

XVIII. RANDOM NUMBER GENERATION	2
INTRODUCTION:	2
REPORTED PROBLEMS:.....	4
BASIC TESTS ON PRNS	4
MEASURES OF RANDOMNESS, THE DIEHARD TEST SUITE:.....	5
MEASURES OF RANDOMNESS, THE NIST TEST SUITE:.....	8
MEASURES OF RANDOMNESS, THE TESTU01 TEST SUITE:	9
MEASURES OF UNIFORMITY	10
TESTING EXCEL RNGS	12
EXCEL RANDOM NUMBER GENERATORS (PRNS).....	13
RAND (EXCEL 2000).....	13
DESCRIPTION.....	13
CHARACTERISTICS	14
RAND-2000 OUTPUT:	17
RAND-2000 PERIOD:	18
THE DIEHARD TESTS	18
RAND (EXCEL 2003 AND 2007).....	21
THE MICROSOFT VERSION.....	21
IS THE ALGORITHM THAT MICROSOFT USES THE TRUE WICKMAN- HILL ALGORITHM?.....	23
SETTING THE SEED	24
PERIOD	24
DIEHARD TEST RESULTS.....	24
TESTU01 TEST RESULTS	25
RANDBETWEEN	27
DATA ANALYSIS TOOL PAC ROUTINES (EXCEL 2000 AND EXCEL 2003) 27	
RANDOM.....	28
SAMPLING	29
RETROSPECT ON RANDOM NUMBER SEQUENCE TESTS	30
FUTURE VERSIONS OF RAND	30
DOES RAND HAVE TO BE THE LATEST MODEL	30
POSSIBLE FUTURE VERSIONS WITH BETTER CAPABILITIES.....	31

XVIII. RANDOM NUMBER GENERATION

INTRODUCTION:

The generation of random numbers from a computer is a large field of interest. Because a computer algorithm is used, the result is not a true random number. Theoretically, the generated number PRN (from a random number generator or PRNG) in a sense is predicted from a set of prior numbers, violating the prime definition of a random number as something that cannot be correctly “guessed” or predicted. True random numbers theoretically can be obtained from physical processes such as turbulent air streams (lottery number balls), mechanical devices, the counting of time intervals between events of radioactive decays, from electrical noise (thermal electron noise) and now random bit sequences from physical quantum devices (Turiel 2007).

The pseudo-random numbers (PRNs) generated by computers (PRNGs) are very close to a true random number, but never can be considered truly random. Some approaches to obtaining true random numbers combine PRNs with outputs from electrical devices to obtain true random number sets. However the results have not been too satisfactory. (Numbers from electronic noise exhibit correlations that are very small, but nevertheless are statistically significant.)

The general areas that require random numbers from a computer are:

- a. Cryptology, secure data transmissions, coding/decoding
- b. Simulations
- c. Statistical applications.
- d. Probabilistic algorithms
- e. Computer games
- f. Gambling, machines, internet, etc.
- g. Security, access control, operations, etc.

Essentially, each one has different requirements on the nature of the PRN sequence.

There are basically now three types of PRNs that come from computers. The first is represented as a sequence of binary integers, each one made up of n random bits. The second is represented as a sequence of floating point numbers and the third is a sequence of bits from a physical device either within or external to the computer. This third device generates random bits or random words based on a physical noise or quantum source. This third device may produce true RNs that may be modified by the computer resulting in PRNs that are not exactly true RNs. For example the application may produce normally distributed integers by a reject/accept algorithm, which inherently returns PRNs. Note however the conversion of random bits to decimal numbers (integers) or vice-versa still retains the RN characteristic. (Turiel 2007).

The integer-to-floating point conversion in Excel is referred to as IRN(n), where n is the bit length. The resulting floating point number is referred to as FPRN, based on the IEEE-754 architecture. By using computer commands (and some assumptions), IRN(n)s can be converted to FPRNs and vice versa. Turiel (2007) quotes work by L’Ecuyer that indicates that this conversion does not change the inherent randomness of the IRN(n) form. Consequently we can take a random bit string (or integer) and do tests of

randomness as floating point numbers. This is the basis for Marsaglia's Diehard test sequences and L'Ecuyer's TEST01 sequences. (Turiel 2007)

The random number (PRN) can represent a value from any statistical distribution. However the main one is a uniform distribution. Again by using a 'program', the uniform PRN can be transformed to another PRN corresponding to the desired distribution. Therefore, the primary focus here will be on the PRN representing a uniform distribution.

Marsaglia (2002) calls these PRN sequences a set Z , where each prior member is called a "seed". If the RNG uses a vector of values as a seed (a set of m -tuples), and generates a new set of m -tuples each call, then this PRNG is more desirable. In general then, the symbol Z can be used to represent the sequence of prior PRNs from a generator.

In Excel, FPRNs (Floating Point Random Numbers) are returned by the RAND function. However the algorithms are different. In Excel 2000, RAND is an FPRN. In Excel 2003 and 2007, RAND is generated from three separate IRN(16)s, converted to FP values, and rescaled to the zero-to-one interval, giving a single FPRN output. The seed in this case is the three separate IRN(16)s which change on every RAND call. These three are hidden from access and can't be modified. This means that a given sequence can never be repeated.

Testing of PRNs is mainly done on IRN(32) sets. In testing RAND outputs, this requires that the FPRN output be converted to a testable IRN(32) sequence. Given that the FPRN number is represented by a value from 0 to 1 in the IEEE-754 format, the equivalent IRN(m) has a length of 53 bits. However, only the leading 32 bits are tested for randomness. The low order bits are discarded.

The diehard tests are on a large set of IRN(32) bits as integers.

The TESTU01 test suite requires that the RNG being tested be programmed in a C module. The test program then generates the necessary bit streams for testing from this module. The user selects what bits to test.

What tests to perform on the Excel RAND output, is a good question. L'Ecuyer and Simand (2004a) say that there is no universal battery of tests that can guarantee from a given RNG, fully reliable random number sequences for any application. McCullough used the Diehard suite, and is now using the TESTU01 suite. There are the NIST tests (Soto 1998) and commercial software test packages that have batteries of tests to apply. The problem now is that these different software packages use common test names, (such as "Birthday Test"), but give entirely different test results from the same RNG, for a test under the same name.

The essential direction now in RNGs is to obtain RNGs that pass cryptology and security tests. New tests are being invented to critically test these special RNGs. This means that a currently acceptable RNG may be found in the future to be "unacceptable". Marsaglia (2003) mentions this problem. To Marsaglia (2002), there are other directions that development needs to go in, such as in situations that require "equal probability of occurrence of any given value from the set Z " in a legal sense, among a very, very large number of combinations that have equal probabilities of occurrence.

Simulations have special requirements that cannot be adequately met with a RNG specifically tailored for security or legal applications.

REPORTED PROBLEMS:

The reported problems with Excel random number generators were generally focused on RAND-2000, the general random number generator in all Excel versions prior to Excel 2003.

Table 18-1: Excel Complaints About RAND and RANDBETWEEN

Application or Function	Problem	Source	Fix or Comments
RAND	Short cycle length	McCullough (2000)	See discussion below
RAND	Fails The Diehard Test Suite	McCullough (2000)	See discussion below
RANDBETWEEN	First value only half the frequency	RSS 1996	RANDBETWEEN generates only random integers, uniformly distributed between a lower limit and an upper limit. I found that the limit integers occur just as frequent as the in-between integers. The claim is incorrect.
Random Set of Integers	Some integers are duplicated	Lists	On a blank worksheet, put the list of integers in column A in order. In column B put =RAND() next to the first integer, and copy down to the end of the list. Go to the menu, select Data, select Sort, Sort on column B ascending. After sorting, select from column A the number of integers you want from the top down. There will be no duplicates in the list.

BASIC TESTS ON PRNS

There is a philosophical issue here that Turin (2007) raises. This is the point that the PRNG can generate PRNs that successfully pass some statistical test. This raises the issue of choice of statistical tests. L'Ecuyer and Simard (Test01) state that there is no single test or battery of tests that proves a generator is definitely random. We can then show that a PRNG does not produce a true RN sequence when it fails a test. We can also say that there is some unknown test that will fail a given PRNG sequence, and therefore all PRNGs are not truly random. Turin (2007) points out that no finite string of bits from a finite generator can be algorithmically random. Choice then of a PRNG to use then is up to the user, knowing that it will never pass all possible tests. Unfortunately, the tests that

pass and fail the PRNG do not give a user any guidance on when to use or not to use a given PRNG,

There are common tests to PRNs that test for the basic properties of a random number sequence. Excel has three different PRNGs, one the function RAND for worksheets, another is the ATP routine that gives RNs from the routine RANDOM and from the function RANDBETWEEN, and the third is the random number generator in the Excel Visual Basic package that can be used to build user Excel functions and routines. The tests described here were applied to the RAND in Excel on both the 2000 and 2003 versions. The ATP RANDOM for Excel 2000 was given a test. Since there was no change in these for the Excel 2007 version, they also apply to Excel 2007.

