FORWARD COMPUTING and CONTROL PTY. LTD.

Investigation of GAUSS' Random Number Generators

18th June 2001

prepared for

Aptech Systems, Inc.

by

Dr. M. P. Ford

and

D. J. Ford

NSW Australia

ABN 11 003 669 994

Table of Contents

Investigation of GAUSS' Random Number Generators

18th June 2001

Introduction

This paper reports on the tests of the random number generators provided in Gauss V3.6. Gauss V3.6 contains two random number generators:- a slightly revised version of the linear congruential one tested by Vinod(Vinod 2000), rndi, and a new recur-with-carry generator, rndkmi. The new generator, rndkmi, passes all the DieHard tests (Marsaglia 1996) and has period of greater then 10^{8888} and dimension of greater than 920. It is recommended that rndkmi be used whenever random numbers are required.

The outline of this report is:-

A brief introduction to random number generators will be given together with a discussion of the period of a random number generator and necessity of testing random number generators. This will be followed by a description of the DieHard suite of test (Marsaglia 1996). Then the two random number generators provided in Gauss V3.6 will be described and the results of the DieHard suite presented. This will be followed by a discussion of the results. Appendix A contains the Gauss code used to generate the input data for the DieHard tests.

What do we mean by a "Random Number"

It is still being debated exactly what a random sequence of numbers is. Computers use pseudo-random number generators (RNGs), which although not truly random have the desired properties of a sequence of random numbers. In fact, in the interests of reproducibility of results, deterministic methods are preferable.

The generally accepted qualities desired by RNGs are (Ripley, 1990)

i) be reproducible from a simply specified starting point.

ii) have a very long period.

iii) produce numbers which are a very good approximation to a uniform

distribution.

iv) produce numbers very close to independent output in a moderate number of dimensions.

The output of a RNG is usually a sequence of 32- or 64-bit integers. There are two ways to view such sequences of numbers. The first, is to consider them as truncations of real numbers in the range (0,1), with a uniform distribution. The second is to view the sequence of integers as a sequence of random bits. Which view one takes depends on the use to which the sequence is being put. Does the sequence represent random positions on a line, or a sequence of coin tosses?

Ideally, a RNG should cater for both these uses. However, viewing the output as a random sequence of bits is more demanding as all the bits of each integer, even the 'least significant', are equally important. This can be seen in several cases of RNGs which pass tests viewing the sequence as truncations of random real numbers in (0,1), but fail bit-stream tests as the less significant bits of the 'random' integers are not very random at all.

Period of Random Number Generators

Almost all RNG's are (more or less) *cyclic* (or periodic). That is to say, they will output a sequence of integers which repeats itself exactly after a number of steps. The length of the part which is repeated is called the *period* of the sequence. To be precise, if we have a sequence x(1), x(2), x(3),.... which is periodic with period k, then x(n+k) = x(n) for all n greater than 1. Many generators also have a small leading 'tail' of integers which do not form part of the repeating part of the sequence. In this case x(n+k) = x(n) once n is larger than the length of the 'tail'.

It is essential that the period of the generator be larger than the number of random numbers to be used. Modern applications are increasingly demanding longer and longer sequences of random numbers, for use in Monte-Carlo simulations and 'boot-strap' methods for example. Some methods require not only millions of random numbers, but billions. Generators with a small period of only 2^{31} or 2^{32} are therefore of limited use.

Tests of Random Number Generators

Good pseudo-random number generators are hard to design, and the literature is littered with RNGs which appeared good at the time, but were later found to fail even the most simple of tests. It is therefore desirable to test the suitability of a RNG with as many tests as possible.

Most hard RNG tests require some advanced probability and statistics theory, as well as some difficult calculations. It is therefore difficult to perform such tests on ones own. Fortunately, a quite comprehensive suite of RNG tests, called DIEHARD, have been provided by George Marsaglia (Marsaglia 1996).

The underlying statistical tests are of three basic types:

i) a number calculated from the random stream should fit a given distribution, and can be converted into point in a uniform [0,1] distribution (a p-value) by applying the inverse distribution. A large number of such p-values is calculated and a Kolmogorov-Smirnov test (see below) is performed to determine if they are collectively uniform [0,1).

ii) a number calculated from the random stream should fit a given distribution which has a small number of possible values. A large sample of such numbers is calculated and a chi-squared test performed to determine how well the sample fits the expected distribution.

iii) the numbers calculated from the random stream should have an expected distribution, so a probability value is found as a single value, or a likelihood ratio test is performed on a sample.

