Nothing is random, only uncertain. *Gail Gasram*

THE MARSAGLIA RANDOM NUMBER CDROM

This CDROM contains 4.8 billion random bits, in sixty 10-megabyte files. They were produced by a combination of several of the best deterministic random number generators (RNG's), together with three sources of white noise, as well as black noise. My intent is to provide an unassailable source for those who absolutely positively have to have a large, reliable set of random numbers for serious simulation (Monte Carlo) studies. Details of the production method are below.

This CDROM also contains files for DIEHARD: a battery of tests of randomness. These files are in various versions: DOS, UNIX, and (i486+)LINUX, with exec files as well as C code from which the DIEHARD battery can be compiled.

Also included are files for programs that will create test files from most of the common random RNG's, as well as several newer and demonstrably better ones.

1 Producing the Random Numbers

The sixty 10-megabyte files of random numbers are produced by combining two or more of the most promising deterministic generators with sources of random noise from three physical devices (white noise), for those who feel that physical sources of randomness are better than deterministic sources. Some of the files had white noise combined with black noise, the latter from digital recordings of rap music. And a few of the files even had naked ladies thrown into the mix, from pixel files on the network. The last two, digitized music and pictures, are thrown in to illustrate the principle that a satisfactory stream of random bits remains so after combination with the bits of any file .

That important principle is this: if

$$x_1, x_2, x_3, \ldots,$$

is a truly random sequence of independent bits taking values 0 and 1 with equal probabilities, and if

$$y_1, y_2, y_3, \ldots$$

is any sequence of *constant* or *random* bits, then the sequence

$$x_1 + y_1 \bmod 2,\; x_2 + y_2 \bmod 2,\; x_3 + y_3 \bmod 2, \ldots$$

is also a truly random sequence of independent bits.

Thus if we are confident that a physical device, carefully constructed to produce randomness (say by means of Johnson noise, as with one of our inputs), really does produces random bits, then combining it with the output of a deterministic generator(s) will maintain that perfect randomness, and still maintain it if we throw in some black noise.

It turns out, however, that no physical devices I have considered pass the stringent randomness requirements of my DIEHARD battery of tests. But the deterministic methods do. So, in effect, I am assuming our deterministic random bits are the truly random ones, the x's. And the perturbing ones, the y's, are coming from physical devices, used in combination with the x's to make prediction virtually impossible.

One can define a Truly Random Sequence Of Bits (TRSOB) recursively:

$$x_1, x_2, x_3, \ldots \textit{is a } \mathrm{TRSOB} \quad \textit{if and only if}$$

$$x_1 + y_1 \bmod 2, x_2 + y_2 \bmod 2, x_3 + y_3 \bmod 2, \ldots \textit{is a } \mathrm{TRSOB}$$

for every sequence of bits $y_1, y_2, y_3, \ldots$, *constant or random.*

This is certainly not a constructive definition—but how can the undefinable be defined? Nothing is truly random. At best, we can take advantage of this idea by hoping that one of the components of our combination, deterministic(s) + noise(s), produces something adequately close to a TRSOB.

As DIEHARD shows, the bits on this CDROM seem to be truly random SOB's. But they are not, nor can they or any others ever be. At best I can only quantify the uncertainty by betting, with heavy odds, that the bits on this CDROM will produce results consistent with probability theory—that mathematical fiction we use to quantify degrees of uncertainty.

Their potential uses range from elaborate scientific simulations to use in games: shoot craps, with the number of throws and number of winning games almost surely within the limits provided by probability theory–and just as surely, occasional results outside those limits, with predicted rarity. They could run the games in Las Vegas for a thousand years, with no one the wiser. (Except those who have this CD. Actually, the most prolific consumers of random numbers are slot machines, and many of them have internal generators based on deterministic methods I have developed. Whether those generators are perturbed with some physical input, I have not been told.)

The CDROM can provide the random keystrokes of a monkey at a typewriter and produce the expected number of particular four-letter words, or total number of different words, etc.—all within the the predicted limits most of the time, and outside with the rarity predicted by theory.

