

Random Number Generation (RNG)

read “Numerical Recipes” on random numbers and the chi-squared test.

Today we discuss how to generate and test random numbers.

What is a random number?

- A single number is not random. Only an infinite sequence can be described as random.
- Random means the absence of order. (Negative property).
- Can an intelligent gambler make money by betting on the next numbers that will turn up?
- All subsequences are equally distributed. This is the property that MC uses to do integrals and that you will test for in homework.

*“Anyone who considers arithmetical methods of random digits is, of course, in a state of sin.”
John von Neumann (1951)*

“Random numbers should not be chosen at random.” Knuth (1986)

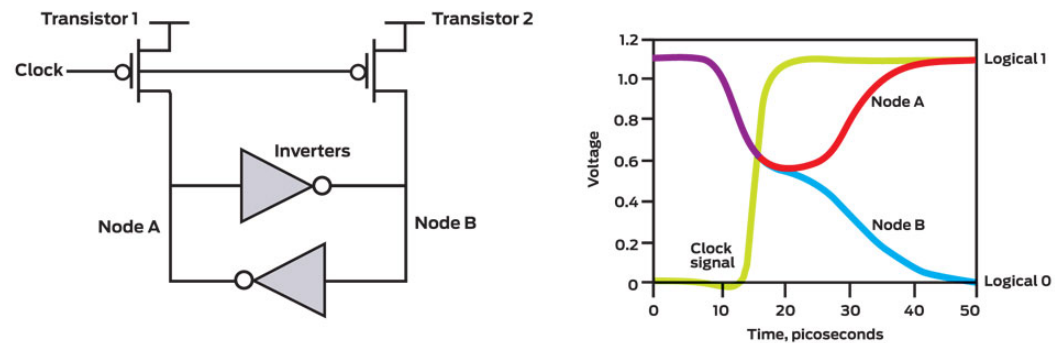
Random numbers on a computer

- **Truly Random** - the result of a physical process such as timing clocks, circuit noise, Geiger counts, bad memory
 - Too slow (we need 10^{10} /sec) and expensive
 - Low quality
 - Not reproducible. But new INTEL chip does this for security
- **Pseudo-random**. *prng* (pseudo means *fake*)
 - Deterministic sequence with a repeat period but with the appearance of randomness (if you don't know the algorithm).
- **Quasi-random** (quasi means *almost random*)
 - “half way” between random and a uniform grid.
 - Meant to fill space with max. distance between space to fill “holes” sequentially (e.g., $[0,100]$ as $0,1,2,3,\dots,100$).

If doing a low-D integral, QRNG builds in correlation so the sequence will be more uniform than PRNG and converge to mean value quicker than $N^{-1/2}$.

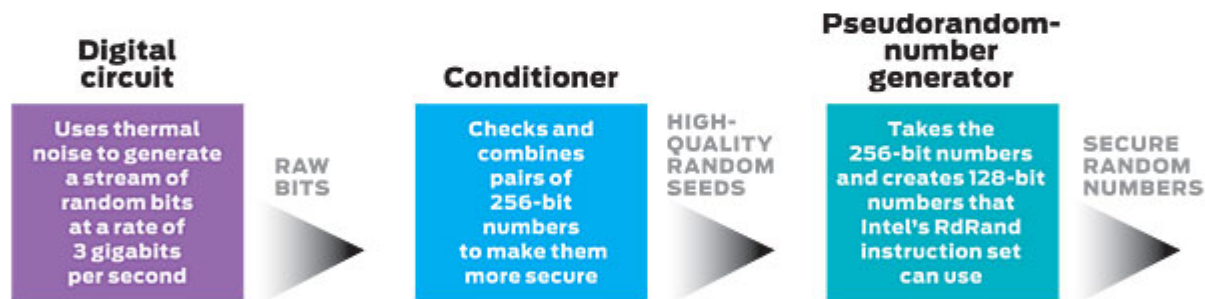
Intel's RNG: Ivy Bridge

- Article describing it: 3 Gbits/sec
- <http://spectrum.ieee.org/computing/hardware/behind-intels-new-random-number-generator/0>