In some tests the sequence of calculated values is not independent, but has a known covariance matrix. In these case the results are transformed into an independent sample of another distribution before applying one of the above tests.

The Kolmogorov-Smirnov test

The Kolmogorov-Smirnov(K-S) test tests if a set of p-values has been sampled from a [0,1) distribution. The result is a probability value in the range 0 to 1.

A p value result near 1 indicates that this set of p-values is too good to be true. For example the set 0.2, 0.4, 0.6, 0.8 (the numbers are too evenly spread). A p value result near 0 indicates that the numbers are unlikely to be from a uniform [0,1) distribution. For example 0.8, 0.8, 0.81, 0.81 … 0.98, 0.98 (the numbers are clumped together). For further information and an on-line calculation of this test see http://www.physics.csbsju.edu/stats/KS-test.html

Description of the DieHard Test Suite

We now list the 18 tests in the DieHard suite and briefly describe what each

one does.

Tests 1 to 11 consider the output as a stream of bits.

1. BS: Birthday spacings test

Choose m birthdays in a year of n days. If j is the number of values occurring more than once in the list of birthdays, then j has an asymptotic Poisson distribution with mean $m^{3/(4n)}$. This test uses $n = 2^{24}$ and $m = 2^9$ so the mean should be $2^{27}/2^{26} = 2$. A sample of 500 j's is taken and a chi-squared test used to find a p-value (using test type ii) from above). This p-value is computed using bits 1-24, 2-25, ... and 9-32, and a K-S test performed on these p-values.

2. OPERM5: Overlapping 5-permutation test

Tests magnitude ordering of numbers. Each set of five consecutive integers can be in any one of 120 = 5! different size orderings. A sequence of 1,000,004 integers are used to get 1,000,000 sets of 5. As these sets overlap, the sample must be transformed by the inverse of the covariance matrix before a likelihood ratio test may be performed (test type iii)).

Binary Rank tests

These tests measure the coverage of an N-dimensional cube using the columns of a matrix as axes. If the rank is the same size as the matrix, eg 31 for the first test, then you can get to any point in the cube using these axes. The likely values for the ranks are precalculated and the actual values compared to these using the chi-squared test (test type ii)).

3. Binary Rank test (31x31)

The leftmost 31 bits (all but the last bit) of 31 integers are used to form a 31x31 binary matrix. The rank of this matrix is between 0 and 31 (but ranks less than 28 are rare and so combined). A sample of 40,000 is taken and a chi-squared test performed on the ranks (test type ii)). This test thus uses 40,000*31 = 1,240,000 integers, and ignores the lowest bit.

4. Binary Rank test (32x32)

As in the previous test, 40,000 ranks are calculated and a chi-squared test performed (test type ii)). This uses 1,280,000 integers.

5. Binary Rank test (6x8)

8 bits are selected from 6 random integers to form a 6x8 binary matrix. The ranks of 100,000 such matrices are calculated and a chi-squared test performed (test type ii)). This uses 100,000*8 = 800,000 integers.

Sparse Occupancy Tests

The next few tests are all "sparse occupancy" tests on the bitstream. It relies on pre-calculating the probability that a given number of spaces will be empty.

6. Bitstream test (20bit words)

The file is viewed as a stream of bits. Each (overlapping, offsetting one at a time) sequence of 20 consecutive bits is taken to be a 20-bit word. A total of 2^{21} overlapping words are taken and the number j of missing 20-bit words should be (very close to) normally distributed with mean 141,909 and sigma 428. This is converted to a uniform [0,1) p-value and the test is repeated 20 times before a K-S test is applied to the p-values (test type i)).

7. OPSO Overlapping pairs sparse occupancy test

A selected 10 bits from each integer are used to get a stream of letters from an alphabet of 1024 letters. From a stream of $2^{21}+1$ letters, 2^{21} overlapping 2 letter words are considered and the number of missing words counted. This number should be very close to normally distributed, with mean 141,909 and sigma 290, and a p-value is calculated (test type i)). This test is performed using bits 1-10, 2-11,... 23-32 of the first $2^{21}+1$ integers in the test file.

