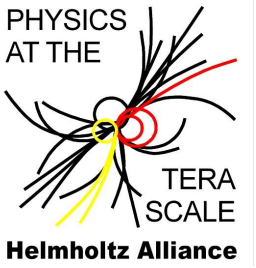
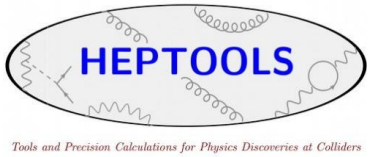
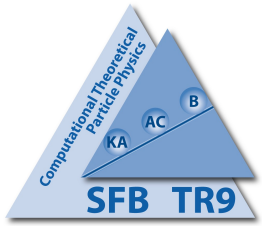
Monte Carlo Methods in High Energy Physics II

Peter Uwer







Contents

Generation of random numbers

So far: MC = method which uses random numbers *)

Where do they come from ?

→ from any truly random process!

Examples:

- Radioactive decay
- Electronic noise
- Silicon Graphics lava lamp

*) In fact it turns out that random numbers which are “less random” are also very useful → quasi random numbers…

Random number generation

→ Generators relying on truly random process are called hardware random generators

Note that it is easy to remove any bias from the hardware generator using a trick:

Suppose the generator generates 0's and 1's with probability p(0) and p(1).

If two following bits are equal skip them, if not keep the second one → random sequence {0,1}

→ possible to construct hardware random number generator without detailed knowledge of the probabilities

Problem:

Hardware generation of random numbers are often slow

LavaRng: 200kbit/s

Random numbers

Requirements

- should be “random”
- should be reproducible
- should be fast
- should be portable (not the question whether the Lava lamp fits in your pocket…)
- should have long period

Why don’t produce them directly on the computer ???

…since they are supposed to be random…

History: The mid square method

"Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin." J. von Neumann

v. Neumann proposed the following algorithm:

Take a large number with # digits, square it and take the middle as the random number

"Quality" of the random numbers rather poor, in particular if you start producing 0's they will start to repeat over an over again, there is also a finite period… at least it is portable…

Clear:

"Calculated random numbers" are not random, but they look random!

→ "Pseudo-random numbers"

(Mixed) Linear Congruential generators

$$x_i = (a x_{i-1} + c) \bmod m$$

[Lehmer]

→ maximum period: m/4 (from number theory)
only reached for special choice!

Usually m is taken to be 2^t, were t is the number of bits in the integer representation (→rather short period for t=32) 2^{32}=4,294,967,296

Problematic:

- Famous and infamous choices for a,c
- Short period
- Random numbers fall into hyperplanes