UNCERTAIN CIRCUITS: When transistor 1 and transistor 2 are switched on, a coupled pair of inverters force Node A and Node B into the same state [left]. When the clock pulse rises [yellow, right], these transistors are turned off. Initially the output of both inverters falls into an indeterminate state, but random thermal noise within the inverters soon jostles one node into the logical 1 state and the other goes to logical 0. © 2011 IEEE Spectrum magazine

CLOSE X



THREE-LAYER NUMBERS: Intel's Bull Mountain random-number generator prevents bias and correlation with a three-step process. First, a digital circuit generates a stream of raw random bits. Next, a conditioner generates healthy random seeds. Third, a pseudorandom-number generator readies the numbers for use in software. © 2011 IEEE Spectrum magazine

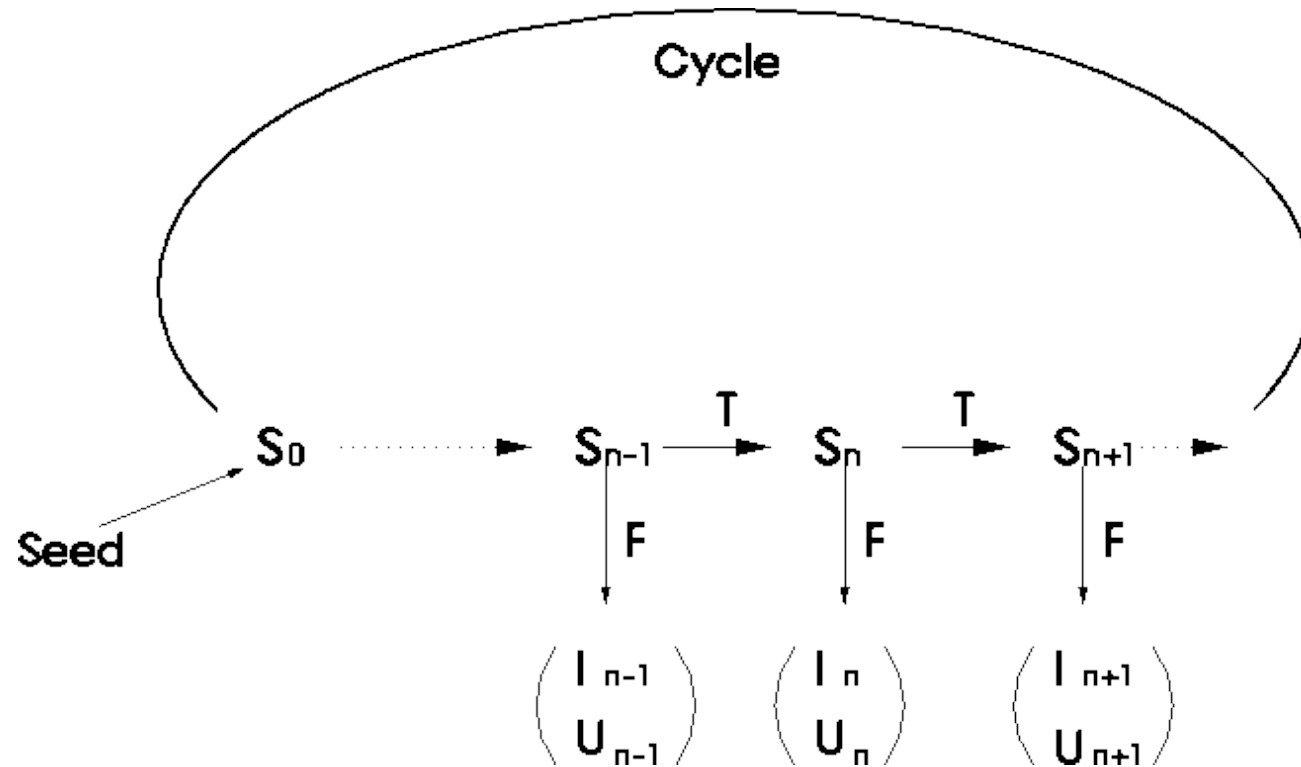
CLOSE X

Desired Properties of RNG on a computer

- **Deterministic:** easy to debug (repeatable simulations).
- **Long Periods:** cycle length is long.
- **Uniformity:** RN generated in space evenly.
 - Go through complete cycle, all integers occur once!
 - TEST: N occurrences of x . The no. in (a,b) is equal to $N(b-a) \pm [N(b-a)]^{-1/2}$
- **Uncorrelated Sequences:**
$$\langle f_1(x_{i+1}) \dots f_k(x_{i+k}) \rangle = \langle f_1(x_{i+1}) \rangle \dots \langle f_k(x_{i+k}) \rangle$$

Monte Carlo can be very sensitive to such correlations, especially for large k .
- **Efficiency:** Needs to be fast.

