n$ is a binary sequence, and specifically proceeds as follows:

1. Parse the sequence into consecutive disjoint strings (words) so that the next word is the shortest string not yet seen.

2. Number the words consecutively in base 2.

3. Assign each word a prefix and a suffix; the prefix is the number of the previous word that matches all but the last digit; the suffix is the last digit.

Note that what drives this compression is the number of substrings in the parsing. It is possible that, for small n, the Lempel-Ziv compression is actually longer than the original representation.

Following the work of Aldous and Shields (1988), let $W(n)$ represent the number of words in the parsing of a binary random sequence of length n. They show that

$$\lim_{n\to\infty} \frac{E[W(n)]}{n/\log_2 n} = 1,$$

so that the expected compression is asymptotically well-approximated by $n/\log_2 n$, and that

$$\frac{W(n) - E[W(n)]}{\sigma[W(n)]} \Rightarrow N(0,1),$$

which implies that a central limit theorem holds for the number of words in the Lempel-Ziv compression. However, Aldous and Shields were unable to determine the value of $\sigma[W(n)]$.

That difficulty was nominally overcome by Kirschenhofer, Prodinger, and Szpankowski (1994) who prove that

$$\sigma^2[W(n)] \sim \frac{n[C + \delta(\log_2 n)]}{\log_2^3 n}$$

where $C = 0.26600$ (to five significant places) and $\delta(\cdot)$ is a slowly varying continuous function with mean zero and $|\delta(\cdot)| < 10^{-6}$.

The given sequence is parsed, and the number of words counted. It is not necessary to go through the complete Lempel-Ziv encoding, since the number of words, W, is sufficient. W is used to calculate

$$z = \frac{W - \frac{n}{\log_2 n}}{\sqrt{\frac{.266n}{\log_2^3 n}}}$$

which is then compared with a standard normal distribution. The test is preferably one-sided, since some patterned sequences actually are flagged for being too long after compression.

It is unclear whether the asymptotics are usefully accurate for values of n of the magnitude that may occur when testing random number generators. A simulation study performed using the Blum-Blum-Shub generator (1986) indicated that the asymptotics are not usefully accurate for sequences of length less than 10 million. Therefore, practitioners are urged to develop empirical estimates of the average compression length and its standard deviation to use in place of E[W(n)] and σ[W(n)], respectively. The accuracy of such empirical estimates depends upon the randomness of the generator used. The Blum-Blum-Shub generator was chosen because its randomness is provably equivalent to the hardness of mathematical factorization.

The $P - value$ is computed as

$$\frac{1}{2} erfc(\frac{\mu - W_{obs}}{\sqrt{2\sigma^2}})$$

For this test, the mean and variance were evaluated using SHA-1 for million bit sequences. The mean and variance were computed to be 69586.25

and 70.448718, respectively.

References for Test

[1] D. Aldous and P. Shields (1988). "A Diffusion Limit for a Class of Randomly-Growing Binary Trees," Probability Theory and Related Fields. 79, pp. 509-542.

[2] L. Blum, M. Blum, and M. Shub (1994), "A Simple Unpredictable Pseudo-Random Number Generator," SIAM Journal on Computing. 15, pp. 364-383.

[3] P. Kirschenhofer, H. Prodinger, and W. Szpankowski (1994), "Digital Search Trees Again Revisited: The Internal Path Length Perspective," SIAM Journal on Computing. 23, pp. 598-616.

[4] U. M. Maurer (1992), "A Universal Statistical Test for Random Bit Generators," Journal of Cryptology. 5, pp. 89-105.

[5] J. Ziv and A. Lempel (1977), "A Universal Algorithm for Sequential Data Compression," IEEE Transactions on Information Theory. 23, pp. 337-343.

3.11 Linear Complexity Test

This test uses linear complexity to test for randomness. The concept of linear complexity is related to a popular part of many keystream generators, namely, Linear Feedback Shift Registers (LFSR). Such a register of length L consists of L delay elements each having one input and one output. If the initial state of LFSR is $(\epsilon_{L-1}, \ldots, \epsilon_1, \epsilon_0)$, then the output sequence, $(\epsilon_L, \epsilon_{L+1}, \ldots)$, satisfies the following recurrent formula for $j \geq L$

$$\epsilon_j = (c_1\epsilon_{j-1} + c_2\epsilon_{j-2} + \cdots + c_L\epsilon_{j-L}) \mod 2.$$

$c_1, \ldots, c_L$ are coefficients of the connection polynomial corresponding to a given LFSR. An LFSR is said to generate a given binary sequence if this sequence is the output of the LFSR for some initial state.

For a given sequence $s^n = (\epsilon_1, \ldots, \epsilon_n)$, its linear complexity $L(s^n)$ is defined as the length of the shortest LFSR that generates s^n as its first n terms. The possibility of using the linear complexity characteristic for testing randomness is based on the Berlekamp-Massey algorithm, which provides an efficient way to evaluate finite strings.

When the binary n-sequence s^n is truly random, formulas exist [2] for the mean, $\mu_n = EL(s^n)$, and the variance, $\sigma_n^2 = Var(L(s^n))$, of the linear complexity $L(s^n) = L_n$ when the n-sequence s^n is truly random. The Crypt-X package [1] suggests that the ratio $(L_n - \mu_n)/\sigma_n$ is close to a standard normal variable, so that the corresponding $P-values$ can be found from the normal error function. Indeed, Gustafson et al. [1] (p. 693) claim that "for large n, $L(s^n)$ is approximately normally distributed with mean $n/2$ and a variance 86/81 times that of the standard normal statistic $z = \left(L(s^n) - \frac{n}{2}\right)\sqrt{\frac{81}{86}}$." This is completely false. Even the mean value μ_n does not behave asymptotically precisely as $n/2$, and in view of the boundedness of the variance, this difference becomes significant. More importantly, the tail probabilities of the limiting distribution are much larger than those of the standard normal distribution.

The asymptotic distribution of $(L_n - \mu_n)/\sigma_n$ along the sequence of even or odd values of n is that of a discrete random variable obtained via a mixture of two geometric random variables (one of them taking only negative values). Strictly speaking, the asymptotic distribution as such does not exist. The cases n even and n odd must be treated separately with two different limiting distributions arising.

Because of this fact the following sequence of statistics is adapted

$$T_n = (-1)^n[L_n - \xi_n] + \frac{2}{9}. \tag{6}$$

Here

$$\xi_n = \frac{n}{2} + \frac{4 + r_n}{18}. \tag{7}$$

These statistics, which take only integer values, converge in distribution to the random variable T. This limiting distribution is skewed to the right. While $P(T = 0) = \frac{1}{2}$, for $k = 1, 2. \ldots$

$$P(T = k) = \frac{1}{2^{2k}}, \tag{8}$$

for $k = -1, -2, \ldots$

$$P(T = k) = \frac{1}{2^{2|k|+1}}. \tag{9}$$

It follows from (8) that

$$P(T \geq k > 0) = \frac{1}{3 \times 2^{2k-2}};$$

for $k < 0$ (9) shows that

$$P(T \leq k) = \frac{1}{3 \times 2^{2|k|-1}}.$$

So the $P - value$ corresponding to the observed value T_{obs} can be evaluated in the following way. Let $\kappa = [|T_{\text{obs}}|] + 1$. Then the $P - value$ is

$$\frac{1}{3 \times 2^{2\kappa-1}} + \frac{1}{3 \times 2^{2\kappa-2}} = \frac{1}{2^{2\kappa-1}}.$$

In view of the discrete nature of this distribution and the impossibility of attaining the uniform distribution for $P - values$, the same strategy can be used that was used with other tests in this situation. Namely, partition the string of length n, such that that $n = MN$, into N substrings each of length M. For the test based on the linear complexity statistic (6), evaluate T_M within the j-th substring of size M, and choose $K + 1$ classes (depending on M.) For each of these substrings, the frequencies, $\nu_0, \nu_1, \ldots, \nu_K$, of values of T_M belonging to any of $K+1$ chosen classes, $\nu_0+\nu_1+\ldots+\nu_K = N$, are determined. It is convenient to choose the classes with end-points at semi-integers.

The theoretical probabilities $\pi_0, \pi_1, \ldots, \pi_K$ of these classes are determined from (8) and (9). For this purpose, M has to be large enough for the limiting distribution given by (8) and (9) to provide a reasonable approximation. M should exceed 500. It is recommended that M be chosen so that 500 $\leq$M$\leq$ 5000.

The frequencies are conjoined by the χ^2-statistic

$$\chi^2 = \sum_0^K \frac{(\nu_i - N\pi_i)^2}{N\pi_i}.$$

which, under the randomness hypothesis, has an approximate χ^2-distribution with K degrees of freedom. The reported $P - value$ is

$$\frac{\int_{\chi^2(obs)}^{\infty} e^{-u/2} u^{K/2-1}\, du}{\Gamma\left(K/2\right) 2^{K/2}} = \mathbf{igamc}\left(\frac{K}{2}, \frac{\chi^2(obs)}{2}\right).$$

As before, a conservative condition for the use of the χ^2-approximation is that

$$N \min_i \pi_i \geq 5.$$

For reasonably large values of M and N, the following classes ($K = 6$) seem to be adequate: $\{T \leq -2.5\}$, $\{-2.5 < T \leq -1.5\}$, $\{-1.5 < T \leq -0.5\}$, $\{-0.5 < T \leq 0.5\}$, $\{0.5 < T \leq 1.5\}$, $\{1.5 < T \leq 2.5\}$, and $\{T > 2.5\}$.

The probabilities of these classes are $\pi_0 = 0.01047, \pi_1 = 0.03125, \pi_2 = 0.12500$, $\pi_3 = 0.50000, \pi_4 = 0.25000, \pi_5 = 0.06250, \pi_6 = 0.020833$. These probabilities are substantially different from the ones obtained from the normal approximation for which their numerical values are: $0.0041, 0.0432, 0.1944, 0.3646, 0.2863, 0.0939, 0.0135$.

References for Test

[1] H. Gustafson, E. Dawson, L. Nielsen, and W. Caelli (1994), "A computer package for measuring the strength of encryption algorithms," Computers and Security. 13, pp. 687-697.

[2] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone (1997), Handbook of Applied Cryptography. CRC Press, Boca Raton, FL.

[3] R.A. Rueppel, Analysis and Design of Stream Ciphers. New York: Springer, 1986.

3.12 Serial Test

The (generalized) serial test represents a battery of procedures based on testing the uniformity of distributions of patterns of given lengths.

Specifically, for $i_1, \cdots, i_m$ running through the set of all 2^m possible $0, 1$ vectors of length m, let $\nu_{i_1 \cdots i_m}$ denote the frequency of the pattern $(i_1, \cdots, i_m)$ in the "circularized" string of bits $(\epsilon_1, \ldots, \epsilon_n, \epsilon_1, \ldots, \epsilon_{m-1})$.

Set

$$\psi_m^2 = \frac{2^m}{n} \sum_{i_1 \cdots i_m} \left(\nu_{i_1 \cdots i_m} - \frac{n}{2^m} \right)^2 = \frac{2^m}{n} \sum_{i_1 \cdots i_m} \nu_{i_1 \cdots i_m}^2 - n,$$

Thus, ψ_m^2 is a χ^2-type statistic, but it is a common mistake to assume that ψ_m^2 has the χ^2-distribution. Indeed, the frequencies $\nu_{i_1 \cdots i_m}$ are not independent.

The corresponding generalized serial statistics for the testing of randomness (Kimberley (1987), Knuth, D. E. (1998), Menezes, van Oorschot and Vanstone, (1997)) are

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

and

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

(Here $\psi_0^2 = \psi_{-1}^2 = 0$.) Then $\nabla \psi_m^2$ has a χ^2-distribution with 2^{m-1} degrees of freedom, and $\nabla^2 \psi_m^2$ has a χ^2-distribution with 2^{m-2} degrees of freedom. Thus, for small values of m, $m \leq \lfloor \log_2(n) \rfloor - 2$, one can find the corresponding $2m$ $P - values$ from the standard formulas.

$$P - value1 \ = \mathbf{igamc} \left(2^{m-2}, \nabla \Psi_m^2 \right)$$

$$P - value2 \ = \mathbf{igamc} \left(2^{m-3}, \nabla^2 \Psi_m^2 \right)$$

The result for $\nabla \psi_2^2$ and the usual counting of frequencies is incorrectly given by Menezes, van Oorschot and Vanstone (1997) on p. 181, formula (5.2): $+1$ should be replaced by -1.

The convergence of $\nabla \psi_m^2$ to the χ^2- distribution was proven by Good (1953).

References for Test

[1] I. J. Good (1953), "The serial test for sampling numbers and other tests for randomness," Proc. Cambridge Philos. Soc.. 47, pp. 276-284.

[2] M. Kimberley (1987), "Comparison of two statistical tests for keystream sequences," Electronics Letters. 23, pp. 365-366.

[3] D. E. Knuth (1998), The Art of Computer Programming. Vol. 2, 3rd ed. Reading: Addison-Wesley, Inc., pp. 61-80.

[4] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone (1997), Handbook of Applied Cryptography. Boca Raton, FL: CRC Press, p. 181.

3.13 Approximate Entropy Test

Approximate entropy characteristics (Pincus and Singer, 1996) are based on repeating patterns in the string. If $Y_i(m) = (\epsilon_i, \ldots, \epsilon_{i+m-1})$, set

$$C_i^m = \frac{1}{n+1-m} \# \{j : 1 \leq j < n-m, Y_j(m) = Y_i(m)\} = \pi_\ell$$

and

$$\Phi^{(m)} = \frac{1}{n+1-m} \sum_{i=1}^{n+1-m} \log C_i^m,$$

C_i^m is the relative frequency of occurrences of the pattern $Y_i(m)$ in the string, and $-\Phi^{(m)}$ is the entropy of the empirical distribution arising on the set of all 2^m possible patterns of length m,

$$\Phi^{(m)} = \sum_{\ell=1}^{2^m} \pi_\ell \log \pi_\ell,$$

where π_ℓ is the relative frequency of pattern $\ell = (i_1, \cdots, i_m)$ in the string.

The approximate entropy $ApEn$ of order $m, m \geq 1$ is defined as

$$ApEn(m) = \Phi^{(m)} - \Phi^{(m+1)}$$

with $ApEn(0) = -\Phi^{(1)}$. "$ApEn(m)$ measures the logarithmic frequency with which blocks of length m that are close together remain close together for blocks augmented by one position. Thus, small values of $ApEn(m)$ imply strong regularity, or persistence, in a sequence. Alternatively, large values of $ApEn(m)$ imply substantial fluctuation, or irregularity." (Pincus and Singer,

1996, p. 2083).

Pincus and Kalman (1997) defined a sequence to be m-irregular (m-random) if its approximate entropy $ApEn(m)$ takes the largest possible value. They evaluated quantities $ApEn(m), m = 0, 1, 2$ for binary and decimal expansions of $e, \pi, \sqrt{2}$ and $\sqrt{3}$ with the surprising conclusion that the expansion of $\sqrt{3}$ demonstrated more irregularity than that of π.

For a fixed block length m, one should expect that in long random (irregular) strings, $ApEn(m) \sim \log 2$. The limiting distribution of $n[\log 2 - ApEn(m)]$ coincides with that of a χ^2-random variable with 2^m degrees of freedom. This fact provides the basis for a statistical test, as was shown by Rukhin (2000).