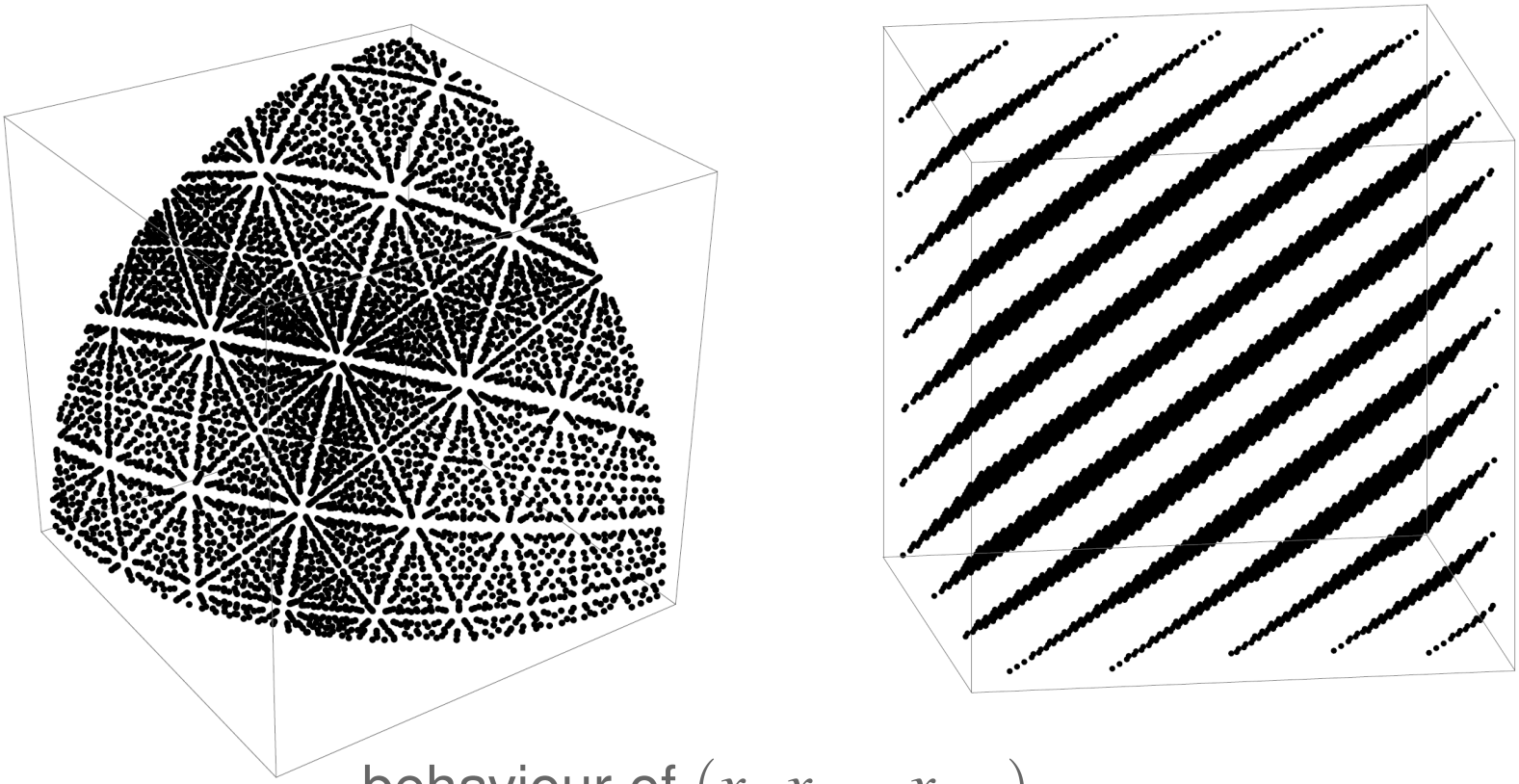
[Marsaglia effect]

→ improvements required

Non-random behaviour a simple example

Simple example: $x_i = (ax_{i-1} + c) \bmod m$

$x_0 = 1234, a = 106, c = 1283, m = 6075$



behaviour of (x_i, x_{i+1}, x_{i+2})

Marsaglia effect

If consecutiv output of linear congruential RNG is interpreted as vector in d dimensional space, points fall into hyper planes

Maximum number of hyperplanes $= (d!\, 2^t)^{1/d}$				
Number of bits(t)	$d=3$	$d=4$	$d=6$	$d=10$
16	73	35	19	13
32	2953	566	120	41
36	7442	1133	191	54
48	119086	9065	766	126
60	1905376	72520	3064	290

[Marsaglia]

(Lagged) Fibonacci type RNG's

Improve period and quality by combining two random numbers

$$x_{j+1} = x_j + x_{j-1} \bmod m \qquad \text{Fibonacci}$$

$$x_j = x_{j-p} + x_{j-q} \bmod m \qquad \text{Lagged Fibonacci}$$

Generalisation

$$x_j = x_{j-p} \otimes x_{j-q} \bmod m$$

$$\otimes \in \{+, -, *, \oplus = \text{xor}\}$$

Special case: Shift-register, Tausworth Generator

$$x_j = x_{j-p} \oplus x_{j-q} \bmod 2$$

→ production of random bits

Subtract-with-carry

Marsaglia, Zaman:

$$\Delta_n = x_{n-s} - x_{n-r} - c_{n-1}, \qquad (2.1)$$

and then determines x_n and c_n through

$$\begin{aligned} x_n &= \Delta_n, & c_n &= 0 \quad \text{if} \quad \Delta_n \geq 0, \\ x_n &= \Delta_n + b, & c_n &= 1 \quad \text{if} \quad \Delta_n < 0. \end{aligned} \qquad (2.2)$$

Further improvements by Lüscher: RanLux

→ very long period + theory behind

How to test random numbers?