And, most importantly, they can provide the source of randomness necessary for serious Monte Carlo studies, ranging from random knotting of protein molecules to the formation of galaxies. In such studies, any discovery raises the question: Is the result a property of the physical system we are simulating, or is it just because of bad random numbers?

With the many deterministic RNG's that have been developed, one can usually answer that question. If the result holds for a variety of different kinds of RNG's, then the result can be confidently said to come from the assumptions about the physical system and not from departure from the assumed uniformity and independence of the random input.

This disk provides a variety of some of the best deterministic methods for such comparisons, as well as the purported TRSOB's that should be the first recourse.

2 Physical Sources of Randomness

I have an elaborate brochure from a company that uses

> "a proprietary technology to create truly random numbers. Since the [device] is based on a naturally occurring random phenomenon (Johnson Noise) rather than a digital logic circuit or computer program, it requires no initial starting value and each new value is independent of all previous values."

Sounds great. Such a device should solve the problem of providing TRSOB's.

In planning for this CDROM, I bought one of those devices, (about $400 US) from Canada. I also bought an equally impressively documented device, promising true randomness, from Germany (about $300) and had access to the 'random output' from a device from California.

So I had purported TRSOB's from Canada, Germany and California. Were they Truly Random SOB's? No, not at all. All three failed spectacularly on the monkey tests, which emphasize testing for independence.

Examples of the output from these devices are in the files

canada.bit   germany.bit   californ.bit

for those who want to try them with the DIEHARD battery.

I have used output from these devices in forming the random bit files in this CDROM, but in combination with some of the most promising deterministic RNG's, described next. The latter pass all tests in DIEHARD, and I view them as the more important part of the combination—the x's of the above discussion. The physical device bits, the y's, are there to prevent predictability. They also make the periods infinite, but that is of no consequence, as the period of the deterministic component is so huge as to be infinite for practical purposes.

3 The Deterministic Generators

As you will learn from the `makewhat` executable files on this disk, there are many deterministic RNG's that pass all the tests in DIEHARD and have extremely long periods. Furthermore, they seem to pass for all substrings of bits, trailing as well as leading, from their 32-bit output. Some of them are:

32-bit multiply-with-carry (MWC) generators, $x_n = ax_{n-1} + \text{carry mod } b = 2^{32}$. Form $ax + c$ in adjoining registers (64 bits). The new c is the top 32 bits, the new x the bottom 32. If a is chosen so that $m = ab - 1$ is a safeprime, the period will be $(m-1)/2$, and the generator seems to pass all tests. See the postscript file mwc1.ps.

The Mother of all RNG's,

$x_n = 2111111111x_{n-4} + 1492x_{n-3} + 1776x_{n-2} + 5115x_{n-1} + \text{carry mod } b = 2^{32}$
To implement, form the linear combination in adjoining registers (64 bits). The new c is the top 32 bits, the new x the bottom 32. The period is about 2^{158}. See the file mwc1.ps.

Concatenated 16-bit MWC generators

While 32-bit multiply-with-carry generators are more desirable, they can only be fully exploited if one has means to form the 64-bit product of 32-bit integers, which usually requires assembler. But versions for base $b = 2^{16}$ are readily implemented in C or Fortran: Choose the multiplier a so that $m = ab - 1$ is a safeprime. For 16-bit seed values x and $c(< a)$, form $ax + c$ in 32 bits. The new c is the top 16, the new x the bottom 16. For example, in C, `w=a*(w&65535)+(w>>16)` forms a new 32-bit word `w` whose upper,lower parts are the new c, x. The period is $ab/2-1$. Two such 16-bit sequences can be concatenated to form a 32-bit sequence that seems to pass all tests. The period is around 2^{120}. See the `makewhat` exec file.

The KISS Generator

The Keep It Simple Stupid generator combines simple generators to get a fast, easily programmed composite with very long period. It forms $x + y + z \bmod 2^{32}$, where x, y, z are formed by

$$\begin{aligned} x_n &= 69069x_{n-1} + 1 \bmod 2^{32} \\ y_n &= y_{n-1}(I + L^{13})(I + R^{17})(I + L^5) \\ z_n &= 2z_{n-1} + z_{n-2} + \text{carry mod } 2^{32} \end{aligned}$$

The y's are a (3)shift-register sequence. The period of KISS is about 2^{127}. See the makewhat exec file as well as make.txt.