Thus, with $\chi^2(obs) = n[\log 2 - ApEn(m)]$, the reported $P - value$ is

$$\mathbf{igamc}\left(2^{m-1}, \chi^2(obs)/2\right).$$

Actually, this limiting distribution of approximate entropy is more exact for its modified definition as

$$\tilde{\Phi}^{(m)} = \sum_{i_1 \cdots i_m} \nu_{i_1 \cdots i_m} \log \nu_{i_1 \cdots i_m},$$

where $\nu_{i_1 \cdots i_m}$ denotes the relative frequency of the template $(i_1, \cdots, i_m)$ in the augmented (or circular) version of the original string, i.e., in the string $(\epsilon_1, \ldots, \epsilon_n, \epsilon_1, \ldots, \epsilon_{m-1})$. Let $\omega_{i_1 \cdots i_m} = n\nu_{i_1 \cdots i_m}$ be the frequency of the pattern $i_1 \cdots i_m$. Under our definition, $\omega_{i_1 \cdots i_m} = \sum_k \omega_{i_1 \cdots i_m k}$, so that for any m, $\sum_{i_1 \cdots i_m} = n$.

Define the modified approximate entropy as

$$\widetilde{ApEn}(m) = \tilde{\Phi}^{(m)} - \tilde{\Phi}^{(m+1)}.$$

By Jensen's inequality, $\log s \geq \widetilde{ApEn}(m)$ for any m, whereas it is possible that $\log s < ApEn(m)$. Therefore, the largest possible value of the modified entropy is merely $\log s$, which is attained when $n = s^m$, and the distribution of all m-patterns is uniform. When calculating the approximate entropy for several values of m, it is very convenient to have the sum of all frequencies of m-templates be equal to n.

When n is large, $ApEn(m)$ and its modified version cannot differ much. Indeed, one has with $\omega'_{i_1\cdots i_m} = (n-m+1)\nu'_{i_1\cdots i_m}$

$$\sum_{i_1\cdots i_m} \omega'_{i_1\cdots i_m} = n - m + 1,$$

and $\omega_{i_1\cdots i_m} - \omega'_{i_1\cdots i_m} \leq m - 1$. It follows that

$$\left|\nu_{i_1\cdots i_m} - \nu'_{i_1\cdots i_m}\right| \leq \frac{m-1}{n-m+1},$$

which suggests that for a fixed m, $\Phi^{(m)}$ and $\tilde{\Phi}^{(m)}$ must be close for large n. Therefore, Pincus' approximate entropy and its modified version are also close, and their asymptotic distributions must coincide.

References for Test

[1] S. Pincus and B. H. Singer, "Randomness and degrees of irregularity," Proc. Natl. Acad. Sci. USA. Vol. 93, March 1996, pp. 2083-2088.

[2] S. Pincus and R. E. Kalman, "Not all (possibly) "random" sequences are created equal," Proc. Natl. Acad. Sci. USA. Vol. 94, April 1997, pp. 3513-3518.

[3] A. Rukhin (2000), "Approximate entropy for testing randomness," Journal of Applied Probability. Vol. 37, 2000.

3.14 Cumulative Sums (Cusum) Test

This test is based on the maximum absolute value of the partial sums of the sequence represented in the ± 1 fashion. Large values of this statistic indicate that there are either too many ones or too many zeros at the early stages of the sequence. Small values indicate that ones and zeros are intermixed too evenly. A dual test can be derived from the reversed time random walk with $S'_k = X_n + \cdots + X_{n-k+1}$. With this definition, the interpretation of the test results is modified by replacing "the early stages" by "the late stages."

The test is based on the limiting distribution of the maximum of the absolute values of the partial sums, $\max_{1\le k\le n}|S_k|$,

$$\lim_{n\to\infty} P\left(\frac{\max_{1\le k\le n}|S_k|}{\sqrt{n}} \le z\right) = \frac{1}{\sqrt{2\pi}}\int_{-z}^{z}\sum_{k=-\infty}^{\infty}(-1)^k \exp\left\{-\frac{(u-2kz)^2}{2}\right\}\,du$$

$$= \frac{4}{\pi}\sum_{j=0}^{\infty}\frac{(-1)^j}{2j+1}\exp\left\{-\frac{(2j+1)^2\pi^2}{8z^2}\right\} = H(z), \quad z>0. \tag{10}$$

With the test statistic $z = \max_{1\le k\le n}|S_k|(obs)/\sqrt{n}$, the randomness hypothesis is rejected for large values of z, and the corresponding $P-value$ is $1-H\left(\max_{1\le k\le n}|S_k|(obs)/\sqrt{n}\right) = 1-G\left(\max_{1\le k\le n}|S_k|(obs)/\sqrt{n}\right)$ where the function $G(z)$ is defined by the formula (11).

The series $H(z)$ in the last line of (10) converges quickly and should be used for numerical calculation only for small values of z. The function $G(z)$ (which is equal to $H(z)$ for all z) is preferable for the calculation for moderate and large values of $\max_{1\le k\le n}|S_k|(obs)/\sqrt{n}$,

$$G(z) = \frac{1}{\sqrt{2\pi}}\int_{-z}^{z}\sum_{k=-\infty}^{\infty}(-1)^k \exp\{-\frac{(u-2kz)^2}{2}\}\,du$$

$$= \sum_{k=-\infty}^{\infty}(-1)^k\left[\Phi((2k+1)z)-\Phi((2k-1)z)\right]$$

$$= \Phi(z)-\Phi(-z)+2\sum_{k=1}^{\infty}(-1)^k\left[\Phi((2k+1)z)-\Phi((2k-1)z)\right]$$

$$= \Phi(z)-\Phi(-z)-2\sum_{k=1}^{\infty}\left[2\Phi((4k-1)z)-\Phi((4k+1)z)-\Phi((4k-3)z)\right]$$

$$\approx \Phi(z)-\Phi(-z)-2\left[2\Phi(3z)-\Phi(5z)-\Phi(z)\right]$$

$$\approx 1-\frac{4}{\sqrt{2\pi}z}\exp\{-\frac{z^2}{2}\}, \quad z\to\infty. \tag{11}$$

where $\Phi(x)$ is the standard normal distribution.

More directly, using Theorem 2.6, p. 17 of Revesz (1990), one obtains

$$P\left(\max_{1\le k\le n}|S_k| \ge z\right)$$

$$= 1 - \sum_{k=-\infty}^{\infty} P\left((4k-1)z < S_n < (4k+1)z\right)$$

$$+ \sum_{k=-\infty}^{\infty} P\left((4k+1)z < S_n < (4k+3)z\right).$$

This formula is used for the evaluation of the $P - values$ with

$$z = \max_{1 \le k \le n} |S_k|(obs)/\sqrt{n}.$$

The randomness hypothesis is rejected for large values of z.

References for Test

[1] Frank Spitzer, Principles of Random Walk. Princeton: Van Nostrand, 1964 (especially p. 269).

[2] Pal Revesz, Random Walk in Random And Non-Random Environments. Singapore: World Scientific, 1990.

3.15 Random Excursions Test

This test is based on considering successive sums of the binary bits (plus or minus simple ones) as a one-dimensional random walk. The test detects deviations from the distribution of the number of visits of the random walk to a certain "state," i.e., any integer value.

Consider the random walk $S_k = X_1 + \ldots X_k$ as a sequence of excursions to and from zero

$$(i, \ldots, \ell) \ : S_{i-1} = S_{\ell+1} = 0, \ S_k \neq 0 \text{ for } i \le k \le \ell.$$

Let J denote the total number of such excursions in the string. The limiting distribution for this (random) number J (i.e., the number of zeros among the sums $S_k, k = 1, 2, \ldots, n$ when $S_0 = 0$) is known to be

$$\lim_{n \to \infty} P\left(\frac{J}{\sqrt{n}} < z\right) = \sqrt{\frac{2}{\pi}} \int_0^z e^{-u^2/2}\, du, \quad z > 0. \tag{12}$$

The test rejects the randomness hypothesis immediately if J is too small, i.e., if the following $P-value$ is small:

$$P\left(J < J(obs)\right) \approx \sqrt{\frac{2}{\pi}} \int_0^{J(obs)/\sqrt{n}} e^{-u^2/2}\, du = \mathbf{P}\left(\frac{1}{2}, \frac{J^2(obs)}{2n}\right).$$

If $J < \max(0.005\sqrt{n}, 500)$, the randomness hypothesis is rejected. Otherwise the number of visits of the random walk S to a certain state is evaluated.

Let $\xi(x)$ be the number of visits to $x, x \neq 0$, during one 0-excursion. Its distribution is derived in Revesz (1990) and Baron and Rukhin (1999):

$$P\left(\xi(x) = 0\right) = 1 - \frac{1}{2|x|} \tag{13}$$

and for $k = 1, 2, \ldots$

$$P\left(\xi(x) = k\right) = \frac{1}{4x^2}\left(1 - \frac{1}{2|x|}\right)^{k-1}. \tag{14}$$

This means that $\xi(x) = 0$ with probability $1 - 1/2|x|$; otherwise (with probability $1/2|x|$), $\xi(x)$ coincides with a geometric random variable with the parameter $1/2|x|$.

It is easy to see that

$$E\xi(x) = 1,$$

and

$$Var(\xi(x)) = 4|x| - 2.$$

A useful formula is:

$$P(\xi(x) \geq a+1) = 2xP(\xi(x) = a+1) = \frac{1}{2|x|}\left(1 - \frac{1}{2|x|}\right)^a, \quad a = 0, 1, 2, \ldots. \tag{15}$$

The above results are used for randomness testing in the following way. For a "representative" collection of x-values (say, $1 \leq x \leq 7$ or $-7 \leq x \leq -1$: $-4 \leq x \leq 4$ is used in the test suite code), evaluate the observed frequencies

$\nu_k(x)$ of the number k of visits to the state x during J excursions which occur in the string. So $\nu_k(x) = \sum_{j=1}^{J} \nu_k^j(x)$ with $\nu_k^j(x) = 1$ if the number of visits to x during the jth excursion ($j = 1, \ldots, J$) is exactly equal to k, and $\nu_k^j(x) = 0$ otherwise. Pool the values of $\xi(x)$ into classes, say, $k = 0, 1, \ldots, 4$ with an additional class $k \geq 5$. The theoretical probabilities for these classes are:

$$\pi_0(x) = P\left(\xi(x) = 0\right) = 1 - \frac{1}{2|x|};$$

$$\pi_k(x) = P\left(\xi(x) = k\right) = \frac{1}{4x^2}\left(1 - \frac{1}{2|x|}\right)^{k-1}, k = 1, \ldots, 4;$$

$$\pi_5(x) = P(\xi(x) \geq 5) = \frac{1}{2|x|}\left(1 - \frac{1}{2|x|}\right)^5.$$

These probabilities have the form

	$\pi_0(x)$	$\pi_1(x)$	$\pi_2(x)$	$\pi_3(x)$	$\pi_4(x)$	$\pi_5(x)$
$x = 1$	0.5000	0.2500	0.1250	0.0625	0.0312	0.0312
$x = 2$	0.7500	0.0625	0.0469	0.0352	0.0264	0.0791
$x = 3$	0.8333	0.0278	0.0231	0.0193	0.0161	0.0804
$x = 4$	0.8750	0.0156	0.0137	0.0120	0.0105	0.0733
$x = 5$	0.9000	0.0100	0.0090	0.0081	0.0073	0.0656
$x = 6$	0.9167	0.0069	0.0064	0.0058	0.0053	0.0588
$x = 7$	0.9286	0.0051	0.0047	0.0044	0.0041	0.0531

Compare these frequencies to the theoretical ones using the χ^2-test,

$$\chi^2(x) = \sum_{k=0}^{5} \frac{(\nu_k(x) - J\pi_k(x))^2}{J\pi_k(x)},$$

which, for any x under the randomness hypothesis, must have approximately a χ^2-distribution with 5 degrees of freedom. This is a valid test when $J \min \pi_k(x) \geq 5$, i.e., if $J \geq 500$. (The test suite code uses $\pi_4(x = 4)$ for $\min \pi_k(x)$.) If this condition does not hold, values of $\xi(x)$ must be pooled into larger classes.

The corresponding battery of $P - values$ is reported. These values are obtained from the formula

$$1 - \mathbf{P}\left(\frac{5}{2}, \frac{\chi^2(obs)(x)}{2}\right).$$

References for Test

[1] M. Baron and A. L. Rukhin, "Distribution of the Number of Visits For a Random Walk," Communications in Statistics: Stochastic Models. Vol. 15, 1999, pp. 593-597.

[2] Pal Revesz, Random Walk in Random and Non-random Environments. Singapore: World Scientific, 1990.

[3] Frank Spitzer, Principles of Random Walk. Princeton: Van Nostrand, 1964, (especially p. 269).

3.16 Random Excursions Variant Test

An alternative to the random excursions test can be derived as follows. Using the notation of the previous subsection, let $\xi_J(x)$ be the total number of visits to x during J excursions. (The test suite code assumes $J \geq 500$.) Since S_k renews at every zero, $\xi_J(x)$ is a sum of independent identically distributed variables with the same distribution as $\xi(x) = \xi_1(x)$. Therefore, the limiting distribution of $\xi_J(x)$,

$$\lim_{J\to\infty} P\left(\frac{\xi_J(x) - J}{\sqrt{J(4|x| - 2)}} < z\right) = \Phi(z),$$

is normal. The randomness hypothesis will be rejected when the $P - value$

$$erfc\left(\frac{|\xi_J(x)(obs) - J|}{\sqrt{2J(4|x| - 2)}}\right)$$

is small.

References for Test

[1] M. Baron and A. L. Rukhin, "Distribution of the Number of Visits For a Random Walk," Communications in Statistics: Stochastic Models. Vol. 15, 1999.

[2] Pal Revesz, Random Walk in Random and Non-random Environments. Singapore: World Scientific, 1990.

[3] Frank Spitzer, Principles of Random Walk. Princeton: Van Nostrand, 1964 (especially p. 269).

4. TESTING STRATEGY AND RESULT INTERPRETATION

Three topic areas will be addressed in this section: (1) strategies for the statistical analysis of a random number generator, (2) the interpretation of empirical results using the NIST Statistical Test Suite, and (3) general recommendations and guidelines.

4.1 Strategies for the Statistical Analysis of an RNG

In practice, there are many distinct strategies employed in the statistical analysis of a random number generator. NIST has adopted the strategy outlined in Figure 1. Figure 1 provides an architectural illustration of the five stages involved in the statistical testing of a random number generator.

Stage 1: Selection of a Generator

Select a hardware or software based generator for evaluation. The generator should produce a binary sequence of 0's and 1's of a given length *n*. Examples of pseudorandom generators (PRNG) that may be selected include a DES-based PRNG from ANSI X9.17 (Appendix C), and two further methods that are specified in FIPS 186 (Appendix 3) and are based on the Secure Hash Algorithm (SHA-1) and the Data Encryption Standard (DES**)**.

Stage 2: Binary Sequence Generation

For a fixed sequence of length *n* and the pre-selected generator, construct a set of *m* binary sequences and save the sequences to a file[7].

Stage 3: Execute the Statistical Test Suite

Invoke the NIST Statistical Test Suite using the file produced in Stage 2 and the desired sequence length. Select the statistical tests and relevant input parameters (e.g., block length) to be applied.

Stage 4: Examine the *P-values*

An output file will be generated by the test suite with relevant intermediate values, such as test statistics, and *P-values* for each statistical test. Based on these *P-values*, a conclusion regarding the quality of the sequences can be made.

Stage 5: Assessment: Pass/Fail Assignment

[7] Sample data may also be obtained from George Marsaglia's *Random Number CDROM*, at http://stat.fsu.edu/pub/diehard/cdrom/.