MEASURES OF RANDOMNESS, THE DIEHARD TEST SUITE:

McCullough and Wilson used this particular test suite in previous articles. Therefore it represents an initial standard for which to judge RAND outputs. They are all based on evaluations of the random number as a IRN(32) sequence.

THE DIEHARD-II TEST SUITE

The diehard-II set of tests (Marsaglia 1995) outputs a long narrative giving probability values on the outcomes of each of the diehard-II tests. Altman (2000) and McCullough (1999) lists the diehard-II suite as 18 tests. Marsaglia (1995) lists the suite as 15 tests. He counts all the monkey tests as one test. To run Diehard-II, a computer binary file of at least 3 Megabytes of IRN(32) values is needed.

Table 18-2: Basic Diehard-II Tests

Reference Number	Diehard Test Number	Menu Titles	Symbol	Test On	Output Block Title	Measure
1	1	Birthday Spacings	BDAY	Bits	Birthday Spacings Test	KS Test on Chi-Square p Values
2	2	Overlapping Permutations	OPERM5	Long Integers	Overlapping 5-Permutation Test	Chi-square GOF test, state counts
3	3	Ranks of 31x31 and 32x32 Matrices	RANK	Long Integers	Binary Rank Test, 31x31	Chi-Square GOF, rank frequencies
4	3	Ranks of 31x31 and 32x32 Matrices	RANK	Long Integers	Binary Rank Test, 32x32	Chi-Square GOF, rank frequencies
5	4	Ranks of 6x8 Matrices	RANK	Bytes	Binary Rank Test, 6x8	KS Test on 25 Chi-Square, rank frequencies
6	5	Monkey Tests on 20-bit Words	MTUPLE	20 bit words	Overlapping 20-tuples Bitstream Test	K-S Test on 20 p values
7	6	Monkey Tests OPSO, QQSO and DNA	OPSO	10 bit words	Overlapping Pairs, Sparse Occupancy	K-S Test on 23 p values
8	6	Monkey Tests OPSO, QQSO and DNA	QQSO	5 bit words	Overlapping Quadruples, Sparse Occupancy	K-S Test on 28 p values
9	6	Monkey Tests OPSO, QQSO and DNA	DNA	2 bit words	DNA Test	K-S Test on 31 p values
10	7	Count the 1's In a stream of bytes		Bytes	Count the ones Test, stream of bytes	Frequency of Words. Chi-Square GOF
11	8	Count the 1's in specific bytes		Bytes	Count the ones Test, specific bytes	KS Test on 25 p Values from Frequency of Specified Bytes
12	9	Parking Lot Test	PARKLOT	Long Integers	Parking Lot test	KS Test for 10 Test Sets on p Value of Number Parked
13	10	Minimum Distance Test	MINDIST	Long Integers	Minimum Distance Test	KS test on 100 Test Sets of p Values
14	11	Random Spheres Test	3D	Long Integers	3D Spheres Test	KS test on 20 Test Sets

Reference Number	Diehard Test Number	Menu Titles	Symbol	Test On	Output Block Title	Measure
15	12	The Squeeze Test	SQUEEZE	Long Integers	Squeeze Test	Chi-square GOF, 43 cell frequencies
16	13	Overlapping Sums Test	OSUM	Long Integers	Overlapping Sums Test	KS Test on 10 Test Sets
17	14	Runs Test	RUNS	Long Integers	Runs Test	Four KS Tests on 10 Test Sets
18	15	The Craps Test	CRAPS	Long Integers	Craps Test	Two p-values on winning

THE DIEHARD-III SUITE OF TESTS

The diehard-III set of tests (Marsaglia 2002) is an improved set of tests. Marsaglia says, “The new Diehard tests contain several new ‘tough’ tests that relate to use of random integers in computational number theory and cryptography. Some of the most effective tests for randomness are based on what I call Monkey Tests...One of the tests in the new version of Diehard (Diehard-III) is called the ‘Gorilla test’, in the sense of a strong monkey test... Another new test is the gcd test... The new tests include a stronger version of my ‘birthday spacings’ test and others.”

The 17 tests in Diehard-III are:

- 1 Birthday Spacings
- 2 GCD
- 3 Gorilla
- 4 Overlapping Permutations
- 5 Ranks of 31x31 and 32x32 matrices
- 6 Ranks of 6x8 Matrices
- 7 Monkey Tests on 20-bit Words
- 8 Monkey Tests OPSO, OQSO, DNA
- 9 Count the 1`s in a Stream of Bytes
- 10 Count-the-1`s in Specific Bytes
- 11 Parking Lot Test
- 12 Minimum Distance Test
- 13 Random Spheres Test
- 14 The Squeeze Test
- 15 Overlapping Sums Test
- 16 Runs Up and Down Test
- 17 The Craps Test

To run the Diehard-III set of tests, a binary file of at least 67,108,889 IRN(32)s are needed.

Assessment of Excel in regard to the diehard tests in the literature, gives a subjective pass or failure (see Altman (2000) and McCullough (1999)), not the probability values as stated in the diehard output report.

Marsaglia says the following about test p values in both Diehard versions:

“Most of the tests in DIEHARD return a p-value, which should be uniform on (0,1) if the input file contains truly independent random bits. Those p-values are obtained by $p=F(X)$, where F is the assumed distribution of the sample random variable X---often normal. But that assumed F is just an asymptotic approximation, for which the fit will be worst in the tails. Thus you should not be surprised with occasional p-values near 0 or 1, such as .0012 or .9983. When a bit stream really FAILS BIG, you will get p's of 0 or 1 to six or more places. By all means, do not, as a Statistician might, think that a $p < .025$ or $p > .975$ means that the RNG has "failed the test at the .05 level". Such p's happen among the hundreds that DIEHARD produces, even with good RNG's. So keep in mind that 'p happens'.”

The essential basis is that the central limit theorem does not apply, and large samples do not converge in probability. For a good random number generator, p values from tests will be uniformly distributed, and very large RN-set p values will vary just as much as small RN sets.

My interpretation is that when p's of 0 or 1 are reported, it is a test failure. Marsaglia is talking about uncertainty in the tail areas from a test. A p value here of 0.99999 from a test would not be considered a failure. If a p value is high and when different seed values are then tried, and the p values are still high, then one would be more inclined to classify it as a test failure, rather than just “p happens”. This again is subjective, and the problem of bean-counting test pass/failures is not resolved. However the bean-counting is necessary for ranking software programs.

MEASURES OF RANDOMNESS, THE NIST TEST SUITE:

These are a suite of tests, developed by the Computer Security and Statistical Engineering Divisions of NIST. The specific objective was to develop a suite of tests suitable in the assessment of binary stream randomness. The main focus was on PRNs used for cryptography and security applications. The tests are:

- Frequency
- Block Frequency
- CUSUM
- Runs
- Longest Run of Ones
- Marsaglia's Rank
- Spectral (DFT)
- Template Matchings
- Maurer's Universal
- Approximate Entropy
- Random Excursions
- Moving Averages

Lempel Ziv Complexity
Linear Complexity

These tests were devised to test the typical linear congruential generators that were commonly used in computers through the mid 90's. The newer RNGs require more extensive testing than the NIST suite.

Microsoft reports in KBA 828795, that RAND-2003 passed these tests, but gives no details.

This test suite was not available, consequently there were no NIST tests done here on RAND 2000 or RAND 2003.

MEASURES OF RANDOMNESS, THE TESTU01 TEST SUITE:

These are a suite of tests developed by Pierre L'Ecuyer of the University of Montreal; Canada. TESTU01 is available as a free download, but only works under the GNU programming language system. TESTU01 only works under the GNU C compiler (GCC). A Windows environment package is available, but requires the Bourne shell or GCC to work.

Test modules, which contain groups of tests, are defined as follows:

- Smultin – Tests on uniformity of t dimensional vectors
- Sentrop – Discrete continuous empirical entropies
- Snpair – Distances between closest points, unit multidimensional cells
- Sknuth – Classical statistical tests from Knuth, "The Art of Computer Programming", 1981 edition
- Smarsa – Marsaglia's tests. The ones with (Diehard) are supposed to be identical to the test in the Diehard suites¹.
 - SerialOver – Overlapping t-tuple test (Diehard)
 - CollisionOver – Overlapping pairs, sparse occupancy (OPSO, Diehard)
 - OPSO – Three special cases of CollisionOver
 - Monkey – Monkey Test (Diehard)
 - Monkeybits – Similar to Monkey, except cells are a string of bits
 - BirthdaySpacings – Birthday Spacings (Diehard)
 - MatrixRank – Rank of a random binary matrix (Diehard)
 - GCD – Greatest common divisor (Diehard-III)
- Svaria – Implement various tests of Uniformity
- Swalk – Tests based on discrete random walks
- Scomp – Three tests on the evaluation of the linear complexity of sequences of bits (Lempel-Ziv)

¹ The diehard tests were written in Fortran and have undergone very extensive "patching" by his students. As a result the specific algorithm cannot be explicitly defined. He has stated that the translation to C results in outputs different from the Fortran outputs. The Diehard versions used in this paper were the Fortran complied programs adapted for DOS.