8. OQSO Overlapping Quadruples-Sparse occupancy test

Overlapping four letter words from an alphabet of 32 letters (5 bits) are taken. The number of missing words is almost normal with mean 141,909 and sigma 295, so a p-value is calculated (test type i)). Note that the sigma (standard deviation) is experimentally determined. This test is performed on bits 1-5, 2-6, ... 28-32 of the first $2^{21}+3$ integers.

9. DNA

Overlapping 10-letter words from an alphabet of 4 letters (2 bits) are taken. The number of missing words is almost normal with mean 141,909 and sigma 290, so a p-value is calculated (test type i)). Note that the sigma (standard deviation) is experimentally determined, correct to three places. This test is

performed on bit 1-2, 2-3, … , 31-32 of the first $2^{21}+9$ integers.

Occurrences of 1's and 0's

The next two tests focus on the occurrence of 1's and 0's in each byte.

10. Count the 1's test (in each byte)

The number of binary 1's in each byte of the file is counted to give a sequence of letters from an alphabet of size 5 (A=0,1,2 B=3 .. D=5 E=6,7,8). Overlapping words of length 5 and of length 4 are taken and the frequencies recorded. After transforming by the weak inverses of the covariance matrices a chi-squared test may be performed (test type ii) after transformation).

11. Count the 1's test (in each byte) each byte separately

The number of binary 1's in a specified 8 bits are taken from each integer (bits 1-8, 2-9, … , 25-32) of the file is counted to give a sequence of letters from an alphabet of size 5 (A=0,1,2 B=3 .. D=5 E=6,7,8). Overlapping words of length 5 and of length 4 are taken and the frequencies recorded. After transforming by the weak inverses of the covariance matrices a chi-squared test may be performed (test type ii) after transformation).

For the remaining tests, the leading (most significant) bits are most important, and the trailing bits less important.

Dimensional Correlations

The next three tests look for 2 and 3 dimensional correlations.

12. Parking Lot test

Cars of radius 1 are randomly parked in a 100x100 square. If there is a crash when attempting to park the car then we try again until it fits. Experimentation indicates that after 12,000 attempts at parking, the number of cars parked is normally distributed with mean 3523 and sigma 21.9. This gives a p-value (test type i)), and the test is performed 10 times before a K-S test is performed on the p-values.

13. Minimum Distance test

Choose 8000 random points in a square of side 10000. If d is the minimum

distance between points then d^2 should be exponentially distributed with mean .995. Thus 1- exp(-d^2/0.995) should be uniform [0,1) (test type i)). After calculating 100 such p-values, perform a K-S test.

14. 3d Spheres

Choose 4000 random points in a cube of side 10000. Calculate the minimum distance between points and cube it. This number should be exponential with mean 30 (experimentally determined), giving rise to a p-value (test type i)). Find 20 such p-values and perform a K-S test.

Miscellaneous tests

15. Squeeze test

The integers are floated to get uniform [0,1) numbers. Starting with k = 2^{31} count the number of times k = ceiling(k*U) must be performed before k is reduced to 1, where U are the random floating point numbers in [0,1). The expected distribution of counts is known, so after finding 100,000 such counts a chi-squared test is performed (test type ii)).

16. Overlapping sums test

Float the integers to get a sequence U(1), U(2), … of uniform [0,1) variables. The overlapping sums S(1) = U(1) + … + U(100), S(2) = U(2) + … +U(101), … are almost normal with a known covariance matrix. A linear transformation of the S's gives a sequence of independent standard normals which are then converted to uniform variables for a K-S test (test type i)). The p-values from 10 such K-S tests are together given a K-S test (test type i)). This test looks for short "cycles" in the high order bits. Cycles will give rise to approximately evenly spaced p-values which will fail the K-S tests. The combination of the two tests just extends the size of the cycle that can be detected.

17. Runs test

Counts the number of runs up and runs down for sequences of 10,000 [0,1) variables. The covariance matrix for runs up and runs down is well known, so a chi-squared test may be performed to obtain a p-value (test type ii)). This is done 10 times and a K-S test performed on the p-values. The whole test is then repeated. This uses 200,000 integers.

18. Craps test

Each integer is divided and multiplied to give a dice roll: 1,2,3,4,5 or 6. With these dice rolls 200,000 games of Craps are played and the number of wins as well as the number of throws for each game is counted. The p-value for the number of wins is calculated, and a chi-squared test is performed on the number of throws per game (test type ii)).

Random number generators in GAUSS V3.6

Gauss V3.6 provides two random number generators:- rndi and rndkmi.

