Random number generators

Marcin Chrząszcz
mchrzasz@cern.ch

University of Zurich[UZH]

Monte Carlo methods,
17 March, 2016

Random and pseudorandom numbers

John von Neumann:

”Any one who considers arithmetical methods of producing random digits is, of course, in a state of sin. For, as has been pointed out several times, there is no such thing as a random number — there are only methods to produce random numbers, and a strict arithmetic procedure of course is not such a method.”

$\Rrightarrow$ Random number: a given value that is taken by a random variable

$\twoheadrightarrow$ by definition cannot be predicted.

$\Rrightarrow$ Sources of truly random numbers:

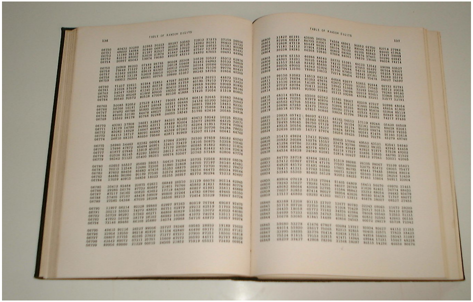
- Mechanical
- Physical

$\Rrightarrow$ Disadvantages of physical generators:

- To slow for typical applications, especially the mechanical ones!
- Not stable; small changes in boundary conditions might lead to completely different results!

Random numbers - history remark

⇛ In the past there were books with random numbers:



⇛ It's obvious that they didn't become very popular ;)

⇛ This methods are comming back!

↠ Storage device are getting more cheap and bigger (CD, DVD).

↠ 1995: G. Marsaglia, $650\mathrm{MB}$ of random numbers, "White and Black Noise".

Pseudorandom numbers

Commercially available physical generators of random numbers are usually based on electronic noise. This kind of generators do not pass simple statistical tests! Before you use them check they statistical properties.

⇛ Pseudorandom numbers- numbers generated accordingly to strict mathematical formula.

⇛ Strictly speaking they are non random numbers, how ever they have all the statistical properties of random numbers.

⇛ How ever modern generators are so good that no one can distinguish the pseudo random numbers generated by then from true random numbers.

⇛ Mathematical methods of producing pseudorandom numbers:

- Good statistical properties of generated numbers.
- Easy to use and fast!
- Reproducible!

⇛ Because of those properties the truelly random numbers are not used in practice any more!

Middle square generator; von Neumann

⇛ The first mathematical generator (middle square) was proposed by von Neumann (1964).

↬ Formula: $$\boxed{X_n = \lfloor X_{n-1}^2 \cdot 10^{-m} \rfloor - \lfloor X_{n-1}^2 \cdot 10^{-3m} \rfloor \cdot 10^{2m}}$$

↬ where X_0 is a constant (seed), $\lfloor \cdot \rfloor$ is the cut-off of a number to integer.

⇛ Example:

Let's put $m = 2$ and $X_0 = 2045$:

↬ $X_0^2 = \underbrace{04}_{\text{rej}} 1820 \underbrace{25}_{\text{rej}}$ $\Rightarrow X_1 = 1820$

↬ $X_1^2 = \underbrace{03}_{\text{rej}} 3124 \underbrace{00}_{\text{rej}}$ $\Rightarrow X_1 = 3124$

↬ Simple generator but unfortunately quite bad generator. Firstly the sequences are very short and strongly dependent on the X_0 number.

Middle square generator; von Neumann

⇛ This was a first generator written and it's a good example how to not write generators.

⇛ It's highly non stable!

```
mchrzasz-ThinkPad-W530% python gen.py 14714 4
21650.0
46872.0
219698.0
4826721.0
2329723538.0
5.42761170924e+14
2.94589685716e+25
8.67830820626e+46
7.53130325698e+89
Traceback (most recent call last):
  File "gen.py", line 29, in <module>
    sys.exit(main())
  File "gen.py", line 22, in main
    tmp=X0**2
OverflowError: (34, 'Numerical result out of range')
```

⇛ E 4.1 Write the von Neumann Middle square generator.

General schematic

⇛ Typical MC generator layout:

- We choose initial constants: X_0, X_1, ... X_{k-1}.
- The k number if calculated based on the previous ones:

$$X_k = f(X_0, ..., X_{k-1}),$$

⇛ Typically one generates $0/1$ which are then converted towards double precision numbers with $\mathcal{U}(0,1)$.

⇛ Generator period (P, l integer numbers): P is the period:

$$\exists_{l,P} : X_i = X_{i+j \cdot P} \;\; \forall_{j \in \mathbb{I}^+} \; \forall_{i>l}$$

⇛ In post of the cases the period can be calculated analytically, although this is sometimes not trivial.