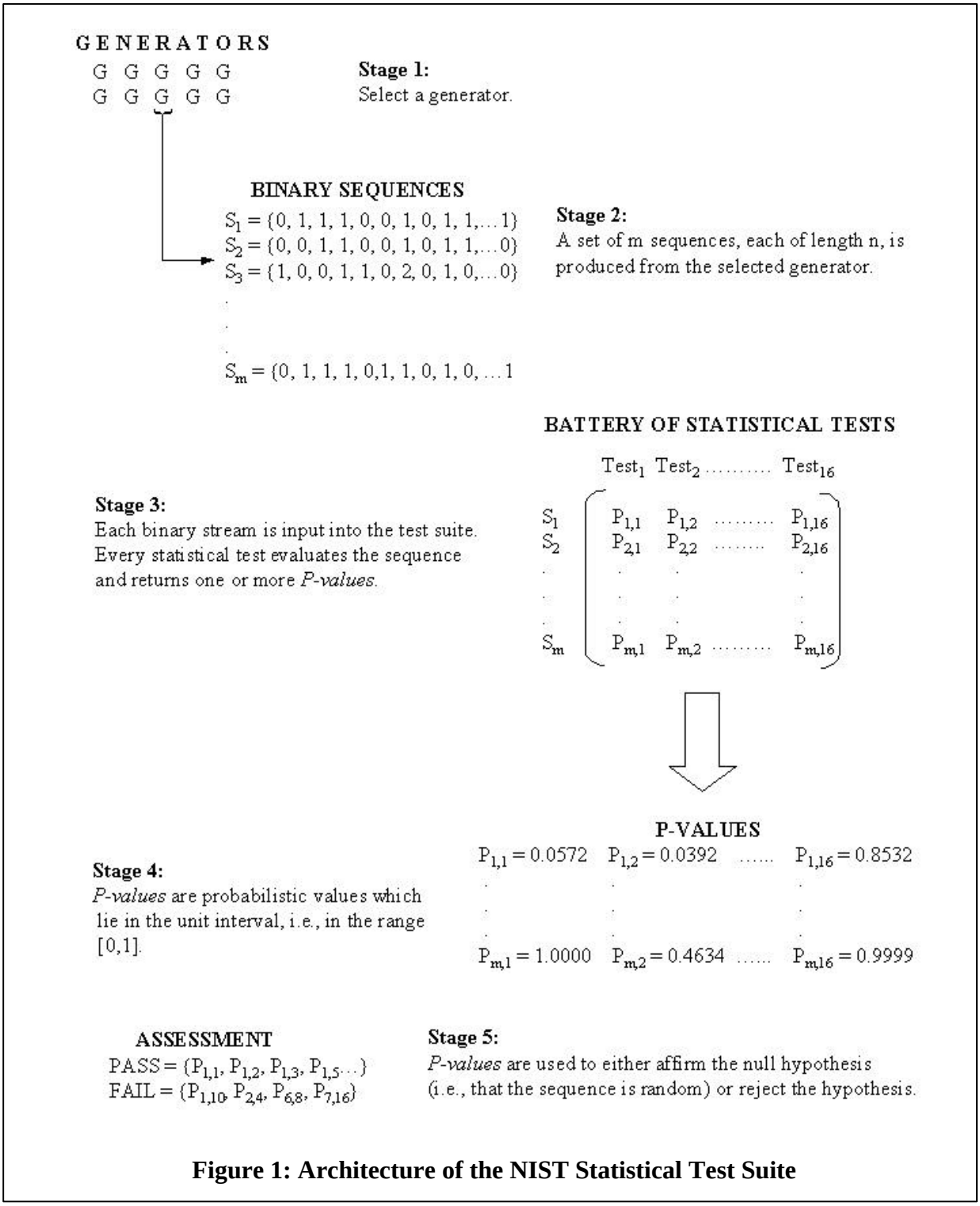


Figure 1: Architecture of the NIST Statistical Test Suite

For each statistical test, a set of *P-values* (corresponding to the set of sequences) is produced. For a fixed significance level, a certain percentage of *P-values* are expected to indicate failure. For example, if the significance level is chosen to be 0.01 (i.e., $a = 0.01$), then about 1 % of the sequences are expected to fail. A sequence passes a statistical test whenever the *P-value* ³ *a*

and fails otherwise. For each statistical test, the proportion of sequences that pass is computed and analyzed accordingly. More in-depth analysis should be performed using additional statistical procedures (see Section 4.2.2).

4.2 The Interpretation of Empirical Results

Three scenarios typify events that may occur due to empirical testing. Case 1: The analysis of the *P-values* does not indicate a deviation from randomness. Case 2: The analysis clearly indicates a deviation from randomness. Case 3: The analysis is inconclusive.

The interpretation of empirical results can be conducted in any number of ways.
Two approaches NIST has adopted include (1) the examination of the proportion of sequences that pass a statistical test and (2) the distribution of *P-values* to check for uniformity.

In the event that either of these approaches fails (i.e., the corresponding null hypothesis must be rejected), additional numerical experiments should be conducted on different samples of the generator to determine whether the phenomenon was a statistical anomaly or a clear evidence of non-randomness.

4.2.1 Proportion of Sequences Passing a Test

Given the empirical results for a particular statistical test, compute the proportion of sequences that pass. For example, if 1000 binary sequences were tested (i.e., $m = 1000$), $a = 0.01$ (the significance level), and 996 binary sequences had *P-values* ³ *.01*, then the proportion is *996/1000 = 0.9960*.

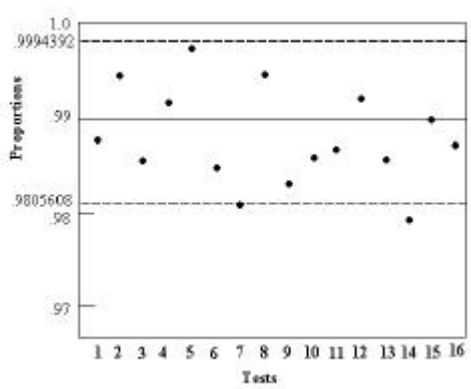


Figure 2: P-value Plot

The range of acceptable proportions is determined using the confidence interval defined as, $\hat{p} \pm 3\sqrt{\frac{\hat{p}(1-\hat{p})}{m}}$, where $\hat{p}$ = 1-α, and m is the sample size. If the proportion falls outside of this interval, then there is evidence that the data is non-random. Note that other standard deviation values could be used. For the example above, the confidence interval is $.99 \pm 3\sqrt{\frac{.99(.01)}{1000}} = .99 \pm 0.0094392$ (i.e., the proportion should lie above 0.9805607. This can be illustrated using a graph as shown in Figure 2. The confidence interval was calculated using a normal distribution as an approximation to the binomial distribution, which is reasonably accurate for large sample sizes (e.g., $n \geq 1000$).

4.2.2 Uniform Distribution of *P-values*

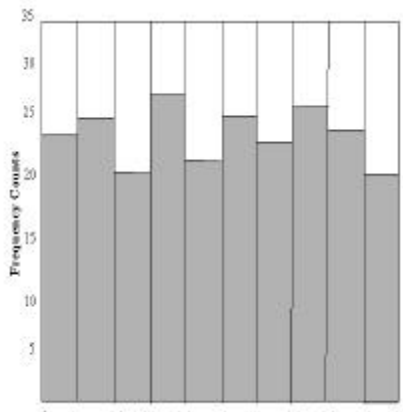


Figure 3: Histogram of P-values

The distribution of *P-values* is examined to ensure uniformity. This may be visually illustrated using a histogram (see Figure 3), whereby, the interval between 0 and 1 is divided into 10 sub-intervals, and the *P-values* that lie within each sub-interval are counted and displayed.

Uniformity may also be determined via an application of a χ^2 test and the determination of a *P-value* corresponding to the Goodness-of-Fit Distributional Test on the *P-values* obtained for an arbitrary statistical test (i.e., a *P-value* of the *P-values*). This is accomplished by computing $c^2 = \sum_{i=1}^{10} \frac{(F_i - s/10)^2}{s/10}$, where F_i is the number of *P-values* in sub-interval *i*, and *s* is the sample size. A *P-value* is calculated such that $P\text{-}value_T =$ **igamc**$(9/2, c^2/2)$. If $P\text{-}value_T$ ³ *0.0001*, then the sequences can be considered to be uniformly distributed.

4.3 General Recommendations and Guidelines

In practice, many reasons can be given to explain why a data set has failed a statistical test. The following is a list of possible explanations. The list was compiled based upon NIST statistical testing efforts.

(a) **An incorrectly programmed statistical test.**

Unless otherwise specified, it should be assumed that a statistical test was tailored to handle a particular problem class. Since the NIST test code has been written to allow the selection of input parameters, the code has been generalized in any number of ways. Unfortunately, this doesn't necessarily translate to coding ease.

A few statistical tests have been constrained with artificial upper bounds. For example, the random excursions tests are assumed to be no more than max{1000, *n/128*} cycles. Similarly, the Lempel-Ziv Compression test assumes that the longest word is in the neighborhood of $log_2 n$, where *n* is the sequence length. Conceivably, fixed parameters may have to be increased, depending on experimental conditions.

(b) **An underdeveloped (immature) statistical test.**

There are occasions when either probability or complexity theory isn't sufficiently developed or understood to facilitate a rigorous analysis of a statistical test.

Over time, statistical tests are revamped in light of new results. Since many statistical tests are based upon asymptotic approximations, careful work needs to be done to determine how good an approximation is.

(c) **An improper implementation of a random number generator.**

It might be plausible that a hardware RNG or a software RNG has failed due to a flaw in the design or due to a coding implementation error. In each case, careful review must be made to rule out this possibility.

(d) **Improperly written codes to harness test input data.**

Another area that needs to be scrutinized is the harnessing of test data. The test data produced by a (P)RNG must be processed before being used by a statistical test. For example, processing might include dividing the output stream from the (P)RNG into appropriate sized blocks, and translating the 0's to negative ones. On occasion, it was determined that the failures from a statistical test were due to errors in the code used to process the data.

(e) **Poor mathematical routines for computing *P-values***

Quality math software must be used to ensure excellent approximations whenever possible. In particular, the incomplete gamma function is more difficult to approximate for larger values of the constant a. Eventually, *P-value* formulas will result in bogus values due to difficulties arising from numerical approximations. To reduce the likelihood of this event, NIST has prescribed preferred input parameters.

(f) **Incorrect choices for input parameters.**

In practice, a statistical test will not provide reliable results for all seemingly valid input parameters. It is important to recognize that constraints are made upon tests on a test-by-test basis. Take the Approximate Entropy Test, for example. For a sequence length on the order of 10^6, one would expect that block lengths approaching $log_2 n$ would be acceptable. Unfortunately, this is not the case. Empirical evidence suggests that beyond $m = 14$, the observed test statistic will begin to disagree with the expected value (in particular, for known good generators, such as SHA-1). Hence, certain statistical tests may be sensitive to input parameters.

Considerations must often be made regarding the numerical experimentation input parameters, namely: sequence length, sample size, block size and template.

Sequence Length

The determination as to how long sequences should be taken for the purposes of statistical testing is difficult to address. If one examines the FIPS 140-1 statistical tests, it is evident that sequences should be about 20,000 bits long.

However, the difficulty with taking relatively short sequence lengths is problematic in the sense that some statistical tests, such as Maurer's Universal Statistical Test, require extremely long sequence lengths. One of the reasons is the realization that asymptotic approximations are used in determining the limiting distribution. Statements regarding the distribution for certain test statistics are more difficult to address for short length sequences than their longer length counterparts.

Sample Size

The issue of sample size is tied to the choice of the significance level. NIST recommends that, for these tests, the user should fix the significance level to be at least 0.001, but no larger than 0.01[8]. A sample size that is disproportional to the significance level may not be suitable. For example, if the significance level (α) is chosen to be 0.001, then it is expected that 1 out of every 1000 sequences will be rejected. If a sample of only 100 sequences is selected, it would be rare to observe a rejection. In this case, the conclusion may be drawn that a generator was producing random sequences, when in all likelihood a sufficiently large enough sample was not used. Thus, the sample should be on the order of the inverse of the significance level (a^{-1}). That is, for a level of 0.001, a sample should have at least 1000 sequences. Ideally, many distinct samples should be analyzed.

Block Size

Block sizes are dependent on the individual statistical test. In the case of Maurer's Universal Statistical test, block sizes range from 1 to 16. However, for each specific block size, a minimum sequence length should be used. If the block size were fixed at 16, a sequence of more than a billion bits would be required. For some users, that may not be feasible.

Intuitively, it would seem that the larger the block size, the more information could be gained from the parsing of a sequence, such as in the Approximate Entropy test. However, a block size that is too large should not be selected either, for otherwise the empirical results may be misleading and incorrect because the test statistic is better approximated by a distinct probability distribution. In practice, NIST advises selecting a block size no larger than $\lfloor \log_2 n \rfloor$, where n is the sequence length. However, certain exceptions hold, and thus NIST suggests choosing a smaller block size.

[8] Note that for FIPS 140-2, the significance level has been set to 0.0001 for the power up tests.

Template

Certain statistical tests are suited for detecting global non-randomness. However, other statistical tests are more apt at assessing local non-randomness, such as tests developed to detect the presence of too many *m*-bit patterns in a sequence. Still, it makes sense that templates of a block size greater than $\lfloor \log_2 n \rfloor$ should not be chosen, since frequency counts will most probably be in the neighborhood of zero, which does not provide any useful information. Thus, appropriate choices must be made.

Other Considerations

In principle, there are many commonly occurring questions regarding randomness testing. Perhaps the most frequently asked question is, "*How many tests should one apply*?" In practice, no one can truly answer this question. The belief is that the tests should be independent of each other as much as possible.

Another, frequently asked question concerns the need for applying a monobits test (i.e., Frequency test), in lieu of Maurer's Universal Statistical test. The perception is that Maurer's Universal Statistical test supercedes the need to apply a monobits test. This may hold true for infinite length sequences. However, it is important to keep in mind that there will be instances when a finite binary sequence will pass Maurer's Universal Statistical test, yet fail the monobits test. Because of this fact, NIST recommends that the Frequency test be applied first. If the results of this test support the null hypothesis, then the user may proceed to apply other statistical tests.

4.4 Application of Multiple Tests

Given a concern regarding the application of multiple tests, NIST performed a study to determine the dependence between the tests. The performance of the tests was checked by using a Kolmogorov-Smirnov test of uniformity on the *P-values* obtained from the sequences. However, it required an assumption that the sequences that were generated to test uniformity were sufficiently random. There are many tests in the suite. Some tests should intuitively give independent answers (e.g., the frequency test and a runs test that conditions on frequencies should assess completely different aspects of randomness). Other tests, such as the cusum test and the runs test, result in *P-values* that are likely to be correlated.

To understand the dependencies between the tests in order to eliminate redundant tests, and to ensure that the tests in the suite are able to detect a reasonable range of patterned behaviors, a factor analysis of the resulting *P-values* was performed. More precisely, in order to assess independence, *m* sequences of binary pseudorandom digits were generated, each of length *n*, and all k=161 tests in the suite were applied to those sequences to determine their randomness. Each test produced a significance probability; denote by p_{ij} the significance probability of test *i* on sequence *j*.