Many sophisticated tests exist

- test of uniformity, χ^2-tests
- Serial correlation test
- Gap test (uniformity of sequences)
- Bit level tests
- …

Important:

Every application represents a new test which might be the first which fails

Test of uniformity, χ^2-test

Generate n numbers in [0,1), multiply by ν and truncated to integers, count the number of occurrence m_ν of each ν

Calculate χ^2:

$$\chi^2 = \sum_{i=1}^{\nu} \frac{(m_i - \langle m \rangle)^2}{Var(m)} = \sum_{i=1}^{\nu} \frac{(m_i - \langle m \rangle)^2}{\langle m \rangle}$$

Poisson distribution

Compare with χ^2-distribution:

$$f(z = \chi^2, n = \nu - 1) = \frac{z^{n/2-1} e^{-z/2}}{2^{n/2}\Gamma(n/2)}$$

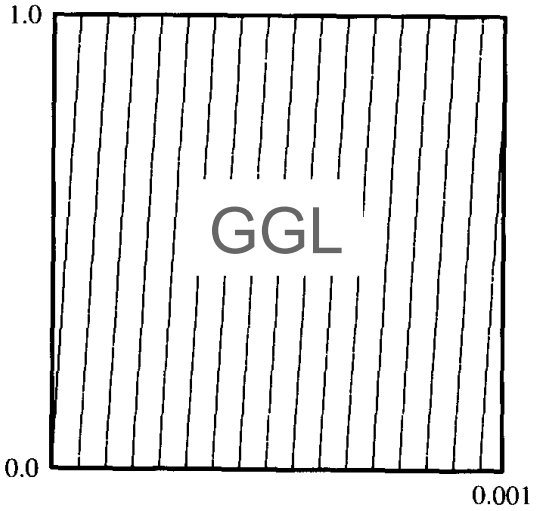
→ Tells us the probability of our observation

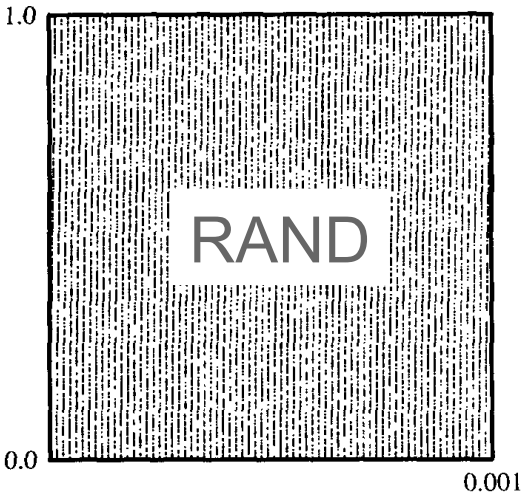
Testing random numbers

[Vattulainen, Kankaala, Saarinen, Ala-Nissila 95]

Table 6
A summary of the performance of the tested generators in standard and bit level tests

Test method	Random number generator							
	GGL	RAND	RANF	G05FAF	R250	RAN3	RANMAR	RCARRY
Standard tests	+	0	+	+	+	0	–	–
Bit level tests	+	–	0	+	+	–	0	0





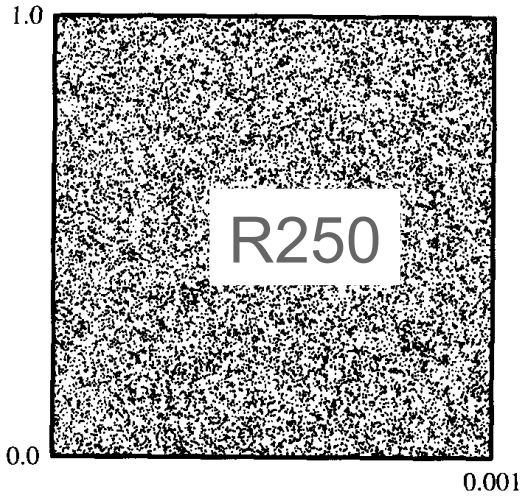


Fig. 1. Spatial distribution of 20 000 random number pairs in two dimensions on a thin slice of a unit square as generated by GGL (a), RAND (b) and R250 (c).

