UPPSALA
UNIVERSITET

Computational Methods for Statistics with Applications

Maya Neytcheva

Department of Information Technology

Uppsala University

NGSSC
FORSKARSKOLA I TEKNISK-VETENSKAPLIGA BERÄKNINGAR
NATIONAL GRADUATE SCHOOL IN SCIENTIFIC COMPUTING

UPPSALA
UNIVERSITET

Random Numbers

- Many algorithms are based on the availability of random numbers.
  Thus, we need random number generators.
- How to generate those?
  - Physical (hardware) methods: dice,coin flipping, roulette (too slow for applications in statistics or in cryptography)
    atmospheric noise, cosmic noise, radioactive decay (`http://www.random.org/`
    However: these are not reproducible. Also, Strong bias could occur due to external reasons (electromagnetic field)
  - Computer-generated random numbers (are these random?)

Pseudo-Random Numbers

Definition: A uniform pseudo-random number generator is an algorithm, which starts from some initial value x_0 and uses particular transformation R in order to produce a sequence of numbers $\{x_k\} = \{R^k(x_0)\}, k = 1, \cdots, n$ with values in $[0, 1]$. For each n, the generated sequence has to reproduce the behaviour of an iid sample of uniformly distributed random variables, when tested through conventional tests.

Definition: Period of a random number generator: the smallest integer P such that

$$x_{k+P} = x_k$$

UPPSALA
UNIVERSITET

Desired properties of a good generator:

- long periods
- low auto correlations
- produce samples which appear to be drawn from a uniform distribution

UPPSALA UNIVERSITET

Some pseudo-random number generation techniques

The congruential method (linear, multiplicative)
If the last pseudo-random number generated was X_{k-1}, then the next number is generated by evaluating

$$X_k = (X_{k-1} * L + I) mod\, M,$$

where L is a constant multiplier, I is a constant increment, and M is a constant modulus.
After being generated, the numbers are converted to uniform deviates by $x_k = X_k/M$.
It can be shown that for large M, $U = X/M$ has asymptotically the moments of $U(0, 1)$.

UPPSALA UNIVERSITET

Some famous multipliers

$L = 23; M = 10^8 + 1$ The original formula used by Lehmer (1948).

$L = 65539; M = 2^{29}$ The algorithm RANDU, used by IBM. Catastrophic higher-order correlations. Now it is know that any multiplyer, whos binary representation ends with '101', whould have been better.

$L = 69069; M = 2^{32}$ Recommended by Marsaglia (1972).

$L = 1664525; M = 2^{32}$ The best multiplier for $M = 2^{32}$ according to criteria of Knuth (1981).

$L = 5^{15}, M = 2^{47}$ Used by CDC, 48-bit integer arithmetic. The period is $\approx 2^{13}$.

An example (borrowed from Jesper Ryden, Math. inst, UU)

$$X_k = (X_{k-1} * L) mod M$$

```
M=7, L=3, x=2
for (k in 1:10){
      x=L*x%%M
      u[k]=x/M
}
```

Bad case: $M = 29241$, $L = 171$: $29241 mod 171 = 0$!

(Period length: at least $2^{30} = 1073741824$)

Portability issues

Shift-register generators

Lagged Fibonacci sequence: each number is the result of some arithmetic or logical operation between two numbers which have occured somewhere in the sequence

$$s_i = (s_{i-p} \cdot s_{i-q}) mod m$$

Very little theory about the distribution of such numbers in general.

UPPSALA
UNIVERSITET

KISS (Keep It Simple, Stupid)

$$X_k = (a_1 X_{k-1} + a_2 X_{k-2} + \cdots + a_d X_{k-d}) mod\, 2$$

a combination of shift-register and congruential generator.

UPPSALA
UNIVERSITET

Linear Feedback Shift Register (LFSR)

The linear feedback shift register (LFSR) is a shift register whose input bit is a linear function of its previous state.
The only linear functions of single bits are xor and $inverse - xor$; thus it is a shift register whose input bit is driven by the exclusive-or (xor) of some bits of the overall shift register value.
$x^{19} + x^{18} + x^{17} + x^{14} + 1$ - period: 524287

UPPSALA
UNIVERSITET

The inverse method

Definition: For an increasing real (cumulative distribution) function F, the generalized inverse $F^{\dagger}$ is defined as

$$F^{\dagger} = inf\{x, F(x) \geq u, 0 \leq u \leq 1\}$$

Theorem: If $X \sim U(0,1)$ then the random variable $F^{\dagger}(X)$ has distribution F.

The inverse method

The inverse transform sampling method works as follows:
1. Generate a random number from the standard uniform distribution; call this u.
2. Compute the value x such that $F(x) = u$.
3. Take x to be the random number drawn from the distribution described by F.

Expressed differently, given a continuous uniform variable $U \in [0, 1]$ and an invertible distribution function F, the random variable $X = F^{\dagger}(U)$ has distribution F.