Given the uniformly distributed p_{ij}, the transformation $z_{ij} = F^{-1}\left(p_{ij}\right)$ leads to normally distributed variables. Let z_j be the vector of transformed significance probabilities corresponding to the ith sequence. A principal components analysis was performed on the $z_1, \ldots, x_m$. Usually, a small number of components suffices to explain a great proportion of the variability, and the number of these components can be used to quantify the number of "dimensions'" of nonrandomness spanned by the suite tests. The principal component analysis of this data was performed. This analysis extracts 161 factors, equal to the number of tests. The first factor is the one that explains the largest variability. If many tests are correlated, their *P-values* will greatly depend on this factor, and the fraction of total variability explained by this factor will be large. The second factor explains the second largest proportion of variability, subject to the constraint that the second factor is orthogonal to the first, and so on for subsequent factors. The corresponding fractions corresponding to the first 50 factors were plotted for the tests, based on Blum-Blum-Shub sequences of length 1,000,000. This graph showed that there is no large redundancy among our tests.

The correlation matrix formed from the $z_1, \ldots, x_m$ was constructed via a statistical software application (SAS). The same conclusion was supported by the structure of these matrices. The degree of duplication among the tests seems to be very small.

5. USER'S GUIDE

This section describes the set-up and proper usage of the statistical tests developed by NIST that are available in the NIST test code. Descriptions of the algorithms and data structures that were utilized are included in this section.

5.1 About the Package

This toolbox was specifically designed for individuals interested in conducting statistical testing of cryptographic (P)RNGs. Several implementations of PRNGs utilized during the development phase of the project have also been included.

Caveat: The test code was developed using a SUN workstation under the Solaris operating system. No guarantee is made regarding the compilation and execution of the PRNG implementations on other platforms. For this reason, a switch has been incorporated into the source codes to disable the inclusion of the PRNGs. The flag ***INCLUDE_GENERATORS*** *can be found in the defs.h header file.*

This package will address the problem of evaluating (P)RNGs for randomness. It will be useful in:

- identifying (P)RNGs which produce weak (or patterned) binary sequences,
- designing new (P)RNGs,
- verifying that the implementations of (P)RNGs are correct,
- studying (P)RNGs described in standards, and
- investigating the degree of randomness by currently used (P)RNGs.

The objectives during the development of the NIST statistical test suite included:

- Platform Independence: The source code was written in ANSI C. However, some modification may have to be made, depending on the target platform and the compiler.
- Flexibility: The user may freely introduce their own math software routines.
- Extensibility: New statistical tests can easily be incorporated.
- Versatility: The test suite is useful in performing tests for PRNGs, RNGs and cipher algorithms.
- Portability: With minor modifications, source code may be ported to different platforms. The NIST source code was ported onto the SGI Origin, and a 200 MHz PC using the Microsoft Visual C++ 6.0 development environment.

- Orthogonality: A diverse set of tests is provided.
- Efficiency: Linear time or space algorithms were utilized whenever possible.

5.2 System Requirements

This software package was developed on a SUN workstation under the Solaris operating system. All of the source code was written in ANSI C. Source code porting activities were successful for the SGI Origin (IRIX 6.5 with the SGI C compiler) and a desktop computer (IBM PC under Windows 98 and Microsoft C++ 6.0).

In practice, minor modifications will have to be introduced during the porting process in order to ensure the correct interpretation of tests. In the event that a user wishes to compile and execute the code on a different platform, sample data and the corresponding results for each of the statistical tests have been provided. In this manner, the user will be able to gain confidence that the ported statistical test suite is functioning properly. For additional details see Appendix C.

For the majority of the statistical tests, memory must be allocated dynamically in order to proceed. In the event that workspace cannot be provided, the statistical test returns a diagnostic message.

5.3 How to Get Started

To setup a copy of the NIST test code on a workstation, follow the instructions below.

- Copy the *sts.tar* file into the root directory. Use the instruction, **tar -xvf sts.tar**, to unbundle the source code.

- Several files and subdirectories should have been created. The eight subdirectories include **data/, experiments/, generators/, include/, obj/, src/** and **templates/.** The four files include *assess*, *grid*, *makefile*, and *stats*.

- The **data/** subdirectory is reserved for pre-existing RNG data files that are under investigation. Currently, two formats are supported, i.e., data files consisting of ASCII zeroes and ones, and binary formatted hexadecimal character strings.

- The **experiments/** subdirectory will be the repository of the empirical results for RNG data. Several subdirectories should be contained in it. These include **AlgorithmTesting/, BBS/, CCG/, G-SHA-1/, LCG/, MODEXP/, MS/, QCG1/, QCG2/,** and **XOR/.** All but the first of these subdirectories is meant to store the results for the corresponding PRNG. The **AlgorithmTesting/** subdirectory is the default subdirectory for empirical results corresponding to RNG data stored in the **data/** subdirectory.

- The **generators/** subdirectory contains the source codes for nine pseudo-

random number generators. These include Blum-Blum-Shub, Cubic Congruential Generator, the FIPS 186 one way function based on SHA-1 (G-SHA-1), Linear Congruential Generator, Modular Exponentiation, Micali-Schnorr, Quadratic Congruential Generator I and II, and Exclusive OR. Code for the ANSI X9.17 generator and the FIPS 186 one way function based on DES (G-DES) were removed from the package because of possible export issues. User defined PRNGs should be copied into this subdirectory, with the corresponding modifications to the *makefile*, *utilities1.c*, *defs.h*, and *proto.h* files.

- The **include/** subdirectory contains the header files for the statistical tests, pseudo-random number generators, and associated routines.

- The **obj/** subdirectory contains the object files corresponding to the statistical tests, pseudo random number generators and other associated routines.

- The **src/** subdirectory contains the source codes for each of the statistical tests.

- The **templates/** subdirectory contains a series of non-periodic templates for varying block sizes that are utilized by the **NonOverlapping Templates** statistical test.

- User prescribed modifications may be introduced in several files. This will be discussed subsequently in Section 5.5.2 and Appendix B.

- Edit the *makefile*. Modify the following lines:
  (a) **CC** (your ANSI C compiler)
  (b) **ROOTDIR** (the root directory that was prescribed earlier in the process, e.g., **rng/**)

- Now execute *Makefile*. An executable file named *assess* should appear in the project directory.

- The data may now be evaluated. Type the following: **assess <sequenceLength>,** e.g., assess 1000000.

Follow the menu prompts. The files *stats* and *grid* correspond respectively to the logs of the per sequence frequency of zeroes and ones and the 0-1 matrix of fail/pass assignments for each individual sequence and each individual statistical test.

5.4 Data Input and Output of Empirical Results

5.4.1 Data Input

Data input may be supplied in one of two ways. If the user has a stand-alone program or hardware device which implements a RNG, the user may want to construct as many files of arbitrary length as desired. Files should contain binary sequences stored as either ASCII characters consisting of zeroes and ones, or as hexadecimal characters stored in binary format. These files can then be independently examined by the NIST Statistical Test Suite.

In the event that storage space is a problem, the user may want to modify the reference implementation and plug-in their implementation of the PRNG under evaluation. The bit streams will be stored directly in the ***epsilon*** data structure, which contains binary sequences.

5.4.2 Output of Empirical Results

The output logs of empirical results will be stored in two files, *stats* and *results*, that correspond respectively to the computational information, e.g., test statistics, intermediate parameters, and *P-values* for each statistical test applied to a data set.

If these files are not properly created, then it is most probably due to the inability to open the files for output. See Appendix J for further details.

5.4.3 Test Data Files

Five sample files have been created and are contained in the **data/** subdirectory. Four of these files correspond to the Mathematica[9] generated binary expansion of several classical numbers for over 1,000,000 bits. These files are *data.e*, *data.pi*, *data.sqrt2*, and *data.sqrt3*. The Mathematica program used in creating these files can be found in Appendix E. A fifth file, *data.sha1*, was constructed utilizing the SHA-1 hash function.

5.5 Program Layout

The test suite package has been decomposed into a series of modules which include the: statistical tests, (pseudo)random number generators, empirical results (hierarchical) directories, and data.

The three primary components of the NIST test suite are the statistical tests, the underlying mathematical software, and the pseudo random number generators under investigation. Other

[9] **Mathematica**, Stephen Wolfram's Computer Algebra System, http://www.mathematica.com.

components include the source code library files, the data directory and the hierarchical directory (**experiments**/) containing the sample data files and empirical result logs, respectively.

5.5.1 General Program

The NIST test suite contains sixteen tests which will be useful in studying and evaluating the binary sequences produced by random and pseudo random number generators. As in previous work in this field, statistical tests must be devised which, under some hypothesized distribution, employ a particular test statistic, such as the number of runs of ones or the number of times a pattern appears in a bit stream. The majority of the tests in the test suite either (1) examine the distribution of zeroes and ones in some fashion, (2) study the harmonics of the bit stream utilizing spectral methods, or (3) attempt to detect patterns via some generalized pattern matching technique on the basis of probability theory or information theory.

5.5.2 Implementation Details

In practice, any number of problems can arise if the user executes this software in unchartered domains. It is plausible that sequence lengths well beyond the testing procedure (i.e., on the order of 10^6) may be chosen. If memory is available, there should not be any reason why the software should fail. However, in many instances, user defined limits are prescribed for data structures and workspace. Under these conditions, it may be necessary to increase certain parameters, such as the MAXNUMOFTEMPLATES and the MAXNUMBEROFCYCLES. Several parameters that may be modified by a user are listed in Table 3.

The parameter *ALPHA* denotes the significance level that determines the region of acceptance and rejection. NIST recommends that *ALPHA* be in the range [0.001, 0.01].

The parameter *ITMAX* is utilized by the special functions; it represents the upper bound on the maximum number of iterations allowed for iterative computations.

The parameter *KAPPA* is utilized by the *gcf* and *gser* routines defined in the *special-functions.c* file. It represents the desired accuracy for the incomplete gamma function computations.

The parameter *MAXNUMOFTEMPLATES* indicates the maximum number of non-periodic templates that may be executed by the Nonoverlapping Template Matchings test. For templates of size $m = 9$, up to 148 possible non-periodic templates may be applied.

The parameters *NUMOFTESTS* and *NUMOFGENERATORS* correspond to the maximum number of tests that may be defined in the test suite, and the maximum number of generators specified in the test suite, respectively.

Lastly, the *MAXNUMBEROFCYCLES* represents the maximum number of cycles that NIST anticipates in any particular binary sequence.

Table 3. User Prescribed Statistical Test Parameters

Source Code Parameter	Default Parameter	Description/Definition
ALPHA	0.01	Significance Level
ITMAX	2000000	Max number of iterations
KAPPA	3e-15	Desired accuracy for incgam
MAXNUMOFTEMPLATES	40	Non-overlapping Templates Test
NUMOFTESTS	16	Max number of tests
NUMOFGENERATORS	12	Max number of PRNGs
MAXNUMBEROFCYCLES	40000	Max number of cycles

5.5.3 Description of the Test Code

5.5.3.1 Global Data Structures

Binary sequences are stored in the **epsilon** data structure. To efficiently store this information, a **bit field** structure was introduced. This is a C structure defined to strictly utilize a single bit to store a zero or a one. In essence, the bit field structure utilizes the minimum amount of storage necessary to hold the information that will be manipulated by the statistical tests. It is flexible enough to allow easy manipulation by accessing individual bits via an index specification.

5.5.3.2 Special Data Structures

For many of the tests, efficiency is desired in both time and space. A binary tree data structure is used for this purpose. In this case, the binary tree is implemented as an array, whose root node serves no particular purpose. The binary tree is used in several different ways. One way is as a Boolean structure where each individual node represents either a zero or a one, but whose content indicates the absence or presence of an individual bit. Parsing this tree indicates the presence or absence of a word of fixed length. In addition, the binary tree structure is used as an efficient means to tabulate the frequency of all 2^m words of length m in a finite binary sequence.

This data structure is also employed in the construction of the dictionary required in the Lempel-Ziv coding scheme. The restriction in this case, however, is that if the stream isn't equidistributed (i.e., is very patterned), then the Lempel-Ziv test may break down. This is due to the unbalanced binary tree[10] which may ensue. In this case, the procedure is halted by the test suite and a warning statement is returned.

[10] The binary tree will be unbalanced due to the presence of too many words exceeding $log_2 n$, where n is the sequence length.

5.5.3.3 Mathematical Software

Special functions required by the test suite are the *incomplete gamma function* and the *complementary error function*. The *cumulative distribution function*, also known as the *standard normal function*, is also required, but it can be expressed in terms of the error function.

One of the initial concerns regarding the development of the reference implementation was the dependencies that were required in order to gain reliable mathematical software for the special functions required in the statistical tests. To resolve this matter, the test suite makes use of the following libraries:

The Fast Fourier Transform routine was obtained at http://www.netlib.org/fftpack/fft.c.
The normal function utilized in this code was expressed in terms of the error function.

Standard Normal (Cumulative Distribution) Function

$$\Phi(z) = \frac{1}{\sqrt{2p}} \int_{-\infty}^{z} e^{-u^2/2} du$$

The complementary error function (*erfc*) utilized in the package is the ANSI C function contained in the *math.h* header file and its corresponding mathematical library. This library should be included during compilation.

Complementary Error Function

$$erfc(z) = \frac{2}{\sqrt{p}} \int_{z}^{\infty} e^{-u^2} du$$

The incomplete gamma function is based on an approximation formula whose origin is described in the *Handbook of Applied Mathematical Functions* [1] and in the *Numerical Recipes in C* book [6]. Depending on the values of its parameters *a* and *x*, the incomplete gamma function may be approximated using either a continued fraction development or a series development.

Gamma Function

$$\Gamma(z) = \int_0^{\infty} t^{z-1} e^{-t} dt$$

Incomplete Gamma Function

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

where $P(a,0) = 0$ and $P(a,¥) = 1$.

Incomplete Gamma Function

$$Q(a,x) \equiv 1 - P(a,x) \equiv \frac{\Gamma(a,x)}{\Gamma(a)} \equiv \frac{1}{\Gamma(a)} \int_x^\infty e^{-t} t^{a-1} dt$$

where Q(a,0) = 1 and Q(a,∞) = 0.

NIST has chosen to use the **Cephes** C language special functions math library in the test software. Cerphes may be found at http://people.ne.mediaone.net/moshier/index.html#Cephes or on the GAMS server at http://math.nist.gov/cgi-bin/gams-serve/list-module components/ CEPHES/CPROB/13192.html. The specific functions that are utilized are *igamc* (for the complementary incomplete gamma function) and *lgam* (for the logarithmic gamma function).

5.6 Running the Test Code

A sample NIST Statistical Test Suite monolog is described below. Note: In this section bold items indicate input.