Pseudo Random Sequence



S: State and initial seed.

T: Iteration process,

F: Mapping from state to integer RN (I) or real RN (U).

Period or Cycle Length (L)

- If internal state has M bits, total state space is 2^M values.
- If mapping is 1-1, space divides into a finite no. of cycles.
- Best case is a single cycle of length 2^M , otherwise $2^M/L$.
- It is easy to achieve very long cycle length: 32 or 24 bit generators are not long enough!
- Entire period of the RNG is exhausted in:

- | |
|--|
| <ul style="list-style-type: none">• rand (1 processor) \sim 100 seconds• drand48 (1 processor) \sim 1 year• drand48 (100 processor) \sim 3 days• SPRNG LFG (10^5 procs) \sim 10^{375} years |
|--|
- Assuming 10^7 RN/sec per processor

e.g., In 1997, computer has $\sim 10^8$ RN/proc/sec (or 2^{26}).

Hence, it takes 1 sec to exhaust generator with 26 bits and 1 year for 2^{51} states.

e.g., consider 1024×1024 sites (spins) on a lattice. With MC, sweep 10^7 times to get averages.

You need to generate 10^{13} RN. A 32-bit RNG from 1980's dangerous on today's computers.

Common PRNG Generators

Linear Congruential Generator (LCG) modulo (mod) is remainder after division

64-bit word (or 2 32-bit LCG) give a period of $\sim 10^{18}$.

<ul style="list-style-type: none"> Multiplicative Lagged Fibonacci Modified Lagged Fibonacci 	$z_n = z_{n-k} * z_{n-l}$ $z_n = z_{n-k} + z_{n-l}$ <p>(modulo 2^m)</p>	<p>vary initialization</p>
<ul style="list-style-type: none"> 48 bit LCG 64 bit LCG Prime Modulus LCG 	$z_n = a * z_{n-1} + p$ <p>(modulo m)</p>	<p>vary p</p>
		<p>vary a</p>
<ul style="list-style-type: none"> Combined Multiple Recursive 	$z_n = a_{n-1} * z_{n-1} + \dots + a_{n-k} * z_{n-k} + \text{LCG}$	<p>vary LCG</p>

Recurrence

Parallelization

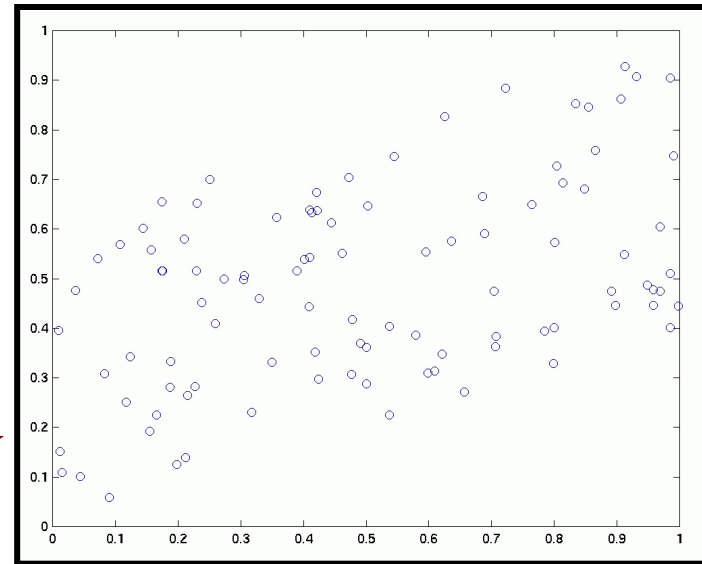
Uniformity

- Output consists of taking **N bits from state and making an integer in $(0, 2^N - 1)$ “ I_k ”**.
- One gets a **uniform real “ U_k ” by multiplying by 2^{-N}** .
- We will study next time how to get other distributions.
- If there is a single cycle, then **integers must be uniform in the range $(0, 2^N - 1)$** .
- Uniformity of numbers taken 1 at a time is usually easy to guarantee. What we need is higher dimensional uniformity!