Spectral – Discrete Fourier Transforms
Sstring – Various tests on strings of bits
Scatter – Two-dimensional scatter plots (shows Marsaglia ‘planes’)

It is difficult to compare the Diehard Suite with the TESTU01 Suite, since the outlook and the “languages” are completely different. Turiel (2007) claims that the TESTU01 suite includes all the DIEHARD and NIST tests. Although test names appear to be similar, the internals are different. Over the last 15 years, Marsaglia and L’Ecuyer have gone in different directions and they promote different RNG structures. There is very little commonality here. Marsaglia has a more tolerant view of p values, but L’Ecuyer takes a narrow view, establishing p limits of 0.01 and 0.99 as test warning points, and recommending that the test (and input data) be repeated until a satisfactory outcome is attained. If not then this is a reject.

MEASURES OF UNIFORMITY

These are measures of how well the distribution conforms to a uniform distribution.

The first three of the following are commonly used to determine uniformity:

The Chi-Square Goodness of Fit Test (GOF):

The GOF test is to divide the 0 to 1 interval into even slots, determine which slot a given RN falls in, and after running through N RNs, counting the number in each slot and doing a chi-square GOF test on the counts. A small p (right tail area) indicates a lack of fit.

The Kolmogorov-Smirnov (KS) test (KS)

The KS test is a supremum class statistic. The KS test takes blocks of N RNs, sorts them in ascending order, determines the difference (D+ or D-) between the RN value and the expected value based on the position in the sorted order. It then tests the maximum difference, adjusted for block size. For a KS test, a zero p corresponds to a perfect fit and a p=1 value indicates a no-fit. Given n, a p value can be obtained from standard tables. (See D’Agostino and Stephens, 1986, chapter 4).

The problem with this test is that it is often misunderstood. There are two different KS distributions. 1st, one has to “ahead of time” decide to test D+ or D- and stick with it. 2nd, pick the maximum, D+ or D- within the given set. If the distribution for the 1st is used, you cannot switch to 2nd. Knuth (1998) bases his test on the 1st decision.

The Anderson-Darling Test (AD):

The AD test is considered a Cramer-von Mises class statistic. The AD test also sorts the RNs in ascending order. It is based on differences between RN values. At a list position of i, the difference between the log of the RN at i and the log of the complement of the RN at a position j, is calculated, where j equals n+1-i. The difference is multiplied by (2i-1) and a sum of these products made. The sum is then adjusted for the size of n, and the resulting AD statistic converted to a p value, based on a standard table. (See D’Agostino and Stephens, 1986, chapter 4).

Wherever the KS test is given in the Diehard Output Report, the actual test done is the AD test.²

The Durbin statistic (C)

The C test is a difference class statistic. The C test takes blocks of n RNs, sorts them in ascending order, determines the difference (C+ or C-) between the RN value and the value $i/(n+1)$. The value used to test (C) is the larger of C+ or C-. It is very similar to the KS test but is slightly different. D'Agostino and Stephens (1986) give upper and lower distribution tail values for C in table 8.2. Discussions on the Durbin test in the newsgroups indicates that there is no closed-computer construct for the Durbin test to obtain p values. The best are these upper and lower limit tables, such as that given in D'Agostino and Stephens (1986)

The spacings statistic (D)

The spacings (D) in an ordered uniform sample from a uniform distribution. Given

$$D_i = U_i - U_{(i-1)}, \quad D_1 = U_1 \quad \text{and} \quad D_{(n+1)} = 1 - U_n$$

Then the D_i values have an exponential distribution $F(u) = 1 - \exp(-u/B)$, $u > 0$, with $B = 1/(n+1)$.

Since D'Agostino and Stephens (1986) there has been a lot of new papers on this statistic, L'Ecuyer and Simand (2004b) have gone extensively into these tests. The spacings module describes these tests. He uses G_i for D_i and the sum of $h(n * G_i)$ as being normally distributed. He also gives many current references on this method.

The Goodness-of-fit tests are independent, and frequently do not come up with the same p values. For sets of RNs (from different RNGs), under the KS criteria using the table 4.2 critical values in D'Agostino and Stephens (1986), the p values are not uniformly distributed, with very few p values below about 0.2, and many more above 0.8 than what would be expected. With AD tests, the p values are more uniformly distributed over the 0 to 1 interval. The AD test is preferred when the lack of fit is more pronounced at the beginning and ending of the sorted data.

The issue of just what does the p value mean needs to be addressed.

In D'Agostino and Stephens (1986) on page 105 (table 4.2) they give what they refer to as modified statistic T, from the D values. These essentially are D_+ or D_- values time square root of n. Large T values are tested as upper tail percentages, and small values are tested as lower tail percentages. For lower tail values, the issue of super-uniformity comes up (para 4.5.1 and para 8.5), where the

² Throughout the Diehard tests and references, Marsaglia uses the term "KS Test". In actuality all the tests are Anderson-Darling tests. Messages on the newsgroups from Marsaglia confirm this. This surfaced when I could not get agreement with the diehard report p values. However a brief check on birthday spacings test p values, confirmed that the Diehard test uses the AD test and the AD-statistic-vs-p values listed in table 4.3 of D'Agostino and Stephens (1986) are the ones used.

observations are too good to be true. That is the D- and D+ values are all small. The upper tail then is a measure of how far the data is from a uniform distribution.

The Diehard tests basically follow the same convention, a p value near zero indicates super-uniformity and a p value near 1 indicates non-uniformity.

In the TESTU01 series, a hypothesis test is made and p values close to 1 indicate excessive uniformity, and for p values close to zero, it is the opposite situation.

“If the distribution of Y is (approximately) continuous, p is (approximately) a U(0,1) random variable under H₀. Sometimes, this p can be viewed as a measure of uniformity, in the sense that it will be close to 1 if the generator produces its values with excessive uniformity, and close to 0 in the opposite situation.” They go on to state that this sometimes leads to problems, when discrete distributions are involved.

“When a p-value is extremely close to 0 or to 1 (for example, if it is less than 1E-10), one can obviously conclude that the generator fails the test. If the p-value is suspicious but failure is not clear enough (p=0.005, for example), then the test can be replicated independently until either failure becomes obvious or suspicion disappears (i.e. one finds that the suspect p-value was obtained only by chance). This approach is possible because there is no limit (other than CPU time) on the amount of data that can be produced by a RNG to increase the sample size and the power of the test.” (L’Ecuyer and Simand, 2004b, p 79-80)

TESTING EXCEL RNGS

Tests that were made on RAND, were all based on Microsoft’s disclosures of the actual equations.

Given the history on Excel errors, the question comes up, did Microsoft correctly implement the described equations in code? To answer this one has to extract RAND values directly from Excel and test these values. This is not easy. The problems in getting actual Excel RAND values out into a Diehard test file are as follows:

1. RAND is not a ‘WorksheetFunction’ object. RAND is not a vba accessible function. (For Excel 2000 the VBA function RNG returns RAND, for Excel 2003 and 2007 rng() is not defined in VBA), Therefore, for Excel 2003 and for 2007, there is no direct way to get a RAND FPRN directly in VBA. An indirect way of putting RAND in a cell, and copying the value to a sequential file however works, but it is slow.
2. The data sets for testing RNs, now require over 68 million numbers. In the indirect mode, building the file takes about 12 hours.
3. The diehard tests require a file of bits, where every 32 bit group directly represents the IRN(32) equivalent of a RAND FPRN, without any gaps. This requires conversion from FPRNs to IRN(32)s. Although the conversion from a FPRN to an IRN(32) can be done in VBA, there is a problem on putting the IRN(32) on a file. A sequential file can’t be used, because of the inserted line

adds the trailing two-byte ASCII 10 and 13 characters. A binary file using the Put command does not work, because the internal operation automatically insert a one byte control in front of each IRN(32) , creating gaps in the record. These files totally fail everything in Diehard.

The only method found was to convert the cell contents to an IRN(32), convert the IRN(32) to a hexadecimal word, and enter the hexadecimal word as a line on a sequential file. A program outside of Excel's VBA, was then used to generate the correct IRN(32) sequences from the sequential file.

EXCEL RANDOM NUMBER GENERATORS (PRNS)

Excel has three different PRNs, one the function RAND, another in the macro RANDOM and the random integer function RANDBETWEEN.