In order to invoke the NIST statistical test suite, type **assess**, followed by the desired bit stream length, *n*. For example, **assess 100000**. A series of menu prompts will be displayed in order to select the data to be analyzed and the statistical tests to be applied. The first screen appears as follows:

```
GENERATOR OPTIONS

[00] Input File                    [01] G Using SHA-1
[02] Linear Congruential           [03] Blum-Blum-Shub
[04] Micali-Schnorr                [05] Modular Exponentiation
[06] Quadratic Congruential I      [07] Quadratic Congruential II
[08] Cubic Congruential            [09] Exclusive OR

OPTION ----> 0
User Prescribed Input File: data/data.pi
```

Once the user has prescribed a particular data set or PRNG, the statistical tests to be applied must be selected. The following screen is displayed:

```
                    STATISTICAL TESTS
[01] Frequency                          [02] Block Frequency
[03] Cumulative Sums                    [04] Runs
[05] Longest Runs of Ones               [06] Rank
[07] Spectral - Discrete Fourier Transform [08] Nonperiodic Template Matchings
[09] Overlapping Template Matchings     [10] Universal Statistical
[11] Approximate Entropy                [12] Random Excursions
[13] Random Excursions Variant          [14] Serial
[15] Lempel-Ziv Complexity              [16] Linear Complexity
                    INSTRUCTIONS
        Enter 0 if you DO NOT want to apply all of the
        statistical tests to each sequence and 1 if you DO.
        Enter Choice: 0
```

In this case, 0 has been selected to indicate interest in applying a subset of the available statistical tests. The following screen is then displayed.

```
                    INSTRUCTIONS
        Enter a 0 or 1 to indicate whether or not the numbered
        statistical test should be applied to each sequence. For
        example, 1111111111111111 applies every test to each
        sequence.
                    1234567891111111
                            0123456
                    0000000010000000
```

As shown above, the only test applied was number 9, the Nonoverlapping templates test. A query for the desired sample size is then made.

```
How many bit streams should be generated? 10
```

Ten sequences will be parsed using the *data.pi* file. Since a file was selected as the data specification mode, a subsequent query is made regarding the data representation. The user must specify whether the file consists of bits stored in ASCII format or hexadecimal strings stored in binary format.

[0] BITS IN ASCII FORMAT [1] HEX DIGITS IN BINARY FORMAT Select input mode: **0**

Since the data consists of a long sequence of zeroes and ones, 0 was chosen. Given all necessary input parameters the test suite proceeds to analyze the sequences.

Statistical Testing In Progress.....

During the execution of the statistical tests, two log files located under the **rng/** directory are updated. One file is the *stats* file, the other is the *grid* file. The former contains the distribution of zeroes and ones for each binary sequence, whereas the latter contains a binary matrix of values corresponding to whether or not sequence *i* passed statistical test *j*. Once the testing process is complete, the empirical results can be found in the **experiments/** subdirectory.

Statistical Testing Complete!!!!!!!!!!!!

Upon completion, an in-depth analysis is performed utilizing a MATLAB[11] script written to simplify the analysis of empirical results. Two types of analyses are conducted. One type examines the proportion of sequences that pass a statistical test. The other type examines the distribution of the *P-values* for each statistical test. More details are supplied in the Section 4.2.

5.7 Interpretation of Results

An analytical routine has been included to facilitate interpretation of the results. A file **finalAnalysisReport** is generated when statistical testing is complete. The report contains a summary of empirical results. The results are represented via a table with *p* rows and *q* columns. The number of rows, *p,* corresponds to the number of statistical tests applied. The number of columns, $q = 13$, are distributed as follows: columns 1-10 correspond to the frequency of P-values[12], column 11 is the P-value that arises via the application of a chi-square test[13], column 12 is the proportion of binary sequences that passed, and the 13th column is the corresponding statistical test. An example is shown in Figure 6.

[11] See Section 1.2, Definitions and Abbreviations.
[12] The unit interval has been divided into ten discrete bins.
[13] I n order to assess the uniformity of P-values in the ith statistical test.

```
------------------------------------------------------------------------------
RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES
------------------------------------------------------------------------------

------------------------------------------------------------------------------
 C1  C2  C3  C4  C5  C6  C7  C8  C9 C10  P-VALUE  PROPORTION  STATISTICAL TEST
------------------------------------------------------------------------------
  6  12   9  12   8   7   8  12  15  11  0.616305   0.9900    Frequency
 11  11  12   6  10   9   8   9  17   7  0.474986   0.9900    Cusum
  6  10   8  14  16  10  10   6   5  15  0.129620   0.9900    Cusum
  7   9   9  11  11  11   8  12  12  10  0.978072   0.9900    Serial
 13   6  13  15   9   7   3  11  13  10  0.171867   0.9600    Serial

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
The minimum pass rate for each statistical test with the exception of the random
excursion (variant) test is approximately = 0.960150 for a sample
size = 100 binary sequences.

For further guidelines construct a probability table using the MAPLE program
provided in the addendum section of the documentation.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Figure 6: Depiction of the Final Analysis Report

APPENDIX A: RANK COMPUTATION FOR BINARY MATRICES

Apply elementary row operations where the addition operator is taken to be the exclusive-OR operation. The matrices are reduced to upper triangular form using forward row operations, and the operation is repeated in reverse in order using backward row operations in order to arrive at a matrix in diagonal form. The rank is then taken to be the number of nonzero rows in the resulting Gaussian reduced matrix.

<u>Forward Application of Elementary Row Operations</u>:

Let each element in the m by m matrix be designated as $a_{i,j}$

1. Set $i = 1$

2. If element $a_{i,i} = 0$ (i.e., the element on the diagonal ≠ 1), then swap all elements in the ith row with all elements in the next row that contains a one in the ith column (i.e., this row is the kth row, where $i < k$ £ m) . If no row contains a "1" in this position, go to step 4.

3. If element $a_{i,i} = 1$, then if any subsequent row contains a "1" in the ith column, replace each element in that row with the exclusive-OR of that element and the corresponding element in the ith row.

   a. Set $row = i + 1$
   b. Set $col=i$.
   c. If $a_{row,col} = 0$, then go to step 3g.
   d. $a_{row,col} = a_{row,col} \oplus a_{i,col}$
   e. If $col=m$, then go to step 3g.
   f. $col=col+1$; go to step 3d.
   g. If $row = m$, then go to step 4.
   h. $row=row+1$; go to step 3b.

4. If $i<m-1$, then $i=i+1$; go to step 2.

5. Forward row operations completed.

<u>Backward Application of Elementary Row Operations</u>:

1. Set $i=m$.

2. If element $a_{i,i} = 0$ (i.e., the element on the diagonal ≠ 1), then swap all elements in the ith row with all elements in the next row that contains a one in the ith column (i.e., this row is the kth row, where $1 \pounds k < i$) . If no row contains a "1" in this position, go to step 4.

3. If element $a_{i,i} = 1$, then if any preceding row contains a "1" in the ith column, replace each element in that row with the exclusive-OR of that element and the corresponding element in the ith row.

   a. Set $row = i - 1$
   b. Set $col=i$.
   c. If $a_{row,col} = 0$, then go to step 3g.
   d. $a_{row,col} = a_{row,col} \oplus a_{i,col}$
   e. If $col= 1$, then go to step 3g.
   f. $col=col - 1$; go to step 3d.
   g. If $row = 1$, then go to step 4.
   h. $row=row-1$; go to step 3b.

4. If $i>2$, then $i=i-1$ and go to step 2.

5. Backward row operation complete.

6. The rank of the matrix = the number of non-zero rows.

<u>Example of Forward Row Operations</u>:

A	1 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 1 0 1 0 1 0 0 0 1 0 1 1 0 0 0 0 1 0	The original matrix.
B	1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 0 1 0 0 0 1 0 1 1 0 0 0 0 1 0	Since $a_{1,1} = 1$ and rows 3 and 4 contain a 1 in the first column (see the original matrix), rows 3 and 4 are replaced by the exclusive-OR of that row and row 1.
C	1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 0 1 0 0 0 1 0 1 1	Since $a_{2,2}$ ¹ 1 and no other row contains a "1" in this column (see B), the matrix is not altered.

	0 0 0 0 1 0	
D	1 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 0 0 0 0 1 0 0 1 0 1 1 0 0 0 0 1 0	Since $a_{3,3}$ ¹ *1*, but the 4th row contains a "1" in the 3rd column (see B or C), the two rows are switched.
E	1 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 0	Since row 5 contains a "1" in the 3rd column (see D), row 5 is replaced by the exclusive-OR of row 1 and row 5.
F	1 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 0	Since $a_{4,4}$ ¹ *1* and no other row contains a "1" in this column (see E), the matrix is not altered.
G	1 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 1	Since $a_{5,5}$ ¹ *1*, but row 6 contains a 1 in column 5 (see F), the two rows are switched. Since no row below this contains a "1" in the 5th column, the end of the forward process is complete.

<u>The Subsequent Backward Row Operations</u>:

H	1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1	Since $a_{6,6} = 1$ and rows 2 and 4 contain ones in the 6th column (see G), rows 2 and 4 are replaced by the exclusive-OR of that row and row 6.
I	1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0	Since $a_{5,5} = 1$and row 3 contains a one in the 5th column (see H), row 3 is replaced by the exclusive-OR or row 3 and row 5.

	0 0 0 0 0 1	
J	1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1	Since $a_{4,4}$ ¹ 1 and no other row has a one in column 4, the matrix is not altered.
K	1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1	Since $a_{3,3} = 1$, but no other row has a one in column 3, the matrix is not altered.
L	1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1	Since $a_{2,2}$ ¹ 1 and no other row has a one in column 2, the matrix is not altered, and he process is complete.

Since the final form of the matrix has 4 non-zero rows, the rank of the matrix is 4.

APPENDIX B: SOURCE CODE

Filename: *defs.h*

Debugging Aides:

```
1.    #define FREQUENCY                          1
2.    #define BLOCK_FREQUENCY                    1
3.    #define CUSUM                              1
4.    #define RUNS                               1
5.    #define LONG_RUNS                          1
6.    #define RANK                               1
7.    #define MATRICES                           0
8.    #define DFT                                1
9.    #define APERIODIC_TEMPLATES                1
10.   #define PERIODIC_TEMPLATES                 1
11.   #define UNIVERSAL                          1
12.   #define APEN                               1
13.   #define SERIAL                             1
14.   #define RANDOM_EXCURSIONS                  1
15.   #define RANDOM_EXCURSIONS_VARIANT          1
16.   #define LEMPEL_ZIV                         1
17.   #define LINEAR_COMPLEXITY                  1
18.   #define DISPLAY_OUTPUT_CHANNELS            1
19.   #define PARTITION                          0
```

Note: For debugging purposes, switches were introduced to display/not display intermediate computations for each statistical test. A one denotes true, i.e., show intermediate results; a zero denotes false, i.e., do not show intermediate results.

Filename: *defs.h*

Statistical Testing Alternatives:

```
1.    #define INCLUDE_GENERATORS                 1
2.    #define LONG_RUNS_CASE_8                   0
3.    #define LONG_RUNS_CASE_128                 0
4.    #define LONG_RUNS_CASE_10000               1
5.    #define SAVE_FFT_PARAMETERS                0
6.    #define SAVE_APEN_PARAMETERS               0
7.    #define SAVE_RANDOM_EXCURSION_PARAMETERS   1
8.    #define SEQ_LENGTH_STEP_INCREMENTS         5000
```

Note: Statistical testing alternatives have been incorporated into the test suite using switches.

Line 1 refers to the inclusion (or exclusion) of the pseudo-random number generators contained in the NIST test suite during compilation. The ability to enable or disable this function was

introduced under the realization that underlying libraries may not port easily to different platforms. The user can disable the sample generators and should be able to compile the statistical tests.

Lines 2-4 refer to different probability values that have been included in the Long Runs of Ones Test. Since the statistical test partitions a sequence into sub-strings of varying length, the user has the freedom to select between several cases. The user should enable only one case and disable the other two cases.

Lines 5-7 refer to the ability to store intermediate parameter values to a file for the sake of constructing graphics. Line 5 will enable or disable the storage of the Fourier points and corresponding moduli into the files, *fourierPoints* and *magnitude*, respectively. Line 6 will enable or disable the storage of the sequence length and approximate entropy value for varying sequence lengths into the files, *abscissaValues* and *ordinateValues*. Line 7 will enable or disable the storage of the number of cycles for each binary sequence into the file, *cycleInfo*.

Line 8 refers to the number of sequence length step increments to be taken during the generation and storage of the approximate entropy values in the file, *ordinateValues*.

Filename: *defs.h*

Global Constants:

1.	#define ALPHA	0.01
2.	#define MAXNUMOFTEMPLATES	148
3.	#define NUMOFTESTS	16
4.	#define NUMOFGENERATORS	9
5.	#define MAXNUMBEROFCYCLES	40000
6.	#define MAXFILESPERMITTEDFORPARTITION	400

Lines 1-6 correspond to test suite parameters that have been preset. Under various conditions, the user may decide to modify them.

Line 1 refers to the significance level. It is recommended that the user select the level in the range [0.001,0.01].

Line 2 refers to the maximum number of templates that may be used in the Nonoverlapping Template Matching test.

Line 3 refers to the maximum number of tests that is supported in the test suite. If the user adds additional tests, this parameter should be incremented.

Line 4 refers to the maximum number of generators that is supported in the package. If the user adds additional generators, this parameter should be incremented.

Line 5 refers to the maximum number of expected cycles in the random excursions test. If this number is insufficient, the user may increase the parameter appropriately.

Line 6 refers to the maximum number of files which may be decomposed by the **partitionResultFile** routine. This routine is applied only for specific tests where more than one *P-value* is produced per sequence. This routine decomposes the corresponding *results* file into separate files, *data001*, *data002*, *data003*, …

APPENDIX C: EMPIRICAL RESULTS FOR SAMPLE DATA

The user is urged to validate that the statistical test suite is operating properly. For this reason, five sample files have been provided. These five files are: (1) *data.pi*, (2) *data.e*, (3) *data.sha1*, (4) *data.sqrt2*, and (5) *data.sqrt3*. For each data file, all of the statistical tests were applied, and the results recorded in the following tables. The Block Frequency, Long Runs of Ones, Non-overlapping Template Matching, Overlapping Template Matching, Universal, Approximate Entropy, Linear Complexity and Serial tests require user prescribed input parameters. The exact values used in these examples has been included in parenthesis beside the name of the statistical test. In the case of the random excursions and random excursions variant tests, only one of the possible 8 and 18 *P-values*, respectively, has been reported.

Example #1: The binary expansion of p

Statistical Test	P-value
Frequency	0.578211
Block Frequency ($m = 100$)	0.014444
Cusum-Forward	0.628308
Cusum-Reverse	0.663369
Runs	0.419268
Long Runs of Ones ($M = 10000$)	0.024390
Rank	0.083553
Spectral DFT	0.012947
Non-overlapping Templates ($m = 9, B = 000000001$)	0.165757
Overlapping Templates ($m = 9$)	0.296897
Universal ($L = 7, Q = 1280$)	0.669012
Approximate Entropy ($m = 5$)	0.634457
Random Excursions ($x = +1$)	0.844143
Random Excursions Variant ($x = -1$)	0.760966
Lempel Ziv Complexity	0.311714
Linear Complexity ($M = 500$)	0.255475
Serial ($m = 5,\ \nabla\Psi_m^2$)	0.583812

Example #2: The binary expansion of e

Statistical Test	P-value
Frequency	0.953749
Block Frequency (*m = 100*)	0.619340
Cusum-Forward	0.669887
Cusum-Reverse	0.724266
Runs	0.561917
Long Runs of Ones (*M = 10000*)	0.718945
Rank	0.306156
Spectral DFT	0.443864
NonOverlapping Templates (*m = 9, B = 000000001*)	0.078790
Overlapping Templates (*m = 9*)	0.110434
Universal (*L = 7, Q = 1280*)	0.282568
Approximate Entropy (*m = 5*)	0.361688
Random Excursions (*x = +1*)	0.778616
Random Excursions Variant (*x = -1*)	0.826009
Lempel Ziv Complexity	0.000322
Linear Complexity (*M = 500*)	0.826335
Serial ($m = 5,\ \nabla\Psi_m^2$)	0.225783

Example #3: A G-SHA-1 binary sequence

Statistical Test	P-value
Frequency	0.604458
Block Frequency (*m = 100*)	0.833026
Cusum-Forward	0.451231
Cusum-Reverse	0.550134
Runs	0.309757
Long Runs of Ones (*M = 10000*)	0.657812
Rank	0.577829
Spectral DFT	0.086702
NonOverlapping Templates (*m = 9, B = 000000001*)	0.496601
Overlapping Templates (*m = 9*)	0.339426
Universal (*L = 7, Q = 1280*)	0.411079
Approximate Entropy (*m = 5*)	0.731449
Random Excursions (*x = +1*)	0.000000
Random Excursions Variant (*x = -1*)	0.000000
Lempel Ziv Complexity	0.398475
Linear Complexity (*M = 500*)	0.309412
Serial ($m = 5,\ \nabla\Psi_m^2$)	0.742275

Example #4: The binary expansion of $\sqrt{2}$

Statistical Test	P-value
Frequency	0.811881
Block Frequency (*m = 100*)	0.289410
Cusum-Forward	0.879009
Cusum-Reverse	0.957206
Runs	0.313427
Long Runs of Ones (*M = 10000*)	0.012117
Rank	0.823810
Spectral DFT	0.267174
NonOverlapping Templates (*m = 9, B = 000000001*)	0.569461
Overlapping Templates (*m = 9*)	0.791982
Universal (*L = 7, Q = 1280*)	0.130805
Approximate Entropy (*m = 5*)	0.853227
Random Excursions (*x = +1*)	0.216235
Random Excursions Variant (*x = -1*)	0.566118
Lempel Ziv Complexity	0.949310
Linear Complexity (*M = 500*)	0.317127
Serial ($m = 5,\ \nabla\Psi_m^2$)	0.873914

Example #5: The binary expansion of $\sqrt{3}$

Statistical Test	P-value
Frequency	0.610051
Block Frequency (*m = 100*)	0.573925
Cusum-Forward	0.917121
Cusum-Reverse	0.689519
Runs	0.261123
Long Runs of Ones (*M = 10000*)	0.446726
Rank	0.314498
Spectral DFT	0.463412
NonOverlapping Templates (*m = 9, B = 000000001*)	0.532235
Overlapping Templates (*m = 9*)	0.082716
Universal (*L = 7, Q = 1280*)	0.165981
Approximate Entropy (*m = 5*)	0.404616
Random Excursions (*x = +1*)	0.783283
Random Excursions Variant (*x = -1*)	0.155066
Lempel Ziv Complexity	0.989651
Linear Complexity (*M = 500*)	0.346469
Serial ($m = 5,\ \nabla\Psi_m^2$)	0.100780

APPENDIX D: CONSTRUCTION OF APERIODIC TEMPLATES

For the purposes of executing the Non-overlapping Template Matching statistical test, all 2^m m-bit binary sequences which are aperiodic were pre-computed. These templates, or patterns, were stored in a file for $m = 2$ to $m = 21$. The ANSI-C program utilized in finding these templates is provided below. By modifying the parameter M, the template library corresponding to the template can be constructed. This parameter value should not exceed B, since the **dec2bin** conversion routine will not operate correctly. Conceivably, this source code can be easily modified to construct arbitrary 2^m m-bit binary sequences for larger m.