R250 seems to be a rather good choice

Every simulation is a new test…

[Ferrenberg, Landau, Wong '92]

TABLE I. Values of the internal energy (top) and specific heat (bottom) for ten independent runs with $L=16$ at K_c obtained using the Wolff algorithm. The last number in each column, labeled "dev," gives the difference between the simulation value and the exact value, measured in terms of the standard deviation σ of the simulation.

	CONG	R250	R1279	SWC	SWCW
	1.453089	1.455096	1.453237	1.452321	1.453058
	1.453107	1.454697	1.452947	1.452321	1.453132
	1.452866	1.455126	1.453036	1.452097	1.453330
	1.453056	1.455011	1.452910	1.452544	1.453219
	1.453035	1.454866	1.453040	1.452366	1.452828
	1.453198	1.455054	1.453065	1.452388	1.453273
	1.453032	1.454989	1.453129	1.452444	1.453128
	1.453169	1.454988	1.453091	1.452321	1.453083
	1.452970	1.455178	1.453146	1.452306	1.453216
	1.453033	1.455162	1.452961	1.452093	1.453266
$-<E>$	1.453055	1.455017	1.453056	1.452320	1.453153
error	0.000030	0.000046	0.000032	0.000044	0.000046
dev.	-0.31σ	42.09σ	-0.27σ	-16.95σ	1.94σ
	1.499210	1.447436	1.497665	1.515966	1.497988
	1.498099	1.451072	1.498049	1.515966	1.497813
	1.498866	1.446619	1.497026	1.514664	1.496413
	1.499150	1.447657	1.498608	1.512534	1.497631
	1.499907	1.450726	1.499018	1.513009	1.499337
	1.498127	1.447349	1.497292	1.513267	1.496294
	1.498484	1.448782	1.498314	1.512298	1.496332
	1.498532	1.449522	1.498801	1.513575	1.497203
	1.499409	1.449012	1.496602	1.516258	1.498850
	1.498814	1.448098	1.497887	1.514838	1.496123
$<C>$	1.498860	1.448627	1.497926	1.514237	1.497398
error	0.000182	0.000467	0.000250	0.000473	0.000356
dev.	0.82σ	-107.16σ	-3.14σ	32.81σ	-3.68σ

Simulation of the Ising model using a cluster algorithm, using different RNG's

Standard C/C++ random number generator drand48()

$$x_{n+1} = (a x_n + c) \text{ mod } m$$

$$m = 2^{48}$$

$$a = 0x5deece66d$$

$$b = 0xb$$

Important lesson

Never try to write your own random generator unless are real expert!

Modern generators come with a lot of theory based on number theory and the study of chaotic systems

→ see for example Ranlux [Lüscher]

The fact that your algorithm looks completely random does not guarantee the numbers are “random”

in fact today the opposite is true: the good generators are rather simple since then we can study their properties

→ see Knuth’s book for an instructive example

Closing remark on random number from the “pope” of random numbers:

"A random number generator is much like sex:
when it's good it's wonderful,
and when it's bad it's still pretty good."
G. Marsaglia

Generating non-uniform distributions

General algorithms to produce non-uniform distributions from uniform random variables:

- Transformation
- Hit and Miss / Acception and Rejection
- Sampling by composition (“Multichannel”)
- Metropolis
- …