RNDI

rndi is a linear congruent type random number generator. That is

$$new\_seed = (a * seed + c) \% m$$

m is 2^{32} (it was 2^{31}-1 in the DOS version).

This type of random number generator has three know failings:-

i) the maximum period is 2^{32}-1 which is too short for many of today's uses.

ii) the lower order bits produces by this type of generator are not random. However in previous versions of Gauss only floating point numbers in the range [0,1) were generated so this failing was not so serious.

iii) when used as a source of points in an N-dimensional cube, the points tend to cluster on plains as the number of dimensions is increased (see Marsaglia 1968 and 1972 and Knuth 1980 p 91).

RNDKMI

rndkmi is an implementation of the "Monster" random number generator proposed by Marsaglia (Marsaglia 2000). It is a "recur-with-carry" type generator with an extremely long period.

Recur-with-carry generators (which include subtract-with-carry as a special case) create each new value of the sequence as a linear combination of the previous r values:

$$x_n = a_1 x_{n-1} + a_2 x_{n-2} + \ldots + a_r x_{n-r} + carry \mod b$$

where the carry is the number of multiples of b dropped when reducing the previous linear combination to lie in the range 0 to b-1.

The period of these generators is the order of b (except for the two trivial seed

sets 0,...,0 and a_1 + a_2 + ... + a_r - 1, b-1 ,b-1,... b-1).

The dimension of such a generator is r. If we break our sequence into blocks of r integers and use each block to plot a point in an r-dimensional cube then the points will random.

Marsaglia has devised a simple, fast, "recur-with-carry" type generator which has a period of about 10^{8859} and dimension of 920.

Experience with generators of this type with shorter period has shown poor performance on some of the DieHard tests. However the huge period of this generator makes such (suspected) failings invisible on 'small' samples of only millions or billions of integers.

The initialisation of this generator requires 920 random numbers. Marsaglia recommends initialising it using the KISS (Keep It Simple Stupid) random number generator, which combines three simple generators and seems to pass all current tests. The KISS generator has a period of about 10^{41}.

The KISS generator itself requires 6 initial values. In order to simplify the use of rndkmi all but one of these values was set to those values suggested by Marsaglia (Marsaglia 2000). The only initial seed is the input to the 3-shift shift-register generator of the KISS generator. For any non-zero seed this 3-shift shift-register generator as a period of 2^{32}-1

Finally, the output of Marsaglia's "Monster" recur-with-carry generator can be further improved adding the outputs of the KISS and the Monster which gives a period of greater than 10^{8888} and ensures a sequence of random numbers at least as good as the Monster or KISS taken separately. This combination also extends the dimension of the generator to greater than 920.

Results of Tests for GAUSS V3.6

The two random number generators provided in Gauss V3.6 were tested. The following table summarises the results of each test where P is Pass and F is fail.

	Rndi	RndKMi
Birthday Spacing	F* (bits 8-31 and 9-32 gave p-value 1.00000)	P
Overlapping 5's	F* (one test gave p-value 0.999993)	P
Binary rank 31x31	P	P
Binary rank 32x32	P	P
Binary rank 6x8	F* (bits 17 to 32 give p-values of 1.000000)	P
Bitstream	F	P* (one test out of 20 was 0.003)
OPSO	F (bits 3 to 32)	P
OQSO	F (bits 6 to 32)	P
DNA	F (bits 9 to 32)	P
Count the 1's (steam of bytes)	F	P
Count the 1's (selective bytes	F (bits 16 to 32)	P
Parking Lot	P	P* (one test out of 10 was 0.005)
Minimum distance	P	P
3d spheres	P	P
Squeeze	P	P
Overlapping sums	P	P
Runs	P	P
Craps	P	P

As explained by Marsaglia (Marsaglia1996) occasional p-values near 0 and 1 are to be expected, here Pass means the p-value was between 0.01 and 0.99. When a test really fails p-values of 0 or 1 to six digits are recorded.

Discussion of the Results

In both tests the integer output of the random number generator was used. This integer output function is new to Gauss V3.6 and gives the user access to the raw bits generated by the random number generator. In previous versions of Gauss only the floating point numbers [0,1) were available.