```
#include <stdio.h>
#include <math.h>

#define B 32
#define M 6

int *A;
static long nonPeriodic;
unsigned displayBits(FILE*, long, long);

int main()
{
  FILE *fp1, *fp2;
  long i, j, count, num;

  A = (unsigned*) calloc(B,sizeof(unsigned));
  fp1 = fopen("template", "w");
  fp2 = fopen("dataInfo", "a");
  num = pow(2,M);
  count = log(num)/log(2);
  nonPeriodic = 0;
  for(i = 1; i < num; i++) displayBits(fp1, i,count);
  fprintf(fp2,"M = %d\n", M);
  fprintf(fp2,"# of nonperiodic templates  = %u\n",
          nonPeriodic);
  fprintf(fp2,"# of all possible templates = %u\n", num);
  fprintf(fp2,"{# nonperiodic}/{# templates} = %f\n",
          (double)nonPeriodic/num);
  fprintf(fp2,"==========================================");
  fprintf(fp2,"===============\n");
  fclose(fp1);
  fclose(fp2);
  free(A);
  return 0;
}
```

```
void displayBits(FILE* fp, long value, long count)
{
  int i, j, match, c, displayMask = 1  << (B-1);
  for(i = 0; i < B; i++) A[i] = 0;
  for(c = 1; c <= B; c++) {
     if (value & displayMask)

        A[c-1] = 1;
     else
        A[c-1] = 0;
     value <<= 1;
  }
  for(i = 1; i < count; i++) {
     match = 1;
     if ((A[B-count]!= A[B-1]) &&
        ((A[B-count]!= A[B-2])||(A[B-count+1] != A[B-1]))) {
        for(c = B-count; c <= (B-1)-i; c++) {
           if (A[c] != A[c+i]) {
              match = 0;
              break;
           }
        }
     }
     if (match) {
        /* printf("\nPERIODIC TEMPLATE: SHIFT = %d\n",i); */
        break;
     }
  }
  if (!match) {
     for(c = B-count; c < (B-1); c++) fprintf(fp,"%u",A[c]);
     fprintf(fp,"%u\n", A[B-1]);
     nonPeriodic++;
  }
  return;
}
```

APPENDIX E: GENERATION OF THE BINARY EXPANSION OF IRRATIONAL NUMBERS

The sample Mathematica program utilized in constructing four sample files is shown below.

Mathematica Program

```
(*************************************************************)
(* Purpose: Converts num to its decimal expansion using   *)
(*          its binary representation.                     *)
(*                                                         *)
(* Caution: The $MaxPrecision variable must be set to     *)
(*          the value of d.   By default, Mathematica      *)
(*          sets this to 50000, but this can be increased.*)
(*************************************************************)

BinExp[num_,d_] := Module[{n,L},
                    If[d > $MaxPrecision, $MaxPrecision = d];
                       n = N[num,d];
                       L = First[RealDigits[n,2]]
                      ];

SE = BinExp[E,302500];
Save["data.e",{SE}];

SP = BinExp[Pi,302500];
Save["data.pi",{SP}];

S2 = BinExp[Sqrt[2],302500];
Save["data.sqrt2",{S2}];