Sequential RNG Problems

- Correlations → non-uniformity in higher dimensions

Uniform in 1-D but
non-uniform in 2-D



This is the important property
to guarantee:

$$\langle f_1(x_{i+1})f_2(x_{i+2}) \rangle = \langle f_1(x_{i+1}) \rangle \langle f_2(x_{i+2}) \rangle$$

MC uses numbers many at a time--they need to be uniform.

e.g., 128 atom MC moving 3N coords. at random needs 4 PRNG (x,y,z, i) or

128 x 4=1024. So every 1024 moves may be correlated!

Compare error for each PRNG, if similar OK. (Use χ^2 -test.)

LCG Numbers fall in planes.

Good LCG generators have the planes close together.

For a k -bit generator, do not use them more than $k/2$ together.

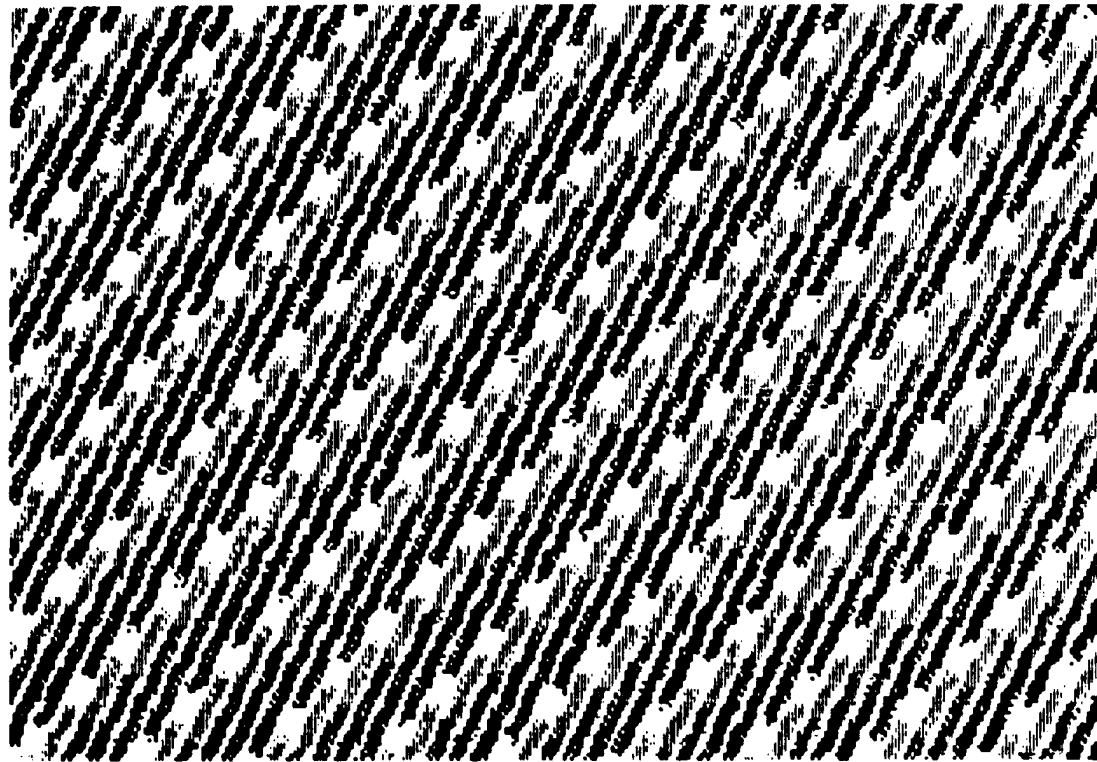


Fig. 8.2 Correlation between the triplets of points \vec{r}_n $\{x_{3n}, x_{3n+1}, x_{3n+2}\}$ generated by a pseudorandom number generator. The value of the third coordinate is represented by the intensity of the point.