RAND was changed for the Excel 2003 version. All previous versions used the Excel 2000 version. These are two entirely different FPRNs with different properties. Therefore the reader should not take the comments made on the 2000 version as applicable to the 2003 version or vice versa. The only things in common were the test methods.

There was no change for the 2007 version.

RANDOM and RANDBETWEEN were not changed. They represent the Excel 5 routines.

RAND (EXCEL 2000)

DESCRIPTION

This is a very simple Pseudo-Random-Number-Generator (PRNG) requiring a seed or starting value. Excel provides a nominal starting value of 0.5. The output value is a computation on the previous value, which is an internal value, not the value of the cell above it. This internal value comes from the previous call to RAND which may be anywhere in the workbook. RAND is described in Microsoft Q86523 and is:

- A1) $v = 9821 * \text{previous value} + 0.211372$
- A2) $\text{RAND} = v - \text{FLOOR}(v)$
Previous value = RAND

This is similar to the linear congruential generator (LCG) that has the formal description (after Knuth, (1998) as

$$x(n) = (a * x(n-1) + c) \text{ mod } m$$

Where:

x(n) is the random number at the current value

x(n-1) is the previous random number

a is a multiplier, usually an integer, large and prime

c is the increment

m is the modulus (Modulus is an integer(m)-divide, multiply and subtract function that extracts the fractional part of the of the result of $a*x*(n-1)+c$).

In RAND:

a=9821. The constant a is not prime, but is the product of 3 primes, 7, 23 and 61
c=0.211372
m=1..

CHARACTERISTICS

The RAND-2000 function is not a true Linear Congruential Generator (LCG) as defined by Knuth. (1998). Knuth and others define it as the same formula, but with these restrictions:

1. $0 < m$
2. $0 \leq a \leq m$
3. $0 \leq c < m$
4. $0 \leq x(0) < m$
5. $x(n)$ is an integer

A true LCG generates random integers, which are then converted to a real number between 0 and 1. The RAND function differs in that conditions 2 and 5 are not met, and as a result the characteristics of RAND are different from true LCGs. The result of not meeting all the conditions, prevents one from applying Knuth's theoretical tests and making any conclusions on whether better values of a and c could have been selected by Microsoft for RAND.

In the implementation of RAND, the significant floating point number bits (out of 53 identified fractional bits) corresponding to the whole number (an integer) are set to zero and after FP normalization, the remaining least significant bits become the leftmost significant bits, and the rightmost (empty) bits become zero. This introduces a definite non-randomization of bits into the FP number, and is one of the reasons for RAND to fail some of the random bit tests.

A true linear congruential generator has a certain periodicity to the results. Marsaglia (1968) and Knuth (1998) show that this characteristics results in points lying on n-dimensional planes. This behavior shows when the RNs are given a spectral test (Knuth 1998, Vol. 2, pages 93-94)). Figure 18-1 shows the one-dimensional spectral grid for RAND-2000, based on 2000 consecutive calls from the 0.5 seed.

Figure 18-1: Spectral Test Results on RAND-2000

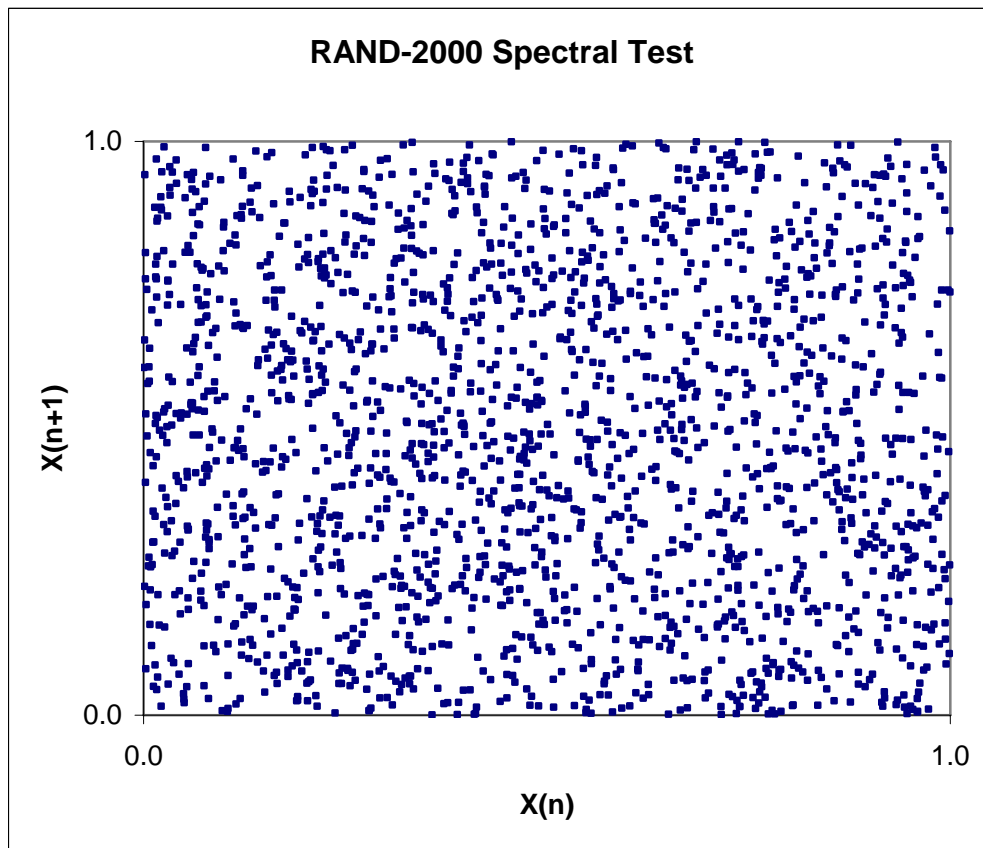
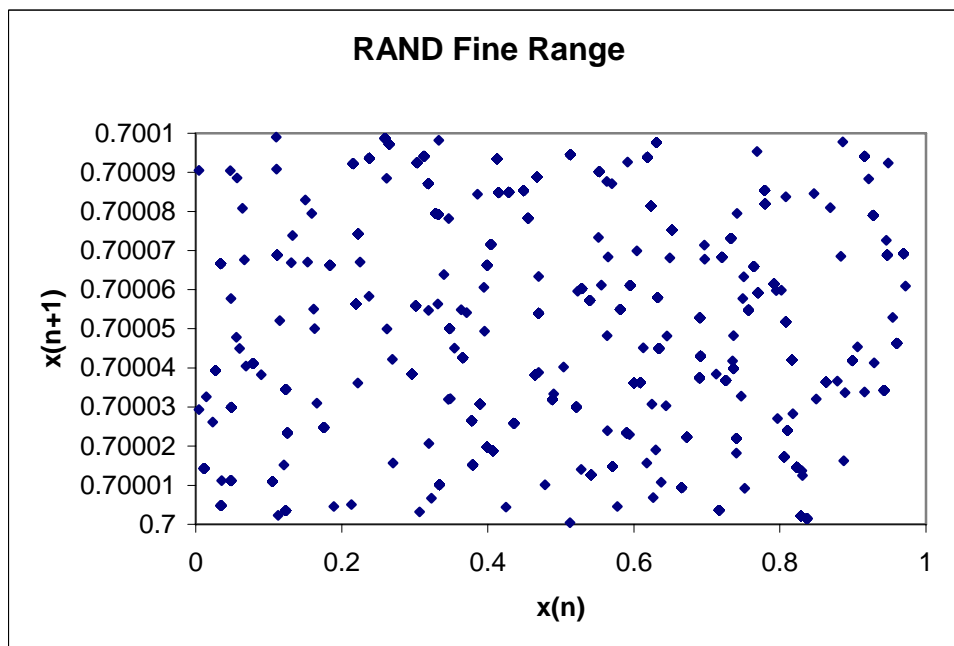


Figure 18-2: Spectral Test Results on RAND-2000, Fine Range



Both the coarse and fine plots show strong independence between successive calls to RAND-2000. There is no lattice structure visible.

Functions and subroutines written in VBA can call RAND-2000 as RND and call another function RANDOMIZE to set the seed (the starting value for RAND) to different values. With a fixed seed, the random number sequence is fixed. The RAND-2000 function has no arguments, and generates a new value every time the worksheet is updated by some new entry or change. Consequently the RAND cell values appear to be constantly changing. Microsoft recommends that you change the automatic update option (under tools) to manual, to avoid the confusion of constantly changing RAND values from automatic updates.