⇛ There is a recommendation about the period of a generator. For N numbers we usually require:

$$N \ll P$$

⇛ In practice: $N < P^{2/3}$ is oki ;)

⇛ For example a generator "Mersenne Twister" (Matsumoto, Nishimura, 1998): $P \sim 10^{6000}$.

Linear generators

⇛ General equation:

$$X_n = (a_1 X_{n-1} + a_2 X_{n-2} + ... + a_k X_{n-k} + c) \bmod m,$$

↬ where a_i, c, m are parameters of a generator(integer numbers).

↬ Generator initialization ⇄ setting those parameters.

⇛ Very old generators. (often used in Pascal, or first C versions):

$$k = 1: \;\; X_n = (aX_{n-1} + c) \bmod m,$$

$$c = \begin{cases} = 0, \text{multiplicative geneator} \\ \neq 0, \text{mix geneator} \end{cases}$$

⇛ The period can be achieved by tuning the seed parameters (multiplicative) :

$$P_{\max} = \begin{cases} 2^{L-2}; \text{ for } m = 2^L \\ m-1; \text{ for } m = \text{prime number} \end{cases}$$

Linear generators

⇛ Some simple linear generators and their periods:

a	c	m	Name/author
2^16+3	0	2^{31}	`RANDU`
$2^2 \cdot 23^7 + 1$	0	2^{35}	Zielinski (1966)
69069	1	2^{32}	Marsaglia (1972)
16807	0	$2^{31}-1$	Park, Miller (1980)
40692	0	$2^{31}-249$	L' Ecuyer (1988)
68909602460261	0	2^{48}	Fishman (1990)

⇛ m - prime number $\rightarrow$ better statistical properties. ⇛ There are some quid lines how to choose the parameters to make the period larger.

The periods of $2^{32} \sim 4 \cdot 10^9$ are not good enough for modern applications!
Remember that in practice $N \ll P^{2/3}$!

⇛ Simple linear generators do not pass newer statistical tests!

Linear generators

⇛ Marsaglia (1995) generators:

1. $X_n = (1176X_{n-1} + 1476X_{n-2} + 1776X_{n-3}) \bmod m, \ m = 2^{32} - 5$
2. $X_n = 2^{13}(X_{n1} + X_{n2} + X_{n3}) \bmod m, \ m = 2^{32} - 5$
3. $X_n = (1995X_{n1} + 1998X_{n2} + 2001X_{n3}) modm, \ m = 2^{35}849$
4. $X_n = 2^{19}(X_{n1} + X_{n2} + X_{n3}) modm, \ m = 2^{32}1629$

⇛ $P = m^3 - 1$ ⇛ They got surprisingly good statistical properties! ⇛ The main disadvantage is that multidimensional distributions look very suspicious:

$$U_i = X_i/m, \ i = 1, 2... \Rightarrow U_i(0,1)$$
$$(U_1, U_2, ..., U_k), (U_2, U_3, ..., U_{k+1}), ...(U_1, U_2, ..., U_k), (U_{k+1}, U_{k+2}, ..., U_{2k}), ...$$

are being located on a resurfaces in a hiper-cube $[0,1]]^k$.

⇛ Using Fourier analysis one can find the distances between the hiper-surfaces.

⇛ Generalization for multiple dimensions:

$$\boxed{X_n = \mathbf{A}\overrightarrow{X}_{n-1} \bmod m,}$$

⇛ E4.2 Code all 4 Marsaglia generators.

Shift register generator

⇛ General equation:

$$b_n = (a_1 X_{n-1} + a_2 X_{n-2} + ... + a_k X_{n-k} + c) \bmod 2,$$

where $a_i \subset (\{0,1\})$

⇛ Super fast and easy to implement due to: $(a+b) \bmod 2 = a \text{ xor } b$

a	b	a xor b
0	0	0
1	0	1
0	1	1
1	1	0

⇛ Maximal period is $2^k - 1$.

⇛ Example (Tausworths generator):

$a_p = a_q = 1$, other $a_i = 0$ and $p > q$. Then: $b_n = b_{n-p} \text{ xor } b_{n-q}$

⇛ How to get numbers from bits (for example):

$U_i = \sum_{j=1}^{L} 2^{-j} b_{is+j},\ s < L$.