What is a chisquare test?

- Suppose $\{x_1, x_2, \dots\}$ are N Gaussian random numbers with mean zero, and variance 1
- What is the distribution of $y = \sum_{k=1}^N x^2$?
- $Q(\chi^2 | N)$ is the probability that $y > \chi^2$
- The mean value of y is N , its variance is $2N$.
- We use it to test hypotheses. Is the fluctuation we see, what we expect?

$$\chi^2 = \sum_{k=1}^N \frac{(x_k - x_k^{fit})^2}{\sigma_k^2}$$

- Roughly speaking we should have $N - \sqrt{2N} < \chi^2 < N + \sqrt{2N}$
- More precisely look up value of $Q(\chi^2 | N)$ “chisquare function with N degrees of freedom”

Chi-squared test of randomness

- **HW exercise:** do test of several generators
 - Divide up “cube” into “N” bins.
 - Sample many “P” triplets.
 - Number/bin should be $n = P/N \pm (P/N)^{1/2}$

In example: expected $n = 10 \pm (10)^{1/2}$

N=100

P=1000

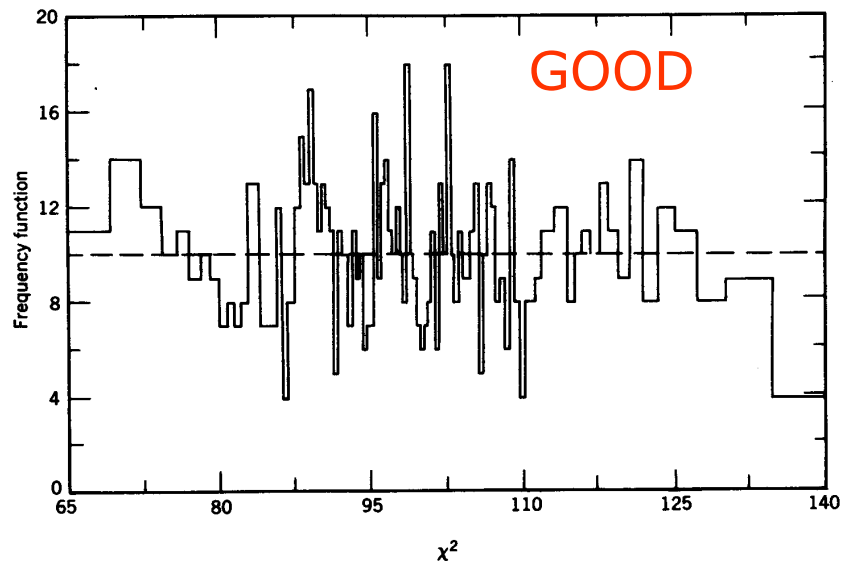


Figure A.1. The frequency function of observed values of χ^2 after 1000 samples. The bins are equally probable intervals where the expected number of χ^2 in each bin is 10. The pseudorandom number generator was Eq. (A.13).

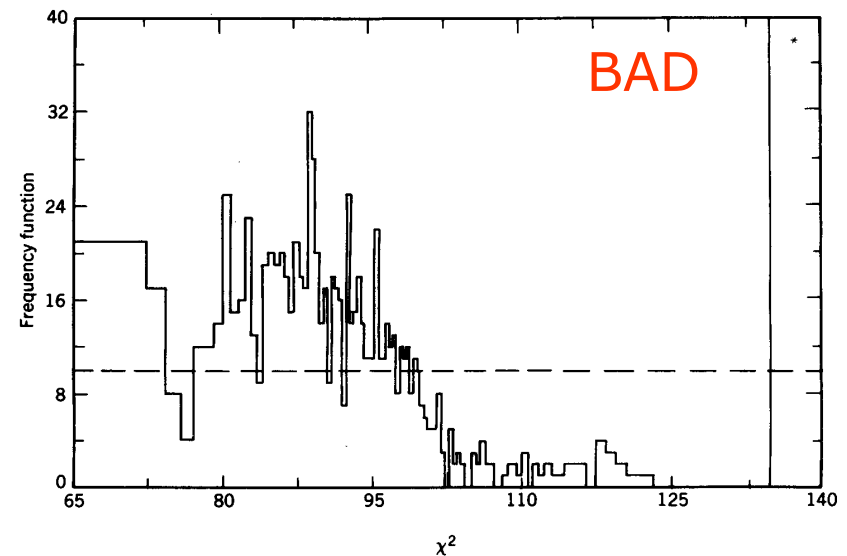


Figure A.2. The frequency function of observed values of χ^2 after 1000 samples using prn generator Eq. (A.14). The expected number of χ^2 in each bin is 10. The value of the bin indicated by (*) is offscale.