I tested the Excel worksheet RAND-2000 equation for about 3000 consecutive calls and found an exact fit to the equation (A1 and A2) for truly consecutive worksheet RAND values. There were several that were way off, but this was assumed to be out-of-sequence values (the internal Excel built-in sequence for recalculating cells). The output from RAND-2000 is 11 significant decimal digits 95% of the time. For the remaining 5%, the value has from 12 to 15 significant decimal digits. This gives a granularity of 1E-11 or better to each random value, allowing a large number of different numbers, at least in theory, 10^{11} different values. Actually there are 10^{17} different values as the result of added bits from normalization.

RAND-2000 is an irreversible PRNG, since multiple previous values can give identical outputs. For example for an output of 0.5, 21 difference values as given below will give the identical 0.5 output. Given the output, there is no way to predict the exact input value.

Table 18-3: Previous Values That Result in an Output of 0.5

0.013746333333333
0.061365380952382
0.108984428571428
0.156603476190483
0.204222523809522
0.251841571428572
0.299460619047635
0.347079666666652
0.394698714285713
0.442317761904773
0.489936809523814
0.537555857142853
0.585174904761935
0.632793952380931
0.680412999999970
0.728032047619054
0.775651095238093
0.823270142857175
0.870889190476127
0.918508238095168
0.966127285714250

Another characteristic is the large gain. For a given input value, small variations in this value will give large variations in the output value. This is shown in the following table.

Table 18-4: Effect of Small Changes in Previous Values on Output Values

Increment	Reference Value	Precursor Value Variation Interval	RAND-2000 Output for Reference Value	Output Difference from First Reference Output
0.00E+00	0.500000000	0.0000E+00	0.711326999999983	0.000000E+00
1.00E-08	0.50000001	4.7619E-10	0.711425210000925	-9.821000E-05
1.00E-07	0.5000001	4.7619E-09	0.712309099999402	-9.821000E-04
1.00E-06	0.500001	4.7619E-08	0.721148000000539	-9.821000E-03
1.00E-05	0.50001	4.7619E-07	0.809536999999182	-9.821000E-02
1.00E-04	0.5001	4.7619E-06	0.693427000000156	1.790000E-02
1.00E-03	0.501	4.7619E-05	0.532326999999896	1.790000E-01

What this table shows is that if any one of the allowable values in table 18-3 had a small perturbation, and generated a Reference Value as given in column 2 of table 18-4, then column 4 will be the succeeding output (two succeeding calls to RAND).

We have then the situation in which a small variation in a previous value will give large variation in an output value. An input variation of 4.76E-10 will give an output variation of 9.82E-05, a gain factor of 2.06E+05. It is this large gain factor that gives long periods.

RAND-2000 OUTPUT:

GRANULARITY:

Individual values are quite fine grained. The smallest difference between values was 9.07E-11, indicating a small level of granularity. Analysis of the differences suggests a granularity of possibly less than 6.85E-13. The former corresponds to a non-repeating set of 1.1E+10 calls and the latter corresponds to a non-repeating set of 1.5E+12 calls.

DISTRIBUTION

The output is a number between 0 and 1, uniformly distributed (it is not precisely uniform but lumpy uniform). For example, 65536 calls to RAND-2000 (one Excel column) the values roughly fit a uniform distribution with a KS Test D^+ of 0.003287 and a D^- of -0.0022547. The corresponding K values for a KS test are $K_n^+ = 0.84$ ($p=0.74$) and $K_n^- = 0.58$ ($p=0.50$). Actual p values for smaller blocks show the following characteristic. (Note: D'Agostino and Stephens (1986). Table 4-2 test statistic versus p values for the left hand tail (used in tables 18-5 and 18-6 for minimum values) are highly questionable as to being correct values.)

Table 18-5: KS Test p Values for Smaller Block Sizes From 65,536 RAND-2000 RNs

Block Size	Number of Blocks	Maximum p Value	Minimum p Value
100	655	0.9979	0.1943
200	327	0.9995	0.2110
500	131	0.9973	0.2362
1000	65	0.9988	0.3645

The high p values in the third and fourth columns are typical of all “good” PRNG outputs. For example, the outputs from another PRNG that is considered “quite good” and passes the Diehard and other RN tests.

Table 18-6: KS Test p Values for Smaller Block Sizes From 65,536 MWC256 RNs

Block Size	Number of Blocks	Maximum p Value	Minimum p Value
100	655	0.99845	0.12029
200	327	0.99742	0.19493
500	131	0.99705	0.32796
1000	65	0.99486	0.28168

Results of a GOF test on the 65,536 RAND values are shown below

Table 18-7: p Values of GOF tests on 65,536 RAND RNs

Number of slots	Seed=0.5	Seed=0.190832	Seed=0.383932
100	0.0140	0.2295	0.4871
1,000	0.2529	0.2569	0.5357
10,000	0.6751	0.0413	0.1804

Both tests show that RAND gives p values typical of PRNGs with regard to uniformity. This also shows why the GOF test is preferred for testing PRNG outputs (it’s weaker in outcome).

RAND-2000 PERIOD:

The fine granularity would suggest a long period. However that is not the case. The actual period is 16,777,216 calls, and this count is the same for any preset seed using Randomize.

THE DIEHARD TESTS

I ran the diehard test suite on several sets of random numbers using RAND. I used the RAND algorithm described above (A1 and A2) to generate 3 million random double precision floating-point numbers each time. Each number was converted to a 32-bit string as shown in Note AA. The string was written to a binary file in sequence. The binary file was directly readable by DIEHARD.EXE. This was in accord with Marsaglia’s instructions on the Diehard inputs.

Table 18-8 is a summary of Altman’s (2000) and McCullough’s (1997) conclusions, together with my results on the original RAND-2000 and two modified RAND-2000 outputs that show improvements in the randomness. The Diehard output report for the normal RAND-2000 output is given as Note AB.

Table 18-8: Results of Diehard Tests on RAND-2000

Test Reference Number	Altman (2000)	McCullough and Wilson (1999)	Normal RAND Output
1	P	P	P
2	P	P	F (.9998)
3	P	P	P
4	P	P	P
5	P	P	P
6	P	P	P
7	F	F	F (1.0)
8	F	F	F (1.0)
9	F	F	F (1.0)
10	F	F	P
11	P	F	P
12	P	P	P
13	P	P	F (1.0)
14	P	F	P
15	P	F	P
16	P	P	P
17	P	P	P
18	P	P	P
Diehard F	2/15	4/15	3/15
Total F	4/18	7/18	5/18

Marsaglia considered the three monkey tests (7, 8 and 9) as one test (Marsaglia 1993), since they had the essentially the same test and the same output analysis with the difference being the group size. Generally when there was a failure from the RNG here, all three failed, not just one. He also considered 3 and 4 as being one test. The difference is that in the 31 bit words, the lowest bit is left out. For 32 bit machines, any difference here was not considered significant. These arguments were the reason for providing the two separate outcome assessments at the bottom.

The explanations in the diehard output (See Note AB) on the outputs are limited. Marsaglia (1985) and Marsaglia (1993) provide some very limited additional explanations. This explains somewhat the misinterpretation of DIEHARD outputs found in the literature. One could say that RAND-2000 fails 3 or 5 Diehard tests, depending on how you counted the number of tests, and interpreted the importance of the test. This whole area of evaluating PRNGs is very subjective and overlaid by software vendors and developers selling “a better mousetrap.” One can’t evaluate claims without disclosure.

There are certain faults in the RAND-2000 bit sequences that make the sequences consistently fail Diehard tests 7, 8 and 9. Knuth (1998) on page 75 refers to this test failure as the “distribution is a bit too ‘flat’ to be random”. This particular test is of importance where cryptography issues are of concern. This basic failure of RAND-2000 cannot be corrected by using modifications to RAND-2000 (such as generating Fibonacci

series). Considering how RAND is used in Excel, the irregular bit sequences found by the Diehard monkey tests 7, 8 and 9 are not considered to be a bar against use of RAND-2000.

Variations on RAND-2000 were explored. These were simple variations that could be done on an Excel worksheet. Variations in seed resulted in minor changes in the test 2, p-values and no changes in the test 7, 8, 9 and 13, p-values. Table 18-9 gives the results of some of these variations that looked promising.

Table 18-9: Results of Diehard Tests on Variations to RAND-2000

Test Reference	Every 3 rd RAND call.	Every 3 rd RAND Call, Different Seeds	Every 7 th RAND Call	Fibonacci Addition, 15/7/15
1	P	P	P	P
2	P	P	P	P
3	P	P	P	P
4	P	P	P	P
5	P	P	P	P
6	P	P	P	P
7	F (1.0)	F (1.0)	F (1.0)	F (1.0)
8	F (1.0)	F (1.0)	F (1.0)	F (1.0)
9	F (1.0)	F (1.0)	F (1.0)	F (1.0)
10	P	P	P	P
11	P	P	P	P
12	F (.9998)	F (.9901-.9999)	P	P
13	F	F	P	P
14	P	P	P	P
15	P	P (.96-.98)	P (.99993)	P
Test Reference	Every 3 rd RAND call.	Every 3 rd RAND Call, Different Seeds	Every 7 th RAND Call	Fibonacci Addition, 15/7/15
16	P	P	P	P
17	P	P	P	P
18	P	P	P	P

For the every-other-3rd RN-procedure, set up three columns with =RAND, doing equation copy across a row, then down the columns. When done, do a copy of columns 3 and do a paste special (value) to an empty column. This will give you a little better random set of numbers. The Fibonacci method gives the best (except it still fails the monkey tests), but it is more complicated to set up.