S3 = BinExp[Sqrt[3],302500];
Save["data.sqrt3",{S3}];
```

APPENDIX F: NUMERIC ALGORITHM ISSUES

For each binary sequence, an individual statistical test must produce at least one *P-value*. *P-values* are based on the evaluation of special functions, which must be as accurate as possible on the target platform. The log files produced by each statistical test, report *P-values* with six digits of precision, which should be sufficient. However, if greater precision is desired, modify the *printf* statements in each statistical test accordingly.

During the testing phase, NIST commonly evaluated sequences on the order 10^6; hence, results are based on this assumption. If the user wishes to choose longer sequence lengths, then be aware that numerical computations may be inaccurate[14] due to machine or algorithmic limitations. For further information on numerical analysis matters, see [6][15].

For the purposes of illustration, sample parameter values and corresponding special function values are shown in Table F.1 and Table F.2. Table F.1 compares the results for the incomplete gamma function for selected parameter values for *a* and *x*. The results are shown for Maple[16], Matlab[10], and the Numerical Recipe[17] routines. Recall that the definitions for the *gamma function* and the *incomplete gamma function* are defined, respectively, as:

$$\Gamma(z) = \int_0^\infty t^{z-1} e^{-t} dt$$

$$Q(a,x) = \frac{\Gamma(a,x)}{\Gamma(a)} = \frac{1}{\Gamma(a)} \int_x^\infty e^{-t} t^{a-1} dt,$$

where Q(a,0) = 1 and Q(a,∞) = 0.

Since the algorithm used in the test suite implementation of the *incomplete gamma function* is based on the numerical recipe codes, it is evident that the function is accurate to at least the seventh decimal place. For large values of *a*, the precision will degrade, as will confidence in the result (unless a computer algebra system is employed to ensure high precision computations).

Table F.2 compares the results for the *complementary error function* (see Section 5.3.3) for selected parameter values for *x*. The results are shown for ANSI C, Maple, and Matlab. Recall that the definition for the *complementary error function* is:

---

[14] According to the contents of the GNU C specifications at "*/usr/local/lib/gcc-lib/sparc-sun-solaris2.5.1/2.7.2.3/specs (*gcc version 2.7.2.3)," the *limits.h* header file on a SUN Ultra 1 workstation, the maximum number of digits of precision of a *double* is 15.

[15] Visit http://www.ulib.org/webRoot/Books/Numerical_Recipes/ or http://beta.ul.cs.cmu.edu/webRoot/Books/Numerical_Recipes/, particularly, Section 1.3 (*Error, Accuracy, and Stability*).

[16] See Section 1.2, Definitions and Abbreviations.

[17] The parameter values for *eps* and *itmax* were fixed at 3×10^{-15} and 2,000,000 respectively. Special function routines based on Numerical Recipe codes will be replaced by non-proprietary codes in the near future.

$$erfc(z) = \frac{2}{\sqrt{p}} \int_z^{\infty} e^{-u^2} du$$

Table F.1: Selected Input Parameters for the Incomplete Gamma Function

a = x = 600	*Q(a,x)*	a = x = 800	*Q(a,x)*
Maple	0.4945710333	Maple	0.4952983876
Matlab	0.4945710331	Matlab	0.4952983876
Test Suite	0.4945710331	Test Suite	0.4952983876
a = x = 1000	*Q(a,x)*	a = x = 10000	*Q(a,x)*
Maple	0.4957947559	Maple	0.4986701918
Matlab	0.4957947558	Matlab	0.4986701917
Test Suite	0.4957947558	Test Suite	0.4986701917
a = x = 100000	*Q(a,x)*	a = x = 1000000	*Q(a,x)*
Maple	0.4995794779	Maple	0.4998670192
Matlab	0.4995794779	Matlab	0.4998670196
Test Suite	0.4995794778	Test Suite	0.4998670205

Table F.2: Selected Input Parameters for the Complementary Error Function

x	*erfc(x)*		*x*	*erfc(x)*	
0.00	Test Suite	1.000000000000000	0.50	Test Suite	0.479500122186953
	Maple	1.000000000000000		Maple	0.479500122186950
	Matlab	1.000000000000000		Matlab	0.479500122186953
1.00	Test Suite	0.157299207050285	1.50	Test Suite	0.033894853524689
	Maple	0.157299207050280		Maple	0.033894853524690
	Matlab	0.157299207050285		Matlab	0.033894853524689
2.00	Test Suite	0.004677734981047	2.50	Test Suite	0.000406952017445
	Maple	0.004677734981050		Maple	0.000406952017440
	Matlab	0.004677734981047		Matlab	0.000406952017445
3.00	Test Suite	0.000022090496999	3.50	Test Suite	0.000000743098372
	Maple	0.000022090497000		Maple	0.000000743098370
	Matlab	0.000022090496999		Matlab	0.000000743098372

Thus, it is evident that the various math routines produce results that are sufficiently close to each other. The differences are negligible. To reduce the likelihood for obtaining an inaccurate *P-value* result, NIST has prescribed recommended input parameters.

APPENDIX G: HIERARCHICAL DIRECTORY STRUCTURE

rng/

makefile The NIST Statistical Test Suite *makefile*. This file is invoked in order to recompile the entire test suite, including PRNGs.

makefile2 The NIST Statistical Test Suite *makefile*. This file is invoked in order to recompile the NIST test suite without the PRNGs (Note: the PRNGs may not compile on all platforms without user intervention).

assess The NIST Statistical Test Suite executable file is called *assess*.

data/ This subdirectory contains the names of all data files to be analyzed. Sample files include the binary expansions to well known constants such as e, p, $\sqrt{2}$, and $\sqrt{3}$.

experiments/ This subdirectory contains the empirical result subdirectories for each RNG.

AlgorithmTesting/	BBS/
CCG/	G-SHA-1/
LCG/	MODEXP/
MS/	QCG1/
QCG2/	XOR/

For each subdirectory there is a set of nested directories, that is,

apen/	block-frequency/
cumulative-sums/	fft/
frequency/	lempel-ziv /
linear-complexity/	longest-run/
nonperiodic-templates/	overlapping-templates/
random-excursions/	random-excursions-variant/
rank/	runs/
serial/	universal/

For each nested directory there are two files created upon execution of an individual statistical test. The *results* file contains a *P-value* list for each binary sequence, and the *stats* file contains a list of statistical information for each binary sequence.

generators/ This subdirectory contains the source code for each PRNG. In the

event that the user is interested in evaluating their PRNG (online), their source code may, for example, be added as *generators4.c* in this directory, with additional changes made in the *utilities2.c* file and the *defs.h* file.

generators1.c: contains BBS, MS, LCG
generators2.c: contains ModExp, QCG1, QCG2, CCG1, XOR
generators3.c: contains G-SHA-1
sha.c : contains routines required by the G-SHA-1 PRNG.

grid This file contains bits which represent the acceptance or rejection of a particular sequence for each individual statistical test that is run. The *P-value* computed for the sequence is compared to the chosen significance level a.

include/ This subdirectory contains all of the header files that prescribe any global variables, constants, and data structures utilized in the reference implementation. In addition, the subdirectory contains all function declarations and prototypes.

cephes-protos.h	config.h
config2.h	defs.h
f2c.h	generators1.h
generators2.h	generators3.h
globals.h	lip.h
lippar.h	matrix.h
mconfig.h	mp.h
proto.h	sha.h
special-functions.h	utilities1.h
utilities2.h	

obj/ This subdirectory contains the object files corresponding to the source codes.

approximateEntropy.o	assess.o
cephes.o	cusum.o
dfft.o	discreteFourierTransform.o
frequency.o	functions.o
generators1.o	generators2.o
generators3.o	lempelZivCompression.o
linearComplexity.o	lip.o
matrix.o	mp.o
nonOverlappingTemplateMatchings.o	
overlappingTemplateMatchings.o	
randomExcursions.o	randomExcursionsVariant.o
rank.o	runs.o
sha.o	special-functions.o

universal.o utilities1.o utilities2.o

src/ This subdirectory contains the source codes for the statistical tests.

assess.c	:	The driver program for this package
dfft.c	:	The discrete fourier transform routine
cephes.c	:	Defines the incomplete gamma function
lip.c	:	Long integer precision library utilized by Pate Williams implementation of the Blum-Blum-Shub and Micali-Schnorr source codes
matrix.c	:	Source code for the determination of rank for binary matrices
mp.c	:	Multiprecision integer package
special-functions.c	:	Numerical routines for the handling of special functions
frequency.c	:	Frequency Test
blockFrequency.c	:	Block Frequency Test
cusum.c	:	Cumulative Sums Test
runs.c	:	Runs Test
longestRunOfOnes.c	:	Longest Run Of Ones Test
rank.c	:	Rank Test
discreteFourierTransform.c	:	Spectral Test
nonOverlappingTemplateMatchings.c	:	Nonoverlapping Template Matchings Test
OverlappingTemplateMatchings.c	:	Overlapping Template Matchings Test
universal.c	:	Universal Test
approximateEntropy.c	:	Approximate Entropy Test
randomExcursions.c	:	Random Excursions Test
randomExcursionsVariant.c	:	Random Excursions Variant Test
serial.c	:	Serial Test
lempelZivComplexity.c	:	Lempel-Ziv Complexity Test
linearComplexity.c	:	Linear Complexity Test
serial.c	:	Serial Test
utilities1.c	:	Utility functions…
utilities2.c	:	Utility functions…

stats This file contains the frequency distributions for individual sequences.

templates/ Non-Periodic Template Library:

This subdirectory contains the templates (or patterns) which are evaluated in the NonOverlapping Template Matching Test. The corresponding file is opened for the prescribed template block length *m*. Currently, the only options for which nonperiodic templates have been stored are those which lie in [2,21]. In the event that $m > 21$, the user must pre-compute the non-periodic templates.

template2	template3	template4	template5	template6
template7	template8	template9	template10	template11
template12	template13	template14	template15	template16
template17	template18	template19	template20	template21

APPENDIX H: VISUALIZATION APPROACHES

There are several visualization approaches that may be used to investigate the randomness of binary sequences. Three techniques involve the Discrete Fourier Transform, approximate entropy and the linear complexity profile.

(a) **Spectral - Discrete Fourier Transform (DFT) Plot**

Figure H.1 depicts the spectral components (i.e., the modulus of the DFT) obtained via application of the *Fast Fourier Transform* on a binary sequence (consisting of 5000 bits) extracted from the Blum-Blum-Shub pseudo-random number generator[18]. To demonstrate how the spectral test can detect periodic features in the binary sequence, every 10th bit was changed to a single one. To pass this test, no more than 5 % of the peaks should surpass the 95 % cutoff, (determined to be sqrt(3*5000) ≈ 122.4744871). Clearly, greater than 5 % of the peaks exceed the cutoff point in the figure. Thus, the binary sequence fails this test.

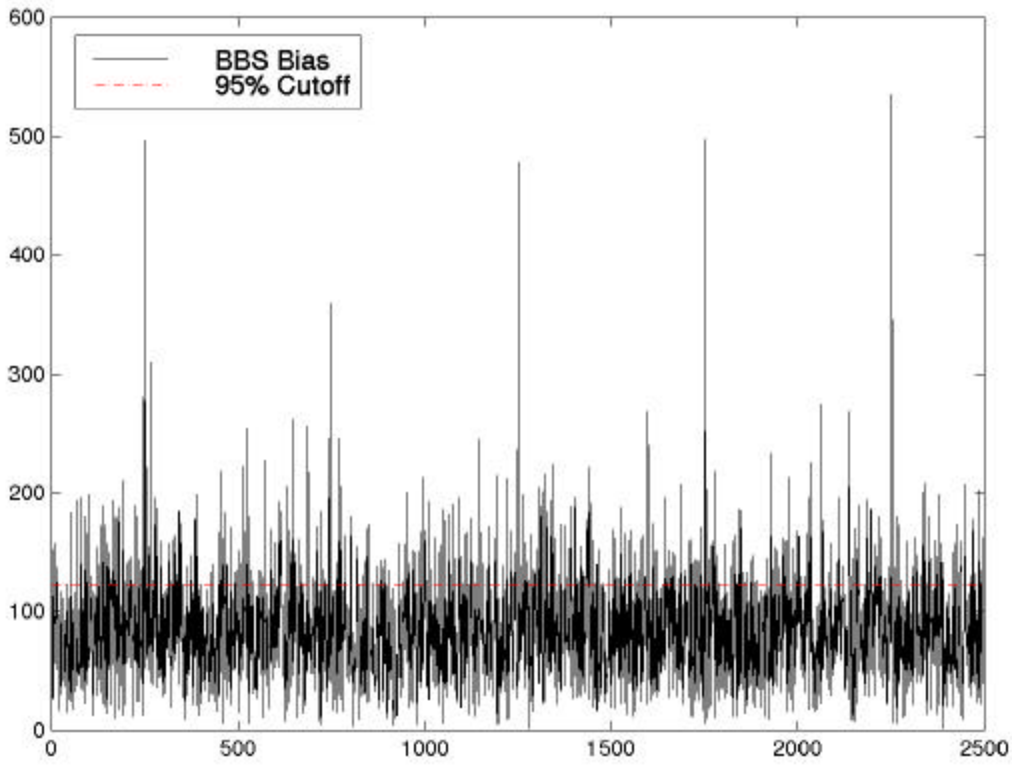


Figure H.1*:* Discrete Fourier Transform Plot

[18] The Blum-Blum-Shub pseudo random number generator, based on the intractability of the quadratic residuocity problem is described in the Handbook of Applied Cryptography, by Menezes, et. al.

(b) **Approximate Entropy (ApEn) Graph**

Figure H.2 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 the SHA-1 pseudo-random number generator. In theory, for an *n*-bit sequence, the maximum entropy value that can be attained is *ln (2) » 0.69314718*. 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-1[19] sequence. However, for larger block sizes, this is not the case.

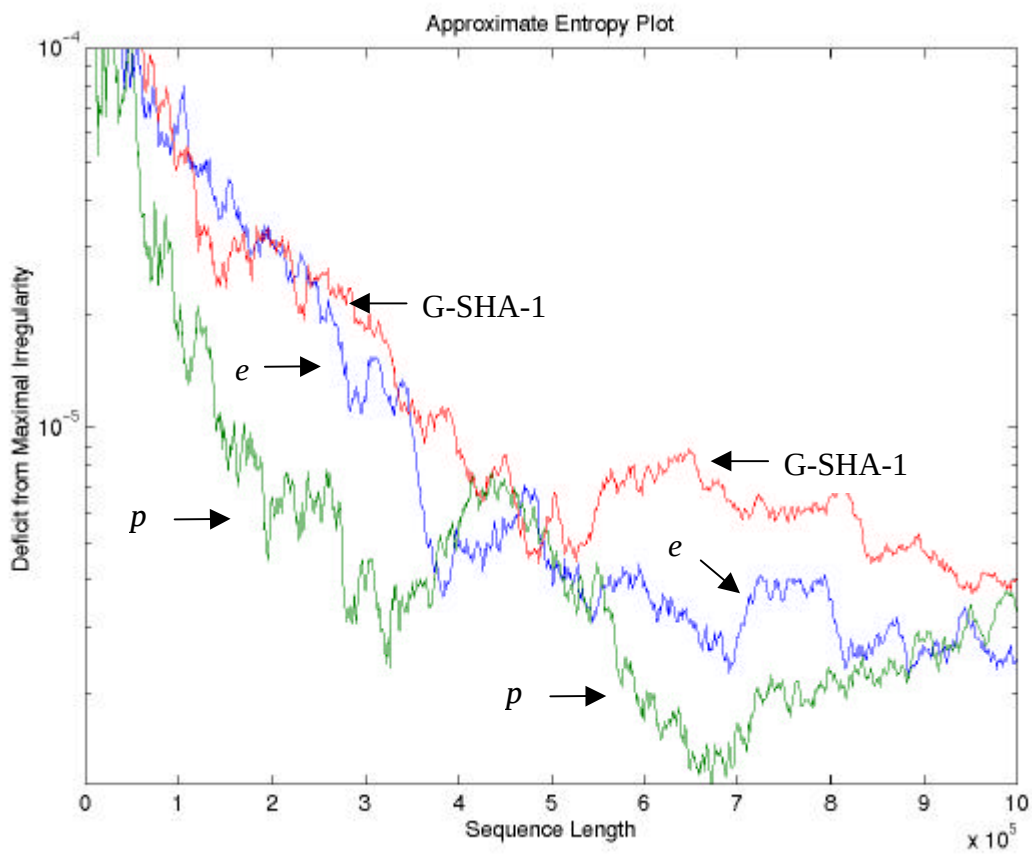


Figure H.2: Approximate Entropy Graph

[19] It is worth noting that, for larger block sizes and sequence lengths on the $O(10^6)$, SHA-1 binary sequences yield deficit values on the $O(10^{-9})$.

(c) Linear Complexity Profile

Figure H.3 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, $x_1, x_2, \cdots, x_{127}$, subsequent bits in the sequence are generated according to the rule, $x_i = x_{i-1} \oplus x_{i-127}$ for i ≥ 128.

The Berlekamp-Massey [20]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.

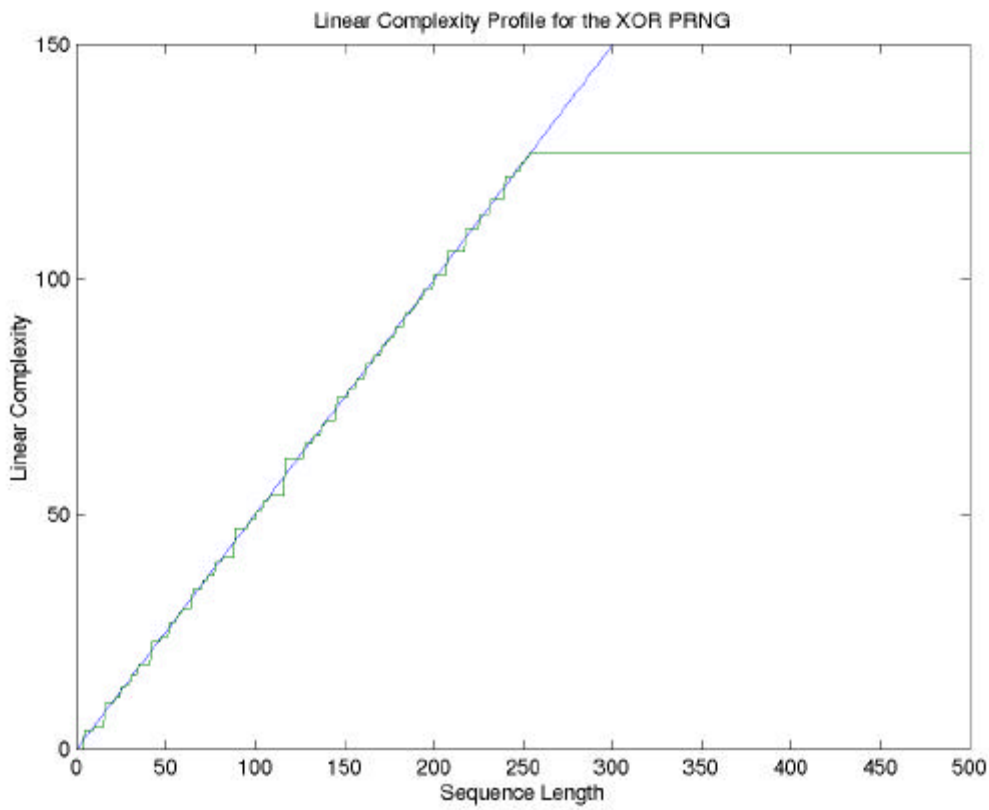


Figure H.3*:* Linear Complexity Profile

[20] For a description of the algorithm see Chapter 6 - Stream Ciphers, which may be accessed at http://www.cacr.math.uwaterloo.ca/hac/.

APPENDIX I: INSTRUCTIONS FOR INCORPORATING ADDITIONAL STATISTICAL TESTS

In order to add another statistical test to the test suite, the user should make the following modifications:

1. [In the file **include**/*defs.h*]

   Insert any test input parameters into the **testParameters** structure. Increment the value of **NUMOFTESTS** by the number of tests to be added.

2. [In the file **include**/*proto.h*]

   Insert the statistical test function prototype declaration.

3. [In the file **src**/*utilities1.c*]

   Embed the test function call into the **nist_test_suite** function. For example, if the current number of tests is 16, and one test is to be added, insert the following code:

```
if ((testVector[0] == 1) || (testVector[17] == 1))
     myNewTest(tp.myNewTestInputParameters,tp.n);
```

4. [In the file **src**/*myNewTest.c*]

   Define the statistical test function. Note: The programmer should embed fprintf statements using *stats*[*x*], and *results*[*x*] as the corresponding output channel for writing intermediate test statistic parameters and *P-values*, respectively. *x* is the total number of tests.

5. [In the file **src**/*utilities2.c*]

   (a) In the function, **openOutputStreams**, insert the string, "myNewTest" into the **testNames** variable. In the function, **chooseTests**, insert the following lines of code (as modified by the actual number of total tests):

```
printf("\t\t\t    12345678911111111\n");
printf("\t\t\t              01234567\n");
```

   Note: For each PRNG defined in the package, a sub-directory **myNewTest** must be created.

   (b) In the function, **displayTests**, insert a printf statement. For example, if the total number of tests is 17, insert

```
printf("   [17] My New Test\n");
```

(c) If an input test parameter is required, in the function, **fixParameters**, insert the following lines of code (under the assumption that **myNewTestParameter** is an integer). For example, if the total number of tests is 17, insert

```
if (testVector[17] == 1) {
     printf("\tEnter MyNewTest Parameter Value:  ");
     scanf("%d", &tp.myNewTestParameter);
}
```

APPENDIX J: INSTRUCTIONS FOR INCORPORATING ADDITIONAL PRNGs

In order to add a PRNG to the test suite, the user should make the following modifications:

1. [In the file **include**/*defs.h*]

   Increment the variable **NUMOFGENERATORS** by one.

2. [In the file **include**/*generators4.h*]

   Insert the generator function prototype declaration. For example,

```
void myNewPRNG();
```

3. [In the file **generators**/*generators4.c*]

   Define the generator function. The general scheme for each PRNG defined in the test suite is as follows:

```
Allocate space for epsilon, the n-bit BitField array.
for i = 1 to numOfBitStreams {
  Construct an n-bit sequence using myNewGenerator and
  store in epsilon.
  Invoke the nist_test_suite.
}
Deallocate the space given to epsilon.
```

   Note: A sub-directory called **myNewPRNG/** must be created. Under this new directory, a set of sub-directories must be created for each of the test suite statistical tests. The script **createScript** has been included to facilitate this operation.

4. [In the file **src**/*utilities2.c*]

   (a) In the function, **generatorOptions**, insert the following lines of code:

```
case 12:  *streamFile = "myNewPRNG";
          break;
```

   (b) In the function, **invokeTestSuite**, insert the following lines of code:

```
case 12:  myNewPRNG();
          break;
```

   (c) In the function, **openOutputStreams**, insert the generator string name into the **generatorDir** variable. For example,