See “Numerical Recipes”

- **Chi-squared statistic** is

$$\chi^2 = \sum_i \frac{(N_i - n_i)^2}{n_i}$$

- **n_i (N_i) is the expected (observed) number in bin i .**
 - N_i is an integer! (Omit terms with $0 = n_i = N_i$)
 - Term with $n_i = 0$ and $N_i \neq 0$, correctly give $\chi^2 = \infty$.)
 - Large values of χ^2 indicate the “null” hypothesis, i.e. unlikely.
- The **chi-squared probability** is $Q(\chi^2|N-1)$
($N-1$) because of “sum rule”. The incomplete Γ -fct
N.R. Sec. 6.2 and 14.3
 - Table may help with interpreting the **chi-squared probability**.

<i>Probability</i>	<i>Interpretation</i>
< 1 %	reject
1 - 5 %	suspect
5 - 10 %	almost suspect
10 - 90 %	pass
90 - 95 %	almost suspect
95 - 99 %	suspect
> 99 %	reject

Recommendation: For careful work use several generators!

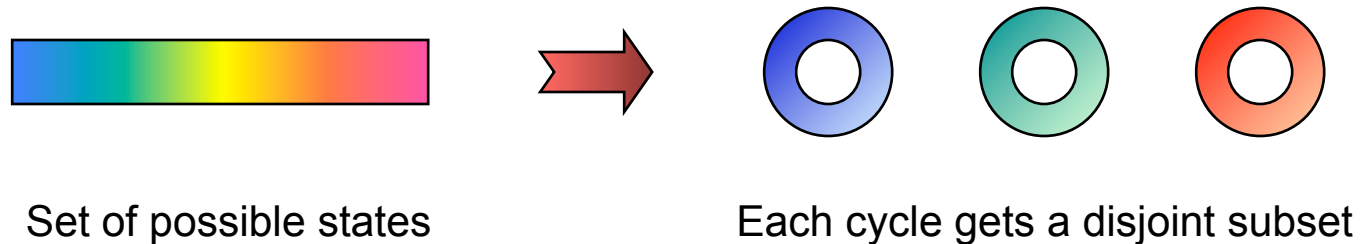
- For most real-life MC simulations, *passing existing statistical tests is necessary but not sufficient.*
- Random number generators are still a **black art**.
- **Rerun with different generators**, since *algorithm may be sensitive to different RNG correlations.*
- **Computational effort is not wasted**, since results can be *combined to lower error bars.*

Parallel Simulations

- Parallel Monte Carlo is easy? Or is it?
- Two methods for easy parallel MC:
 - **Cloning**: same serial run, different random numbers.
 - **Scanning**: different physical parameters (density,...).
- Any parallel method (MPI, ..) can be used.
- Problems:
 - Big systems require excessive wall clock time.
 - Excessive amounts of output data generated.
 - Random number correlation?

Parallelization of RNGs

- Cycle Division: Leapfrog or Partition. Correlation problems
- Cycle Parameterization: If PRNG has several cycles, assign different **cycles** to each stream



- Iteration Parameterization: Assign a different iteration function to each stream

Examples of Parallel MC codes and Problems

- Embarrassingly parallel simulations. Each processor has its own simulation. **Scanning** and **cloning**.

Unlikely to lead to problems unless they stay in phase.

- Lots of integrals in parallel (e.g. thousands of Feynman diagrams each to an accuracy of 10^{-6}).

Problem if cycle length is exhausted.

- Particle splitting with new particles forking off new processes. Need lots of generators.

Problem if generators are correlated initially.

- Space-time partitioning. Give each local region a processor and a generator.

Problem if generators are correlated.