These tests indicate that RNDI produces acceptable random real numbers for small sample sizes in [0,1) but is not suitable as a source of random bits due to the non-randomness of the low order bits. The maximum period of RNDI is 2^{32}-1 which is too small for many of today's uses. Another problem with RNDI is that when it is used as a source of points in an N-dimensional cube, the points tend to cluster on plains as the dimensions is increased (see Marsaglia 1968 and 1972 and Knuth 1980 p 91).

On the other hand the RNDKMI passes all the DieHard tests and appears to be suitable for all purposes. It has a period of greater then 10^{8888} and dimension of greater than 920. One problem of a generator like this, with such a long period, is that it is possible to get results much closer to 1 and 0 in the normal course of use. So occasionally the DieHard results for this generator give p-values close to 1 or 0. In these cases, generate further sets of random integers for DieHard to process and then apply the K-S test to the set of results.

It is recommended that rndkmi be used whenever random numbers are required.

Appendix A - Test Code

The following Gauss program was used to output the test file needed by DieHard.

```
/* Generate random numbers and convert to text hex */

intelPC = 1;
/* if set to non-zero ascii output arranged so that asc2bin
**   produces bin in lsb first as required by DieHard on Intel PCs
**   if set zero ascii is in lsb
*/
/* need 80Mbits */
inc = 4096*2;
m = 0;
newstate = 1; /* starting state for Kiss-Monster */
rndseed 1;  /* starting state for original random number generator */
```



```
print "Writing up to 40 ";
screen on;
output file=km.txt reset;
do while m<40;  /* 10 numbers per line so 400 in this loop */
    /* choose one of the following lines for
    ** either the original random number generator rndi
    ** or the new Kiss-Monster random number generator
    */
    /* {y,newstate}=rndKMi( inc, 10, newstate); */  /* Kiss-Monster */
    y = rndi(inc, 10);       /* original random number generator rndi */
    m = m+1;
    output off;
    format /rd 2,0;
    print m;;
    hex = {"0","1","2","3","4","5","6","7","8","9","A","B","C","D","E","F"};
    format /sa 0,0;
    screen off;
    output on;
    for k (1,rows(y),1);
        for j (1,10,1);
            a = y[k,j];
            hexStr = "";
            hexByte = "";
            for i (1,4,1);
             r = a%16;
             a = (a-r)/16;
             hexByte = "" $+hex[r+1];
             r = a%16;
             a = (a-r)/16;
             hexByte = "" $+hex[r+1] $+ hexByte;
             if intelPC;/* ascii is msb first */
                 hexStr = hexByte $+ hexStr;
             else;
                 hexStr = hexStr $+ hexByte;
             endif;
            endfor;
            print $hexStr;;
        endfor;
        print;
    endfor;
    output off;
    screen on;
endo;
output off;
screen on;
print;
```

The DieHard results were produced by

i) modifying the above Gauss program to select either the new Kiss-Monster random number generator (rndKMi) or the original linear congruent random number generator (rndi) and then running the program using Gauss V3.6.13 Windows version. This produces a text output file km.txt.

ii) The ASC2BIN.EXE program supplied with DieHard was then used to produce a binary file (km.out) from the text file km.txt

iii) Diequick.exe was then used to analyse the binary file, km.out, and produce a report on the randomness of the generator.

References

Marsaglia, G. (1968), "Random Numbers Fall Mainly in the Planes", Proceedings of the National Academy of Sciences of the United States of America, 60, pp 25-28.

Marsaglia, G. (1972), "The structure of linear congruential sequences", Applications of Number Theory to Numerical Analysis, Academic Press.

Marsaglia, G. (1984) "A current view of Random Number Generators," Proceedings of Computer Science and Statistics: 16th Symposium on the Interface, Atlanta, 1984.

Marsaglia, G. (1993) "Monkey Tests for Random Number Generators," Computers and Mathematics with Applications, 26, 1-10.

Marsaglia, G. (1996) "DIEHARD: A battery of Tests of Randomness," http://stat.fsu.edu/~geo.

Marsaglia, G. (2000) "The Monster, a random number generator with period over 10^2857 times as long as the previously touted longest-period one", preprint.

Ripley, B. D. (1990), "Thoughts on Pseudorandom Number generators", Journal of Computational and Applied Mathematics, 31, 153-163.

Vinod, H. D. (2000) "Review of Gauss for Windows, including its numerical accuracy", J. Appl. Econ. 15: 211-220 (2000).

Knuth, D.E. (1980) "The Art of Computer Programming" 2nd ed. Addison-Wesley.

