

COMPUTING RANDOM NUMBERS

Jonathan Rabone
Four Delta Consulting Ltd.

Why Random Numbers?

- Simulation of physical “noise” – e.g. Brownian motion, electrical noise, statistical error – by random walk techniques
- Monte Carlo simulation – exploring a problem using probabilistic methods and random stimuli
- Random perturbation optimisation techniques such as simulated annealing

Randomness in Simulated Biofilms

- Random walk simulation of rate-controlled processes
 - motion and diffusion of nutrient & cells through substrate medium
- Virtual cell membrane modelling
 - diffusion of biocidal substances into living cells

Random Numbers

- “True” randomness is hard to find
 - Noise of electrical components (e.g. resistor, reverse-biased PN junction)
 - Observation of random process (e.g. lava lamps, radioactive decay)
- Entropy gathering daemons
 - mouse motion
 - key-stroke delay
 - hard disk latency

Pseudo Random Numbers (1)

- “Real” randomness only justified in some cases (cryptographical applications)
- “Pseudo” random numbers can be good enough:
 - provided statistical properties of randomness are met!
 - experiments can be repeated exactly by using same seed

Pseudo Random Numbers (2)

- “Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin.” – John von Neumann

How Random is Random?

- Chi-square testing
- Monte Carlo methods for constants (e.g. π)
- George Marsaglia's DIEHARD suite (FORTRAN)
- But exercise caution!

Face-off: Sun vs Mersenne Twister

- The standard Java RNG is implemented in `java.util.Random`. It is a simple linear congruential generator with some tweaks.
- Mersenne Twister is a twisted generalized feedback shift register generator, developed by Makoto Matsumoto and Takuji Nishimura^[1].

Demonstration



The screenshot shows a Java applet window titled "RNG Comparison Applet". The main title is "Comparison of two RNGs". On the left, there are "Start" and "Stop" buttons. The window contains two side-by-side plots of random numbers. The left plot is labeled "java.util.Random" and the right plot is labeled "MersenneTwisterFast". Below the plots, there are two input fields: "Generation" with the value "6" and "Frames / second" with the value "14.3". The Four Delta Consulting Limited logo is visible in the top right corner of the applet window.

What Just Happened? (1)

- Sun's LCG algorithm is straight from Knuth (probably via the `rand48()` C library function).
- It uses a 48 bit seed but only 17 bits contribute to the LSB, producing severe non-randomness & periodicity in the low-order bits, despite extra code to compensate for this.
- This is a known problem with LC generators.

What Just Happened? (2)

- Mersenne Twister is designed for Monte Carlo simulations with a huge, proven periodicity of $2^{19937} - 1$ results.
- The “twist” is a transformation which assures equi-distribution of the generated numbers in 623 dimensions. This indicates negligible serial correlation between successive results.
- MT is also faster than the default Java generator!

DIEHARD Testing

- Java and Mersenne generators tested using DIEHARD compiled for Win32^[2].
- 25 seconds runtime (40MB of random 32-bit numbers) on a 3.2GHz Pentium 4 for all 15 tests.
- Mersenne Twister passes all tests, but Java fails some...

Java PRNG DIEHARD Results

OQSO for java.32	using bits 8 to 12	141733	-.598	.2750
OQSO for java.32	using bits 7 to 11	141576	-1.130	.1293
OQSO for java.32	using bits 6 to 10	141989	.270	.6064
OQSO for java.32	using bits 5 to 9	216206251.853	1.0000	
OQSO for java.32	using bits 4 to 8	209216228.158	1.0000	
OQSO for java.32	using bits 3 to 7	131865-34.049	.0000	
OQSO for java.32	using bits 2 to 6	146281 14.819	1.0000	
OQSO for java.32	using bits 1 to 5	126059-53.730	.0000	
DNA for java.32	using bits 31 to 32	142748	2.474	.9933
DNA for java.32	using bits 30 to 31	142818	2.680	.9963
DNA for java.32	using bits 29 to 30	141945	.105	.5419

Conclusions

- Don't trust the platform! Choose a suitable RNG algorithm based on quality, size and speed trade-offs.
- Look for theoretical analysis & tests (e.g. spectral & weighted spectral tests) of the chosen implementation.
- Verify implementation against standard empirical test suites.

Alternatives

- Hardware RNG (slow & expensive!)
- Media containing random digits (CD-ROM etc.)
- Internet and web-services^[3] which provide random numbers on demand.

Links

1. <http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html>
2. <http://www.stat.fsu.edu/pub/diehard/>
3. <http://www.random.org/>