For the use in general numerical applications such as the majority of applications with Excel, tests 1, 12, 13, 14, 15, 16 and 17 are the important tests that RAND should pass. RAND-2000 as is, does not pass test 13. If this is an important consideration, then the modified RAND (every 7th) should be used, since it will pass these 7 tests.

The better PRNGs will pass all of the tests. All of Marsaglia's PRNGs pass these Diehard tests. It is easy to program one of Marsaglia's recent RNGs (mwc256) as a VBA function

in Excel. It will generate random numbers that will pass the Diehard suite and all the new PRNG tests (so it is claimed). The VBA code for it is in Note AC.

PRNG random numbers in some cases are very poor at solving multiple dimension integration problems. A PRNG that passes all Diehard tests for randomness may be useless in some Monte Carlo and integration problems, because it does not “cover the field of dimensions” adequately or is not sufficiently uniform to adequately cover the space. (NAG 2001). PRNs that have a tendency toward super-uniformity are preferred here.

RAND (EXCEL 2003 AND 2007)

THE MICROSOFT VERSION

Microsoft completely changed RAND for Excel 2003. It was carried over to the 2007 version. KBA 828795 describes the change made in RAND. They refer to it as the Wichmann-Hill algorithm (KBA 828795). The term WHA is used as an abbreviation.

Microsoft wrote, “The algorithm that is implemented in Excel 2003 was developed by B.A. Wichman and I.D. Hill. This random number generator is also used in the RAT-STATS software package that is provided by the Office of the Inspector General, U.S. Department of Health and Human Services. It has been shown by Rotz et al to pass the DIEHARD tests and additional tests developed by the National Institute of Standards and Technology (NIST, formerly National Bureau of Standards)” (KBA 828795). The period is 2^{43} .

L’Ecuyer and Simard (2004) refer to it as “unif01_Gen * ulcg_CreateCombWH3” and a version by Sobel and Leviton (1999) as “unif01_Gen * uvaria_CreateSobol199”.

The following are three statements about the code.

1. KBA 828795 describes the algorithm:

```
IX, IY, IZ should be set to integer values between 1 and 30000 before first entry
IX = MOD(171 * IX, 30269)
IY = MOD(172 * IY, 30307)
IZ = MOD(170 * IZ, 30323)
RAND = AMOD(FLOAT(IX) / 30269.0 + FLOAT(IY) / 30307.0 + FLOAT(IZ) / 30323.0, 1.0)
```

2. The ACM source gives:

```
// Returns a pseudo-random number rectangularly distributed
// between 0 and 1. The cycle length is 6.95E+12 (See page 123
// of Applied Statistics (1984) vol.33), not as claimed in the
// original article.
// IX, IY and IZ should be set to integer values between 1 and
// 30000 before the first entry.
// Integer arithmetic up to 30323 is required.
whIx = (171 * whIx) Mod 30269
whIy = (172 * whIy) Mod 30307
whIz = (170 * whIz) Mod 30323
random = whIx / 30269# + whIy / 30307# + whIz / 30323#
If random > 1# Then random = random - 1#
If random > 1# Then random = random - 1#
```

3. The current Wichmann-Hill Code (McCullough 2008)

```
F Real function random()
C Algorithm AS 183 Appl.Statist.(1982)vol 11, no. 2
C Returns a pseudo-random number rectangularly distributed between 0&1
C TX,IY and IZ should be set to integer values between 1 and 30000 before the first entry.
C Integer arithmetic up to 3023 is required
F integer ix,iy,iz
F common/randc/ix,iy,iz
F1 ix=171*mod(ix,177)-2*(ix/177)
F1 iy=172*mod(ix,176)-35*(iy/177)
F1 iz=170*mod(ix,178)-63*(iz/177)
F1 If (ix.lt.0) ix=ix+30269
F1 If (iy.lt.0) iy=iy+30307
F1 If (iz.lt.0) iz=iz+30323
C If integer arithmetic up to 5212632 is available use F2 instead of F1
F2 ix=mod(171*ix,30269)
F2 iy=mod(172*iy,30307)
F2 iz=mod(170*iz,30323)
C Either uses F3 to end
F3 random=amod(float(ix)/30269+float(iy)/30307+float(iz)/30323,1.0)
```

Essentially the three versions are identical, but there are differences. A critical difference is how the mod (or modulus) function actually works and which numbers are integers and which numbers are floating point. Knuth (1) on page 39 defines the mod operation in terms of the “ceiling” and the “floor” of a real number.

Floor(x) = the greatest integer less than or equal to x

Ceiling(x) = the least integer greater than or equal to x

The mod operation is defined as :

$$X \text{ mod } Y = X - Y * \text{Floor} (X / Y)$$

The MOD function in the ACM source is: X mod Y

The MOD function in Fortran and Excel is: mod (X, Y)

The MOD function with X and Y positive, returns a zero or a decimal value less than 1

$$\text{MOD} (n, d) = n - d * \text{INT} (n / d) \text{ \{As defined by Microsoft\}}$$

This is not easily and correctly implementable in all cases. The FLOOR function should return an integer having any number of digits. The MOD function should return a floating point DOUBLE. The numbers in Excel are not identifiable as to being either integers or floating points. There are no Excel IF functions that identify a value as being integer or floating point. The worksheet MOD function returns a floating point number.

For random numbers, the INT(n / d) should be a bit operation on unsigned 16 or 32 bit integers. Excel and Visual Basic do it on the left 31 or 63 bits of an integer. As a result there is some inequalities that show up in classical testing, based on 32 or 64 bit registers.

MOD does not return an integer, it returns a decimal fraction as a double. When a mod is done according to Knuth, it is all by integers and there is no round-off. When done by use of the INT function there is round-off not the required truncation.

The " whIx / 30269# + whIy / 30307# + whIz / 30323#" calculation is done using doubles.

If random > 1# Then random = random - 1#

If random > 1# Then random = random - 1#

The implementation of the new equations into an algorithm was however incorrectly done (when Excel 2003 was released late in 2003). Frequently negative random numbers appeared when RAND-2003 was called. This is described in KBA 834520. Microsoft issued a hot fix back in January 2004 with the corrected algorithm and finally incorporated it into the first upgrade (KBA 834691, SR-1) to Excel 2003.

Microsoft never gave any information about the cause of this error.

I implemented the basic WHA (as shown above) on a worksheet to explore the actual behavior of the algorithm. The MOD function (a double) was used. Variations on seed values (both + and -) did not result in any of the WHA values being negative. The basic algorithm in the form of worksheet equations will compute output values in the range of 0 to 1, given seed values between $-2.3E+10$ to $+2.2E+10$. Outside of this range "NUM" is returned. During the exploration of allowable seed values, in no cases, were negative WHA values observed. Therefore it is not clear why negative values were once returned.

The succeeding WHIx, WHIy and WHIz values never exceed the modulus values of 30269, 30307 and 30323, respectively, and appear to be always >0. The AMOD function is not in Excel or in Visual Basic, so it is not clear exactly how Microsoft converts the calculated sum (<3 and >0) to a positive value between 1 and 0.

A seed value of 0 results in constant WHI zero values. The function will still output random values unless all 3 seeds are zero, and in this case, the output will always be zero.

IS THE ALGORITHM THAT MICROSOFT USES THE TRUE WICKMAN-HILL ALGORITHM?

McCullough in a recent paper (Microsoft's "Not The Wichmann-Hill" RNGs.) (McCullough 2007B) shows that what actually comes out of the RAND() worksheet function is not the true Wichman-Hill sequence (RE: algorithm 2). He shows that the WH sequence (using the Zeisel recursion in R) is equal to the direct WH recursions in R. He also shows that the Excel sequence with any given starting seed does not follow the Zeisel recursions. Considering the different algorithms, this is expected.

The difficulty in implementation in Excel is the fact that IX, IY and IZ are both inputs and outputs of the basic function. This is contrary to the basic Excel requirement that only one value can be output from a function. The IX, IY and IZ values that are the inputs to the function can never be identified in terms of a "constant" value. The way that Excel sets up the true, actual computing sequences from cell to cell, is entirely internal and results in changes to p values every time another cell computation involvint RAND is encountered. Consequently one can never identify the IX, IY and IZ seed values associated with a sequence of RAND values. Consequently you can never identify which

cells on the worksheet were actually a true sequence pair. The exception of course is when there are only 2 RAND calls in the entire workbook, and these are visible.