```
char generatorDir[20][20] = {"AlgorithmTesting/",
                                …,"XOR/", "MYNEWPRNG/"};
```

Similarly, in the routine, **partitionResultFile**, in the file, *assess.c*.

APPENDIX K: GRAPHICAL USER INTERFACE (GUI)

K.1 Introduction

A simple Tcl/Tk graphical user interface (GUI) was developed as a front-end to the NIST Statistical Test Suite. The source code may be found in the file, *rng-gui.tcl*. The interface consists of a single window with four regions. The topmost region contains the software product laboratory affiliation. The left half of the window consists of a checklist for the sixteen statistical tests. The user should select or de-select the set of statistical tests to be executed.

The right half of the window is sub-divided into an upper and lower portion. The upper portion consists of required parameters that must be provided in order to execute the tests. The lower portion consists of test dependent parameters that must be provided only if the corresponding test has been checked.

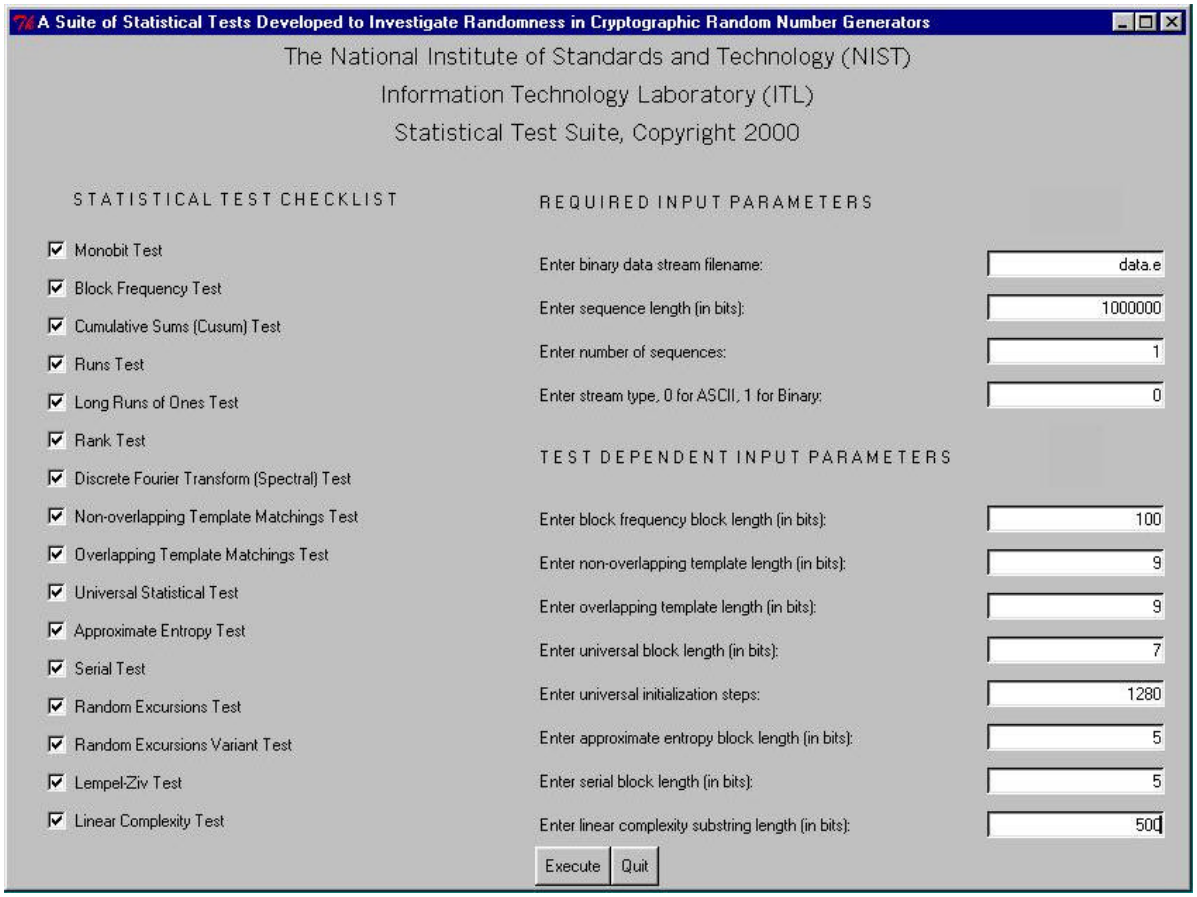


Figure K.1: Tcl/Tk GUI for the NIST Statistical Test Suite

Once the user has selected the statistical tests, the required input parameters, and the test dependent input parameters, then the user should depress the *Execute* button to invoke the

battery of statistical tests. This will result in the de-iconification of the GUI. Upon completion, the GUI will re-iconify. The user should then proceed to review the file, *finalAnalysisReport.txt* to assess the results.

K.2 An Example

The following table presents an example of the use of the GUI. The user has checked all sixteen of the statistical tests and entered:

data.e as the binary date stream filename	• *9* as the overlapping template block length
• a sequence length of *1000000* bits	• *7* as the universal block length
• *1* as the number of binary sequences	• *1280* as the universal initialization steps
• *0* as the stream type	• *5* as the approximate entropy block length
• *100* as the block frequency block length	• *5* as the serial block length
• *9* as the nonoverlapping template block length	• *500* as the linear complexity substring length

K.3 Guidance in the Selection of Parameters

Section 2 provides the recommended parameter choices for each statistical test.

K.4 Interpretation of Results

Section 4.2 contains information regarding the interpretation of empirical results.

K.5 Tcl/Tk Installation Instructions

Tcl/Tk may be obtained from the Scriptics website at http://www.scriptics.com/. Download Tcl/Tk 8.1 for the target platform from http://dev.scriptics.com/software/tcltk/choose.html.

K.6 References

[1] Brent Welch, Practical Programming in Tcl and Tk, 2nd edition**.** Prentice Hall PTR, 1997.
[2] Clif Flyntf, Tcl/Tk for Real Programmers. Academic Press, 1999.

APPENDIX L: DESCRIPTION OF THE REFERENCE PSEUDO RANDOM NUMBER GENERATORS

The NIST Statistical Test Suite supplies the user with nine pseudo-random number generators. A brief description of each pseudo-random number generator follows. The user supplied sequence length determines the number of iterations for each generator.

L.1 Linear Congruential Generator (LCG)

The input parameter for the Fishman and Moore[21] LCG[22] is fixed in code but may be altered by the user.

Input Parameter:

z_0 = 23482349

Description:

Given a seed z_0, subsequent numbers are computed based on $z_{i+1} = a*z_i \bmod (2^{31}-1)$, where a is a function of the current state. These numbers are then converted to uniform values in [0,1]. At each step, output '0' if the number is ≤ 0.5, otherwise output '1'.

L.2 Quadratic Congruential Generator I (QCG-I)

The input parameters to the QCG-I are fixed in code, but may modified by the user.

Input Parameters:

p = 987b6a6bf2c56a97291c445409920032499f9ee7ad128301b5d0254aa1a9633fdbd378
d40149f1e23a13849f3d45992f5c4c6b7104099bc301f6005f9d8115e1

x_0 = 3844506a9456c564b8b8538e0cc15aff46c95e69600f084f0657c2401b3c244734b62e
a9bb95be4923b9b7e84eeaf1a224894ef0328d44bc3eb3e983644da3f5

[21] Fishman, G. S. and L. R. Moore (1986). An exhaustive analysis of multiplicative congruential random number generators with modulus 2**31-1, SIAM Journal on Scientific and Statistical Computation, 7, 24-45.
[22] Additional information may be found in Chapter 16 (Pseudo-Random Sequence Generators & Stream Ciphers), Section 16.1 (Linear Congruential Generators) of Bruce Schneier's book, Applied Cryptography: Protocols, Algorithms and Source Code in C, 2nd edition, John Wiley & Sons, 1996.

Description:

Using a 512-bit prime p, and a random 512-bit seed x_0, construct subsequent elements (each 512-bit numbers) in the sequence via the rule:

$$x_{i+1} = x_i^2 \bmod p, \text{ for } i \geq 0.$$

L.3 Quadratic Congruential Generator II (QCG-II)

The input parameter to the QCG-II is fixed in code, but may be modified by the user.

Input Parameter:

x_0 = 7844506a9456c564b8b8538e0cc15aff46c95e69600f084f0657c2401b3c244734b62e
a9bb95be4923b9b7e84eeaf1a224894ef0328d44bc3eb3e983644da3f5

Description:

Using a 512-bit modulus, and a random 512-bit seed x_0, construct subsequent elements (each 512-bit numbers) in the sequence via the rule:

$$x_{i+1} = 2x_i^2 + 3x_i + 1 \bmod 2^{512}, \text{ for } i \geq 0.$$

L.4 Cubic Congruential Generator (CCG)

The input parameter to the CCG is fixed in code, but may be modified by the user.

Input Parameter:

x_0 =7844506a9456c564b8b8538e0cc15aff46c95e69600f084f0657c2401b3c244734b62ea
9bb95be4923b9b7e84eeaf1a224894ef0328d44bc3eb3e983644da3f5

Description:

Given a 512 bit seed x_0, construct subsequent 512-bit strings via the rule:

$$x_{i+1} = x_i^3 \bmod 2^{512}, \text{ for } i \geq 0.$$

L.5 Exclusive OR Generator (XORG)

The input parameter to the XORG is a 127-bit seed that is fixed in code, but may be user modified.

Input Parameter:

$x_1, x_2, \ldots, x_{127}$ = 00010110110110010001011110010010100110111011010001000000101
01111111010100100001010110110000000000010011000010111001111111100111

Description:

Choose a bit sequence, $x_1, x_2, \ldots, x_{127}$. Construct subsequent bits via the rule:

$$x_i = x_{i-1} \oplus x_{i-127}, \text{ for } i \geq 128.$$

L.6 Modular Exponentiation Generator (MODEXPG)

The input parameters to the MODEXPG are fixed in code, but they may be user modified.

Input Parameters:

seed = 7AB36982CE1ADF832019CDFEB2393CABDF0214EC

p = 987b6a6bf2c56a97291c445409920032499f9ee7ad128301b5d0254aa1a9633fdbd378
d40149f1e23a13849f3d45992f5c4c6b7104099bc301f6005f9d8115e1

g = 3844506a9456c564b8b8538e0cc15aff46c95e69600f084f0657c2401b3c244734b62ea
9bb95be4923b9b7e84eeaf1a224894ef0328d44bc3eb3e983644da3f5

Description:

A sequence $\{x_i\}$ of 512-bit pseudo-random numbers can be generated as follows:
Choose a 512-bit **prime** p, and a **base** g, as in the Digital Signature Standard (DSS). Choose an arbitrary 160-bit seed y. Let $x_1 = g^{seed} \bmod p$ and $x_{i+1} = g^{y_i} \bmod p$, for $i \geq 1$ where y_i is the lowest-order 160 bits of x_i. Splicing together the $\{x_i\}$ will generate an n-bit sequence.

L.7 Secure Hash Algorithm Generator (SHA1G)

The input parameters to the SHA1G are fixed in code, but may be user modified. The length of the key, *keylen* should be chosen in the interval [160, 512].

Input Parameters:

seedlen = 160
Xseed = 237c5f791c2cfe47bfb16d2d54a0d60665b20904

keylen = 160
Xkey = ec822a619d6ed5d9492218a7a4c5b15d57c61601

Description:

For a detailed description of SHA1G (the FIPS 186 one-way function using SHA-1), visit http://www.cacr.math.waterloo.ca/hac/about/chap5.pdf.zip, especially p. 175.

L.8 Blum-Blum-Shub Generator (BBSG)

The input parameters to the BBSG are not fixed in code. They are variable parameters, which are time dependent. The three required parameters are two primes, p and q, and a random integer s.

Input Parameters:

Two primes p and q such that each is congruent to 3 modulo 4. A random integer s (the seed), selected in the interval $[1, pq-1]$ such that $gcd(s,pq) = 1$. The parameters p, q and s are not fixed in code; thus, the user will not be able to reconstruct the original sequence because these values will vary (i.e., they are dependent on the system time). To reproduce a sequence the user must modify the code to fix the variable input parameters.

Description:

For a detailed description of the Blum-Blum-Shub pseudo-random number generator, visit http://www.cacr.math.waterloo.ca/hac/about/chap5.pdf.zip, especially p. 186. Pate Williams' ANSI C reference implementation may be located at ftp://www.mindspring.com/users/pate/crypto/chap05/blumblum.c.

L.9 Micali-Schnorr Generator (MSG)

The input parameters to the MSG are not fixed in code. They are variable parameters, which are time dependent. The four required parameters are two primes, p and q, an integer e, and the seed x_0.

Input Parameters:

Two primes p and q. A parameter e, selected such that $1 < e < f = (p-1)(q-1)$, $gcd(e, f) = 1$, and $80e < N = floor(lg\ n + 1)$. A random sequence x_0 (the seed) consisting of r (a function of e and n) bits is chosen. The parameters e, p, q, and x_0 are not fixed in code; thus, the user will not be able to reconstruct the original sequence because these values will vary (i.e., they are dependent on the system time). To reproduce a sequence the user must modify the code to fix the variable input parameters.

Description:

For a detailed description of the Micali-Schnorr pseudo-random number generator, visit http://www.cacr.math.waterloo.ca/hac/about/chap5.pdf.zip, especially p. 186. Pate Williams'

ANSI C reference implementation may be located at ftp://www.mindspring.com/users/pate/crypto/chap05/micali.c.

L.10 Test Results

The following table depicts test-by-test failures for the above reference generators.

Statistical Test	Excessive Rejections	Lacks Uniformity	Generator
Frequency	X	X	Modular Exponentiation
	X	X	Cubic Congruential
	X	X	Quadratic Congruential (Type I)
Block Frequency		X	Cubic Congruential
	X	X	XOR
Cusum	X	X	Micali-Schnorr
	X	X	Modular Exponentiation
	X	X	Cubic Congruential
	X	X	Quadratic Congruential (Type I)
Runs	X		Modular Exponentiation
	X	X	Cubic Congruential
	X		Quadratic Congruential (Type I)
Rank	X	X	XOR
Spectral	X	X	Cubic Congruential
		X	Quadratic Congruential (Type II)
Aperiodic Templates	X		ANSI X9.17
	X		Micali-Schnorr
	X		Modular Exponentiation
	X	X	Cubic Congruential
	X		Quadratic Congruential (Type I)
	X		Quadratic Congruential (Type II)
	X	X	XOR
Periodic Templates	X		Modular Exponentiation
	X	X	XOR
Approximate Entropy	X	X	Modular Exponentiation
	X	X	Cubic Congruential
	X	X	Quadratic Congruential (Type I)
	X	X	XOR
Serial	X	X	Modular Exponentiation
	X	X	Cubic Congruential
	X	X	Quadratic Congruential (Type I)
	X	X	XOR

Table M.1: Illustration of Rejection/Uniformity Failures

APPENDIX M: REFERENCES

[1] M. Abramowitz and I. Stegun, Handbook of Mathematical Functions, Applied Mathematics Series. Vol. 55, Washington: National Bureau of Standards, 1964; reprinted 1968 by Dover Publications, New York.

[2] T. Cormen, C. Leiserson, & R. Rivest, Introduction to Algorithms. Cambridge, MA:The MIT Press, 1990.

[3] Gustafson et al., "A computer package for measuring strength of encryption algorithms," Journal of Computers & Security. Vol. 13, No. 8, 1994, pp. 687-697.

[4] U. Maurer, "A Universal Statistical Test for Random Bit Generators," Journal of Cryptology. Vol. 5, No. 2, 1992, pp. 89-105.

[5] A. Menezes, et al., Handbook of Applied Cryptography. CRC Press, Inc., 1997. See http://www.cacr.math.uwaterloo.ca/hac/about/chap5.pdf.zip.

[6] W. Press, S. Teukolsky, W. Vetterling, Numerical Recipes in C : The Art of Scientific Computing**,** 2nd Edition. Cambridge University Press, January 1993.

[7] G. Marsaglia**,** DIEHARD Statistical Tests: *http://stat.fsu.edu/~geo/diehard.html.*

[8] T. Ritter, "Randomness Tests and Related Topics," http://www.io.com/~ritter/ RES/RANDTEST.HTM.

[9] American National Standards Institute: Financial Institution Key Management (Wholesale), American Bankers Association, ANSI X9.17 - 1985 (Reaffirmed 1991).

[10] FIPS 140-1, Security Requirements for Cryptographic Modules, Federal Information Processing Standards Publication 140-1. U.S. Department of Commerce/NIST, National Technical Information Service, Springfield, VA, 1994.

[11] FIPS 180-1, Secure Hash Standard, Federal Information Processing Standards Publication 180-1. U.S. Department of Commerce/NIST, National Technical Information Service, Springfield, VA, April 17, 1995.

[12] FIPS 186, Digital Signature Standard (DSS), Federal Information Processing Standards Publication 186. U.S. Department of Commerce/NIST, National Technical Information Service, Springfield, VA, May 19, 1994.

[13] MAPLE, A Computer Algebra System (CAS). Available from Waterloo Maple Inc.; http://www.maplesoft.com.