The first 3 are the most important for us

Transformation

$$F(u) = \int_{-\infty}^{u} f(x)dx$$

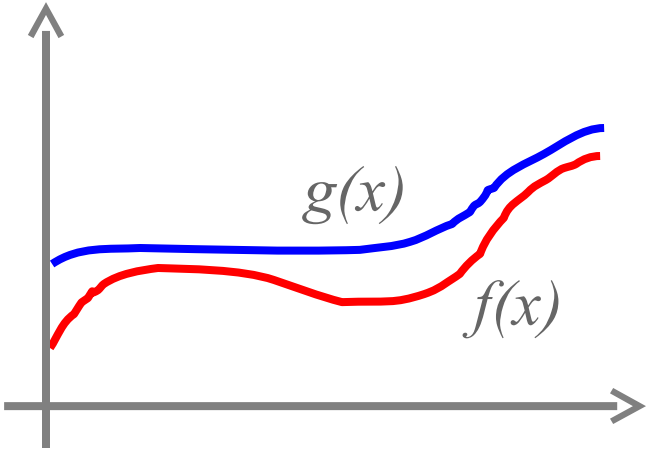
If F^{-1} exist then generate numbers according to

$$x_i = F^{-1}(u_i)$$

with ui uniform in [0,1).

→ Proof + examples

Hit and miss



Algorithm:

1. Generate xi according to $g(x)$
2. Generate y uniform between 0 and $g(xi)$
3. Return xi if $y < f(x)$, else goto 1

→ the better $g(x)$ matches $f(x)$ the more efficient the alg. i

→ modified version used in event generators to “unweight” events

Sampling by composition

How to generate:

$$f(x) = \sum_{i=1}^{k} p_i f_i(x)$$

?

with

$$\int dx f_i(x) = 1, \quad \sum_i p_i = 1, \quad p_i > 0$$

Algorithm:

1. Chose z from $\{1,\ldots,k\}$ with probability pi
2. Generate y according to $f_z(x)$ ← Transform, Hit and Miss,…

Metropolis algorithm

Methods presented so far not useful when generating

$$p(x_1, x_2, \ldots, x_n)$$

with n large (Ising model, lattice gauge theory…)

If ergodicity and detailed balance is satisfied then the following algorithm gives samples following p (after warming up):

- start with arbitrary $\phi_0 = (x_1, x_2, \ldots x_n)$
- generate a new ϕ_1 from ϕ_0
- calculate $\Delta s = -\ln(p(\phi_1)/p(\phi_0))$
- if $\Delta S > 0$ replace ϕ_0 by ϕ_1
- if $\Delta S < 0$ accept the new candidate with probability $p(\phi_0)/p(\phi_1)$

detailed balance: $p(\phi_1)W(\phi_1 \to \Phi_2) = p(\phi_2)W(\phi_2 \to \phi_1)$ ergodicity: every configuration can be reached

Sampling discrete probabilities

For uniform distribution we have

$$f(x) = 1, F(x) = \int dx f(x) = x$$

$$p(0 \leq x_1 \leq y \leq x_2 \leq 1) = F(x_2) - F(x_1) = x_2 - x_1$$

- To generate discrete probabilities pi decompose [0,1) in subintervalls of length pi,
- Generate uniform distributed variable ([0,1))
- Return the event of the corresponding subintervall

Normal distribution

Box-Muller method

$$x = [-2\ln(1-u_1)]^{1/2}\cos(2\pi u_2)$$

$$y = [-2\ln(1-u_1)]^{1/2}\sin(2\pi u_2)$$

→ Proof

from the central limit theorem:

$$x = \sqrt{\frac{12}{N}}\left(\sum_{k=1}^{N} u_k - \frac{N}{2}\right)$$

N=6: $$x = \sum_{k=1}^{N} u_k - 6$$

Breit-Wigner / Cauchy distribution

$$f(x) = \frac{1}{\pi(1+x^2)}$$

Transformation:

$$F(x) = \frac{1}{2} + \frac{1}{\pi}\arctan(y)$$

$$x = F^{-1}(u) = \tan(\pi(u - 1/2))$$

Alternative:

$$x = \frac{u_1}{u_2}$$

Gerneral case

$$f(y) = \frac{\beta}{\pi(\beta^2 + (y-\alpha)^2)}$$

from

$$y = \alpha + \beta x$$

Special tricks

$$f(z) = 6z(1-z)$$

is obtained from $z = \text{mid}(u_1, u_2, u_3)$ with u_i uniform

→ Proof

$$f(z) = kz^{k-1}$$

$$F(z) = z^k$$

from $z = \max\{u_1, \ldots, u_k\}$

→ Proof

The End