SETTING THE SEED

One issue here is the setting the seed when the series are started (previous IX, IY and IZ values). Microsoft gave no clue how this is done. Sobel and Leviton (1999) specify that the seed sequence should start from 5, 11 and 17. In Excel 2003 and 2007, there is no way to set this seed. The fact that a user cannot set a seed vector, and Microsoft's error in returning negative values (KBA 834520), has raised the question in the e-mails and discussions as to whether the 2003-2007 RAND is truly the W-H algorithm.

In Excel 2003, the VBA function RANDOMIZE (which changed the seed in Excel 2000) does not work. VBA recognizes this as a valid function, but it does not affect the RAND-2003 outputs.

PERIOD

Microsoft claims a period in excess of 10^{13} for RAND-2003.

The brute-force method of taking one RN and then cycling through repeated calls to the RNG until one of the RN's equal the selected reference RN, is one method of finding out the period. For RAND-2003, the problem is that the RNG is slow. In running $1E+09$ RAND-2003 calls, it took about 15 hours. In this particular sequence, a new RN equal to the starting RN was not found, indicating that the period is greater than 10^9 .

DIEHARD TEST RESULTS

The RAND equations in Excel 2003 (with the SR-1 upgrade) will generate sets of random numbers that pass the Diehard-II and Diehard-III tests. Note AD is the Diehard report for RAND-2003. The section "DIEHARD-II OUTPUT, WH EQUATIONS" is the Diehard-II report on values computed from the WH equations. The section "DIEHARD-II OUTPUT, ACTUAL EXCEL CELL RAND VALUES" is the Diehard report on values extracted from an Excel sheet cell with the cell equation "=RAND()". The section "DIEHARD-III OUTPUT" is the Diehard-III report on values computed from the WH equations (Algorithm 1).

From the results, we can conclude that:

1. All the Diehard tests are passed.
2. Marsaglia's statement from the Diehard report applies:

"Most of the tests in DIEHARD return a p-value, which should be uniform on $[0,1)$, if the input file contains truly independent random bits. Those p-values are obtained by $p=F(X)$, where F is the assumed distribution of the sample random variable X---often normal. But that assumed F is often just an approximation, for which the fit will likely be worst in the tails. Thus you should not be surprised with occasional p-values near 0 or 1, such as .0012 or .9983. When a bit stream really FAILS BIG, you will get p's of 0 or 1 to six or more places. By all means, do not, as a Statistician might, think that a $p < .025$ or $p > .975$ means that the

RNG has "failed the test at the .05 level". Such p`s happen among the hundreds that DIEHARD produces, even with good RNGs."

The high p value (0.99155) on the 31x31 Rank Test is such a random occurrence.

3. Microsoft correctly fixed the negative PRN problem by the hot fix (KBA 834691). In over 2 gigabytes of FPRN values tested from the cell equation '=RAND()', no values were less than zero and no values were over one.

TESTU01 TEST RESULTS

McCullough and Wilson (2004) state that RAND-2003 will not pass TESTU01. They give the following information on test failures.

9-Multinomial Bits Over	p=0.9958
10-Birthday Spacings (t = 2)	p<1E-15
11-Birthday Spacings (t = 4)	p<1E-15
13-Birthday Spacings (t = 13)	p=2.4E-03
14-Close Pairs (t = 2)	p<1E-15
15-Close Pairs (t = 4)	p=3.6E-14

P values close to zero indicate "super-uniformity" or "to good to be true". McCullough and Wilson's (2004) criteria for test failure is "Four failures, four p-values less than 1e-10. If any had been greater than 1-1e-10, those would have been counted as failures, too." "I did not take 0.01 to 0.99. The program flags values that are outside that interval. There were six outside that interval." (as listed above). To a question of whether there were any repetitions, McCullough replied, "Not necessary, as there were failures with "eps" and rerunning isn't going to change those to successes. If I had been interested in whether it fails a specific test, rather than whether it fails any test at all, I might have rerun the 2.4e-3 or 0.9958. I have with TESTU01 done this for other generators and in my admittedly limited experience (I've analyzed perhaps 20 RNGs) these values have always "disappeared", i.e., if I were to rerun the 2.4e-3 would change to something inside the [0.01, 0.99] interval." (McCullough 2005)

The TESTU01 series is based on the literature WH algorithm (See above under "3. The current Wichmann-Hill Code (McCullough 2008)") in C code. This form did not pass the Birthday Spacings test, but the version in Excel (as a string of random numbers) did in Diehard II. This tends to support the view that the Excel version is not the "3. The current Wichmann-Hill Code (McCullough 2008)" listing. Although L'Ecuyer claims that his Birthday Spacings test is the same as the Diehard Birthday Spacings test, there is an obvious difference in test results.

L'Ecuyer and Simand (2004b) provide the following information on these tests:

"The 3 parameters N, n and r are common to each test function. The parameter r gives the number of the most significant bits that are discarded from each generated random number. S is another parameter that represents the bits of each uniform that are effectively used by the test." (S is presumed to be set in the generator module.)

The TESTU01 BirthdaySpacings test is identical³ to the Diehard Birthday Spacings test, where t represents the number of dimensions.

The ClosePairs test is similar in concept to the Diehard Spheres test.

The Multinomial Bits Over test is similar to the Diehard Minimum Distance Test

There are substantial differences. L'Ecuyer and Simand (2004b) essentially take Marsaglia's tests to higher dimensions. The issue here is about the relevance of the p-values from these higher dimensions.

Also the programming languages are different, Fortran and C++ when compiled; do not generate the same machine language code sequences. Also there is a conceptual translation from the mathematical expressions to actual programming language constructs, and this is different, because the two languages are quite different in structure and in application. Programming involved two entirely separate (conceptually) groups (Florida, Montreal), each having different programming standards and directions.

L'Ecuyer and Simand (2004b) state:

“When a p-value is extremely close to 0 or to 1 (for example, if it is less than $1E-10$), one can obviously conclude that the generator fails the test. If the p-value is suspicious but failure is not clear enough ($p=0.005$, for example), then the test can be replicated independently until either failure becomes obvious or suspicion disappears (i.e. one finds that the suspect p-value was obtained only by chance). This approach is possible because there is no limit (other than CPU time) on the amount of data that can be produced by a RNG to increase the sample size and the power of the test.” (p. 80)

There will be users of Excel 2007 who will try to use RAND for a security application. The inability of being able to set a seed, makes it almost impossible to use the sequence for passwords and other security controls. Also the inability of accessing RAND(2003) in vba also is a bar to using RAND(2003) for any cryptology or security purposes. Also the fact that we cannot prove that RAND(2003) is an exact implementation of the WHA, also is a bar against its use for security applications.

Given the inability to get repeatable RAND(2003) sequences or to set RAND(2003) sequences, means that RAND(2003) usage will always be a problem.

Also given the facts:

- Excel users commonly use RAND for short sequences (less than 132,000 calls).
- Excel users cannot use RAND(2003) in a VBA module, unless they write the K-H-code as a function or subroutine. If they write the WHA as a vba function, then we say that because the WH fails TEST01, it should not be used.
- It would be impossible to use RAND(2003) for any consistent, manageable security or cryptology application.

³ Really not identical. See footnote 1

- Excel (as-is) is not structured for advanced statistical or simulation problems. It certainly can be used for Markov Chain Monte Carlo simulations, but the size limitations built into Excel (pre 2007 versions), limit the applications to small sets. Here the sets would be constantly changing, making it impossible to retain stable outcomes.

The fact that the WHA failed the TEST01 set does not seem to be a bar here to its actual usage. Test failure was not any bar to the use of the old Excel 2000 RAND.

Since both the Diehard and NIST tests were passed, RAND(2003) is considered an acceptable RNG for Excel users, considering the pre-2007 built-in Excel Worksheet limitations on sequence lengths. Since RAND(2003) can't be called in VBA, there is no easy way to generate large sets of random numbers using RAND(2003). A far better solution would be to write in a better RNG.

The WH algorithm will tend to always result in test failures of some kind.

This brings out my real concern, the fact that TESTU01 can't work on actual random numbers from a specific software package, which may not truly follow the TEST01 C coded sequences.

RANDBETWEEN

This is an unusual function that returns only whole numbers (integers) between a lower value and an upper value

The input lower value V1 is converted to a whole number by the CEILING function to N1 and the upper value V2 converted to a whole number by the FLOOR function to N2. The RANDBETWEEN function returns random whole numbers between N1 and N2, including N1 and N2 values. Both N1 and N2 can exceed long integer limits. If N2 is less than N1, the error #NUM appears. Negative V1 and V2 values can be input, and appropriate random numbers between them will be output in accordance with the conventional view of positive and negative numbers on a horizontal X axis.