A Simple Combination Generator

The two sequences are

$x(n) = x(n-1) * x(n-2) \bmod 2^{32}$
$y(n) = 30903 * y(n-1) + \text{carry mod } 2^{16}$,

The first is a lagged Fibonacci generator using multiplication on 32-bit odd integers, the second a multiply-with-carry generator for 16-bit integers. With addition of the two, modulo 2^{32}, the resulting generator, called COMBO, has period $> 2^{60}$ and seems to pass all tests. The x's themselves pass almost all tests except those dependent on the rightmost bit, which will always be 1. The y's are an easy way to make the rightmost 16 bits as good as the left. COMBO is easily implemented in Fortran or C. In C, the y's are the right half of an integer `w` generated recursively by `w=30903*(w&65535)+(w>>16);`

Extended Congruential Generators

Ordinary congruential generators modulo a prime have too short a period and are a nuisance to implement. They do behave very well on all tests. As long as one must go to double precision to implement arithmetic modulo a prime, one might as well get the much longer period by going to an extended congruential generator. For them, the recursion is a linear combination modulo a prime p. With proper choice of the k coefficients, the period is $p^k - 1$. Examples from the `makewhat` program:

$$x_n = 1176x_{n-1} + 1476x_{n-2} + 1776x_{n-3} \bmod 2^{32}-5$$
$$x_n = 2^{13}[x_{n-1} + x_{n-2} + x_{n-3}] \bmod 2^{32}-5$$
$$x_n = 1995x_{n-1} + 1998x_{n-2} + 2001x_{n-3} \bmod 2^{35}-849$$
$$x_n = 2^{19}[x_{n-1} + x_{n-2} + x_{n-3}] \bmod 2^{35}-1629$$

All pass the tests in DIEHARD, but the second and fourth can be implemented (as double precision reals) to avoid multiplication. The second can also be implemented in 32-bit integer arithmetic. A DOS assembler version takes 170 nanoseconds on a Pentium 120, a C version, 400. It is one of the fastest pass-all-tests generators with so long a period, $\approx 2^{96}$. I had misgivings when developing it, remembering the use of multiplier $2^{16} - 3$ in RANDU, chosen to avoid multiplication, with poor results. But integers from the recursion $x_n = 2^{13}[x_{n-1} + x_{n-2} + x_{n-3}] \bmod 2^{32}-5$ seem to pass all tests—so far. You may want to develop a fast assembler version yourself and test it.

4 How the Random Bit Files Were Formed

The random bit files are `bit.01, bit.02, ... , bit.60`, each 10 megabytes. They were formed by combining the 32-bit output of at least two of the above deterministic generator with 32-bit integers formed by streams of bits from one or more of the physical devices from Canada, Germany or California. Most of the files had the deterministic component made from the sum of the Mother-of-All and the KISS generators. Some of the files had yet another 32-bit integer combined, from a rap music CD, or, a few times, from capture of the screen of a TV program or the bit map of a naked lady. The combinations were by exclusive-or of the 32-bit words. This means bits in corresponding positions are added modulo 2, as described above with the x and y sequences of bits.

Confidence in final random bit files stems from the suitability of the deterministic generators themselves. If they can properly serve by themselves as sources of randomness, then perturbing them with the output of physical devices, rap music or TV screen captures, can only add to their unpredictability, not detract from their suitability as a source of randomness.

The resulting integers were written to binary files, so the `bit.??` files can be viewed as a sequence of bits, or bytes or 16- or 32-bit integers, depending on input required for an application.

5 Summary of Files on This CDROM

In the base directory, 10 megabyte files `bit.01, bit.02, ... , bit.60` as well as files of about 10 meg from the output of three physical devices: `canada.bit, germany.bit, california.bit`.

There is a directory `pscript` containing this file, `cdrom.ps`, as well as files describing random number generators (`keynote.ps`), tests (`monkey.ps`) or the new multiply-with-carry RNG's (`mwc1.ps`).

There are three directories, `dos, linux and unix`, containing files for running DIEHARD on those particular systems.