Fibonacci generator

⇛ In 1202 Fibonacci with Leonardo in Piza:

$$f_n = f_{n-2} + f_{n-1},\ n \geqslant 2$$

⇛ Based on this first generator was created (Taussky and Todd, 1956):

$$X_n = (X_{n-2} + X_{n-1}) \bmod m,\ n \geqslant 2$$

This generator isn't so good in terms of statistics tests.
⇛ Generalization:

$$X_n = (X_{n-r} \odot X_{n-s}) \bmod m,\ n \geqslant r,\ s \geqslant 1$$

$\odot$	P_{max}	Stat. properties
$+,-$	$(2^r-1)2^{L-1}$	good
x	$(2^r-1)2^{L-13}$	very good
xor	(2^r-1)	poor

MZT

⇛ Popular generator MZT, better known as RANMAR (Marsaglia, Zaman, Tsang, 1990):

- Very universal! Will give the same results on all computers that have integer numbers with $\geqslant 16$ bit and floating with $\leqslant 24$ bits.

⇛ It's effectively a combination of two generators:

- The Fibonacci:

$$F(97, 33, \bullet) \rightarrowtail V_n \in [0, 1)$$

where

$$x \bullet y = \begin{cases} x - y, & x \geqslant y \\ x - y + 1, & x < y \end{cases}$$

- The initialization is done by setting V_i ,$i = 1, ..., 97$ numbers.
- They are initialized by bits: $V_1 = 0.b_1 b_2 ... b_{24}$, $V_2 = 0.b_{25} ... b_{48}$,...
- The series b_n is generated via two generators:

$$\left\{ \begin{array}{l} y_n = (y_{n-3} \cdot y_{n-2} \cdot y_{n-1}) \mathrm{mod} 179 \\ z_n = (53 z_{n-1} + 1) \mathrm{mod} 169 \end{array} \right\} \Rightarrow b_n \left\{ \begin{array}{ll} 0, & (y_n \cdot z_n) \mathrm{mod} 64 < 32 \\ 1, & (y_n \cdot z_n) \mathrm{mod} 64 \geqslant 32 \end{array} \right\}$$

- Initialization: provide 4 numbers 4: $y_1, y_2, y_3 \in 1, ...178$, $z_1 \in 0, ..., 168$
- Period $P = 2^{120}$

MZT

$\Rrightarrow$ The second generator $c_n \in (0,1)$:

$$c_n = c_{n-1} \circ (7654321/16777216), \quad n \geqslant 2,\ c_1 = 362436/16777216,$$

where:

$$c \circ d = \left\{ \begin{array}{ll} c - d, & c \geqslant d \\ c - d + (16777213/16777216), & c < d \end{array} \right\}, c, d \in [0,1)$$

$\Rrightarrow$ Period: $P = 2^{144}$ $\Rrightarrow$ The full MZT generator is calculated:

$$U_n = V_n \bullet c_n$$

- Period $P = 2^{144} \sim 10^{43}$

$\Rrightarrow$ It fulfils all know statistical test! $\Rrightarrow$ E4.3 Code the Fibonacci generator $\Rrightarrow$ A4.1 Code the RANMAR generator.

Multiply with carry, generator

$\Rrightarrow$ We start from:

$$X_n = (a_1 X_{n-1} + a_2 X_{n-2} + ... + a_k X_{n-k} + c) \bmod m,$$

where $a_1, .., a_k \in \mathbb{N}$ are constant parameters.
$\Rrightarrow$ The c parameters is calculated foe each step:

$$c = \lfloor (a_1 X_{n-1} + a_2 X_{n-2} + ... + a_k X_{n-k} + c)/m \rfloor,$$

$\Rrightarrow$ Initialization: $a_1, .., a_k, c$.
$\Rrightarrow$ Advantages:

- Fast and easy to implement.
- Large period.
- Good statistical properties.
- First proposed by Marsaglia.

Multiply with carry, generator, example

⇛ MWC1:

$$\left.\begin{aligned} X_n &= (18000X_{n-1} + c_x)\mathrm{mod}2^{16} \\ Y_n &= (30903Y_{n-1} + c_y)\mathrm{mod}2^{16} \end{aligned}\right\} \quad 16 - \text{bit digits}$$

$$\Rightarrow Z_n = b_1^{X_n}...b_{16}^{X_n}b_1^{Y_n}...b_{16}^{Y_n} \quad 32 - \text{bit digits}$$