The RANDBETWEEN function was not separately tested⁴. McCullough et. al. did not test it, Therefore it remains an unproven random number generator. There were no reports found that stated failures in the expected sequence.

DATA ANALYSIS TOOL PAC ROUTINES (EXCEL 2000 AND EXCEL 2003)

These are the provided routines that have not changed since Excel version 5, and apply to any version since then. One has to be very selective on using them, since **they are faulty.** **Microsoft admits that this a faulty algorithm, but will not fix it.**

⁴ The Excel VBA is different from Microsoft's Visual Basic (or Visual Studio). One cannot in VBA build the proper Diehard binary files, since the PUT command (which works) ends up putting additional bits in after the hexadecimal, which corrupts the bit sequences. The nested binary bits (as hexadecimals) in sequence are corrupted, and Diehard fails to properly run the tests. Its very time consuming to build up a sequenced floating point file, exit Excel then go into Visual Basic to generate the proper bit sequences for a file equivalent to 68,000,000 32 bit integers, and then go into DOS to run the tests.

RANDOM

This is a Data Analysis Tool-Pac routine. When used, a message box appears for inputs.

NUMBER OF VARIABLES

Enter the number of columns of values you want in the output table. If you do not enter a number, Microsoft Excel fills all columns in the output range you specify.

NUMBER OF RANDOM NUMBERS

Enter the number of data points you want to see. Each data point appears in a row of the output table. If you do not enter a number, Excel fills all rows in the output range you specify.

DISTRIBUTION

Click the distribution method you want to use to create random values.

Uniform

Characterized by lower and upper bounds. Variables are drawn with equal probability from all values in the range. A common application uses a uniform distribution in the range 0...1.

Normal

Characterized by a mean and a standard deviation. A common application uses a mean of 0 and a standard deviation of 1 for the standard normal distribution.

Bernoulli

Characterized by a probability of success (p value) on a given trial. Bernoulli random variables have the value 0 or 1. For example, you can draw a uniform random variable in the range 0...1. If the variable is less than or equal to the probability of success, the Bernoulli random variable is assigned the value 1; otherwise, it is assigned the value 0.

Binomial

Characterized by a probability of success (p value) for a number of trials. For example, you can generate number-of-trials Bernoulli random variables, the sum of which is a binomial random variable.

Poisson

Characterized by a value lambda, equal to 1/mean. Poisson distribution is often used to characterize the number of events that occur per unit of time— for example, the average rate at which cars arrive at a toll plaza.

Patterned

Characterized by a lower and upper bound, a step, repetition rate for values, and repetition rate for the sequence.

Discrete

Characterized by a value and the associated probability range. The range must contain two columns: The left column contains values, and the right column contains probabilities associated with the value in that row. The sum of the probabilities must be 1.

PARAMETERS

Enter values to characterize the distribution selected.

RANDOM SEED

Enter an optional value from which to generate random numbers. You can reuse this value later to produce the same random numbers.

OUTPUT RANGE

Enter the reference for the upper-left cell of the output table. Excel automatically determines the size of the output area and displays a message if the output table will replace existing data.

NEW WORKSHEET PLY

Click to insert a new worksheet in the current workbook and paste the results starting at cell A1 of the new worksheet. To name the new worksheet, type a name in the box.

NEW WORKBOOK

Click to create a new workbook and paste the results on a new worksheet in the new workbook.

WHAT THE ROUTINE ACTUALLY DOES

For generation of random values from probability distributions, using the macro RANDOM, Excel uses an entirely different algorithm, and one that really gives bad values. It basically is a shift operation on an integer (16 bits, with 1 sign giving numbers between -32768 and $+32767$) rather than doing it at the machine level where shift operations are easy to do. Excel takes the uniform random integer between 0 and 32768 and converts it to a floating-point number within the desired range. We can roughly say that the period is 2^{+15} , which is very small based on contemporary applications of random numbers. When floating-point numbers between 0 and 1 are generated, they exhibit severe granularity, being made up of increments of $3.05185094759972E-05$. Therefore this method should not be used except for very crude estimations.

RANDOM does not lend itself to the generation of the large diehard test input files needed. As a result, the routine was not given any diehard tests. RANDOM is not accessible in VBA.

DO NOT USE THE RANDOM ROUTINE

The faults and errors in the random numbers generated by the Data Analysis Tool-Pak have not been fixed for Excel 2003. It should be clear that these should not be used under any circumstance. They are very likely to give unsuspected faults, even for small sets. This problem is listed frequently on the newsgroups.

SAMPLING

The “Help” selector here returns, “The Sampling analysis tool creates a sample from a population by treating the input range as a population. When the population is too large to process or chart, you can use a representative sample. You can also create a sample that contains only values from a particular part of a cycle if you believe that the input data is periodic.”

“For example, if the input range contains quarterly sales figures, sampling with a periodic rate of four places values from the same quarter in the output range.” Clearly this help is of almost zero help and does not define what it does. The input requires an input range, a sampling method (periodic or random), and the standard “output option” on where to put the result. The criteria for a random method is not defined. It is assumed the defective Data Analysis random number generator is used (the “Random” routine described above).

Wherever possible this routine should not be used for random samples. The periodic method just selects from the input range every other “n”th value, where n is the input period.

For any serious sampling, the three Algorithms from Knuth (1998) (Section 3.4.2) should be used. The first, algorithm S, generates a correct random sample from a “column of values” that is a fixed length N. The second, algorithm R, generates a sample of n values from a column having an unknown number of values. The third (algorithm P) generates a random shuffling (sequence) of a column of values. In each of these three, a subroutine (macro) has to be written in VBA with input boxes, giving the range of values, the range for the output values, and the sample size. A simple cell function will not work.

RETROSPECT ON RANDOM NUMBER SEQUENCE TESTS

One should note that Marsaglia and L’Ecuyer use different criteria for determining acceptance/rejection of a random number generators. Marsaglia’s criteria is more relaxed. It is emphasized that the criteria for determining acceptance/rejection is set arbitrarily by different academics. One tests the bit sequence as sequence of unsigned integers (generated by a vba code) and the other as a C++ algorithm. One should always recognize that the “experts” do not speak with one voice or even agree. They are the philosophers in the Agora arguing among themselves. Determining what p values are to be in the rejection region is the responsibility of the investigator. Fisher said that after all, he is the one who knows best.

(See also Marsaglia (2002) for a more recent view on RNG performance. He questions the importance of some of the current L’Ecuyer criteria used to assess a RNG.)

FUTURE VERSIONS OF RAND

DOES RAND HAVE TO BE THE LATEST MODEL

McCullough keeps pushing for better RNGs in Excel. The question should be, is a better RNG really necessary. Stated another way, given the limitations on a given worksheet (for Excel 2003, no more than 65,536 cells, for Excel 2007 no more than 1,048,576 cells (i.e. one function per cell per column)), large random number sets cannot be effectively captured. With the recalculation features, the set changes, but always represents an essentially contiguous set of RNG outputs. An RNG with a very large period is really not required. One with a minimum period of at 2^{40} would be acceptable.

The RAND-2003 RNG represents technology of the 1980’s. There has been a lot of work on deriving better random number generators in the late 1990’s and early 2000’s. Most of this work has been to obtain longer periods, faster generation, greater difficulty in being able to find the mechanism (break the code) when used in encryption, and to pass some of the more recent random testing methods.

POSSIBLE FUTURE VERSIONS WITH BETTER CAPABILITIES

McCullough argues that the seed of the RNG should be accessible, so that a fixed sequence can be obtained for reproducibility. The better RNGs use vectors of seed values, and as such cannot be readily preset. The RAND(2003) algorithm uses a vector of 3 values, and Microsoft could not find a way to preset these in the same fashion as the RAND(2000) seed (e.g. through a VBA function). Resetting the starting seed to selected starters would require identifying a column of vector values and a new Excel function. It also would open the “barn door” to its improper use in security applications.

For future versions of Excel, a better RNG should be considered. Marsaglia (2002) recommends “the complementary-multiply-with-carry (CMWC) types, since they are very fast, can have incredibly long periods and pass tests of randomness as least as well as, and often better than, other kinds of RNGs.” L’Ecuyer recommends his type of RNGs, which differ from Marsaglia’s approach. McCullough (2004) recommends the Mersine Twister. These however all depend on the ability of the programming language to be able to do register shifts, multiplications on unsigned long and long long integers and have full 64 bit register capabilities. VBA does not have this capability, nor is it fast enough, and as a result, these better algorithms will probably never be implemented in Excel.

The author’s view is that if RAND passes the NIST tests and passes Diehard-III, then it is a “decent rng” for use in Excel 2003, but questionable for Excel 2007 and future versions.