⇛ Period: $2^{60} \sim 10^{18}$

⇛ MWC2:

$$\begin{aligned} X_n = (12013X_{n-8} + 1066X_{n-7} + 1215X_{n-6} + 1492X_{n-5} + 1776_{Xn-4} \\ +1812X_{n-3} + 1860X_{n-2} + 1941X_{n-1} + c_X) \bmod 2^{16} \end{aligned}$$

$$\begin{aligned} Y_n = (9272Y_{n-8} + 7777Y_{n-7} + 6666Y_{n-6} + 5555Y_{n-5} + 4444Y_{n-4} \\ +3333Y_{n-3} + 2222Y_{n-2} + 1111Y_{n-1} + c_Y) \bmod 2^{16} \end{aligned}$$

$$\Rightarrow Z_n = b_1^{X_n}...b_{16}^{X_n}b_1^{Y_n}...b_{16}^{Y_n} \quad 32 - \text{bit digits}$$

⇛ Period: $2^{250} \sim 10^{75}$

⇛ E4.4 Code the MWC1 and MWC2.

Subtract with borrow, generator

⇛ Created again by Marsaglia (1991):

$$X_n = (X_{n-r} \ominus X_{n-s}) \bmod m, \ r, s \in \mathbb{N},$$

where :

$$x \ominus y = \begin{cases} x - y - c + m, \ c = 1, \ \text{when x} - \text{y} - \text{c} < 0 \\ x - y - c, \ c = 0, \ \text{when x} - \text{y} - \text{c} \geqslant 0 \end{cases}$$

⇛ Initialization: $X_1, ..., X_{n-r}$ and $c = 0$.

⇛ Fast and easy :)

⇛ Fails some of the basic statistics tests.

Non linear generators

⇛ The natural solutions to problems of linear generators are the non-linear generators (second part of 1980s).

⇛ Eichenauera i Lehna (1986):

$$X_n = (aX_{n1}^{-1} + b) \bmod m,$$

⇛ Eichenauera-Hermanna (1993)

$$X_n = [a(n + n_0) + b]^{-1} \bmod m,$$

⇛ L. Blum, M. Blum, Shub (1986):

$$X_n = X_{n-1}^2 \bmod m,$$

↣ Very popular in cryptography.

⇛ Pros and cons:

- They all pass all statistical tests.
- Much slower then linear generators.

RANLUX generator

⇛ All described generators are based on some mathematical algorithms and recursion. The typical scheme is of constructing a MC generator:

- Think of a formula that takes some initial values.
- Generate large number of random numbers and put them through statistical tests.
- If the test are positive we accept the the generator.

⇛ Now let's think: why the hell numbers obtained that way are showing some random number properties?

RANLUX generator

⇛ All described generators are based on some mathematical algorithms and recursion. The typical scheme is of constructing a MC generator:

- Think of a formula that takes some initial values.
- Generate large number of random numbers and put them through statistical tests.
- If the test are positive we accept the the generator.

⇛ Now let's think: why the hell numbers obtained that way are showing some random number properties? There is no science behind it, it's pure luck!
⇛ M.Luscher (1993) hep-lat/9309020
⇛ Generator RANLUX based on Kolomogorow entropy and Lyapunov exponent. Effectively we are building inside the generator the chaos theory.
⇛ RANLUX and Mersenne Twister (TRandom1, TRandom3) are the 2 most powerful generators in the world that passed every known statistical test.

Chaos theory in a nut shell

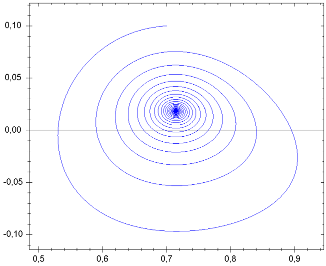
⇛ We know that the solution of classical systems is described by trajectory in phase spaces. Now the problem with this picture starts to be when arround one point in this phase space we are getting more and more trajectories that are drifting a part later on.

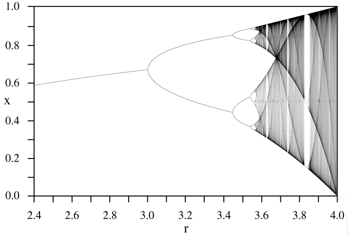
⇛ The Lyapunov exponent tells us how a two solutions drift apart with time:

$$|\delta X(t)| \approx e^{\lambda t} |\delta X_0|$$

⇛ Kolomogorow entropy:

$$h_K = \int_P \lambda d\mu$$





Wrap up

⇛ Things to remember:

- Computer cannot produce random numbers, only pseudorandom numbers.
- We use pseudorandon numbers as random numbers if they are statistically acting the same as random numbers.
- Linear generators are not commonly used nowadays.
- State of the art generators are the ones based on Kolomogorows theorem.

Backup