Uniform rejection

Algorithm:

1: Let $X \sim U(a, b)$ and $Y \sim U(0, 1)$. Simulate x and y.

2: Accept x as a simulated value from $f(x)$ if $y < f(x)$, otherwise reject.

Accept - reject

Target density $f(x)$,instrumental density $g(x)$.
Assume $f(x) \leq M\, g(x)$ for any x.
Current state x_k
Generate x from $g(x)$, compute $r(x) = f(x)/(Mg(x))$
Generate $U \sim U(0,1)$

$$x_{k+1} = \begin{cases} x & if U \leq r(x) \\ \text{reject and new trial} & otherwise \end{cases}$$

The generated sample is an iid sample from the target distribution.

RANMAR

In the first step it employs a lagged Fibonacci generator,

$$X_i = \begin{cases} X_{i-97} - X_{i-33}, & \text{if} X_{i-97} \geq X_{i-33}, \\ X_{i-97} - X_{i-33} + 1, & \text{otherwise.} \end{cases}$$

Only 24 most significant bits are used for single precision reals. The second part of the generator is a simple arithmetic sequence for the prime modulus $2^{24} - 3 = 16777213$,

$$Y_i = \begin{cases} Y_i - c, & \text{if} Y_i \geq c, \\ Y_i - c + d, & \text{otherwise,} \end{cases}$$

where $c = 7654321/16777216$ and $d = 16777213/16777216$. The final random number Z_i is then produced by combining the obtained X_i and Y_i as

$$Z_i = \begin{cases} X_i - Y_i, & \text{if} X_i \geq Y_i, \\ X_i - Y_i + 1, & \text{otherwise.} \end{cases}$$

Total period $\approx 2^{144} \approx 2.23 \times 10^{43}$. Very popular in high-statistics Monte Carlo simulations.

UPPSALA
UNIVERSITET

An information source

Pseudo Random Numbers: Generation and Quality Checks
Wolfhard Janke
Quantum Simulations of Complex Many-Body Systems:
From Theory to Algorithms, Lecture Notes, J. Grotendorst, D.
Marx, A. Muramatsu (Eds.),
John von Neumann Institute for Computing, Julich,
NIC Series, Vol. 10, ISBN 3-00-009057-6, pp. 447-458, 2002.

Matlab

`rand(1000)`	uniform values from the interval [0, 1]
`r = a + (b-a).*rand(100,1)`	uniform values from the interval [a, b]
`r = ceil(n.*rand(100,1))`	integers uniform on the set 1:n
`randn(50)`	normally distributed random numbers
`r = 1 + 2.*randn(100,1)`	normal values with mean 1 and sd 2

Matlab: Creating and controlling one or multiple Random Number Streams

The `RandStream` class allows you to create a random number stream.

```
myStream=RandStream('mlfg6331_64');
rand(myStream,1,5)

ans = 0.6530  0.8147  0.7167  0.8615  0.0764
```

Multiple streams:

```
[s1,s2,s3]=RandStream.create('mlfg6331_64','NumStreams',3)
r1=rand(s1,100000,1);
r2=rand(s2,100000,1);
r3=rand(s3,100000,1);
corrcoef([r1,r2,r3])
```

UPPSALA UNIVERSITET

Matlab: available generators

mt19937ar	Mersenne-Twister (default)	$2^{19937} -$
mcg16807	Multiplicative congruential generator	$2^{31} - 2$
mlfg6331_64	Multiplicative lagged Fibonacci generator	2^{124}
mrg32k3a	Combined multiple recursive generator	2^{127}
shr3cong	Shift-register generator summed with linear congruential generator	2^{64}
swb2712	Modified subtract with borrow generator	2^{1492}

Matlab: available generators

`?RNGkind()`

Mersenne-Twister	$2^{19937} - 1$
Wichmann-Hill	6.9536×10^{12}
Marsaglia-Multicarry	$> 2^{60}$
Super-Duper	$\approx 4.6 * 10^{18}$
Knuth-TAOCP-2002	2^{129}
user-supplied	

`rnorm(20)`	normal distrib.
`sample`	sample from a set
`rbinom(10, mean=7,sd=5)`	binomial distrib.
`runif(n, min=0, max=1)`	uniform distribution on $[min, max]$

UPPSALA UNIVERSITET

The Law of large numbers (LLN)

The LLN was first described by Jacob Bernoulli (1713).
Poisson, Chebyshev, Markov, Cantell, Kolmogorov.
If $X_1, X_2, \cdots, X_n$ are iid with finite expectation μ, then the sample average $\overline{X_n} = 1/n \sum_{k=1}^{n} X_k$ converges to μ.

$$\lim_{n\to\infty} P(|\overline{X_n} - \mu| < \varepsilon) = 1 \qquad P(\lim_{n\to\infty} \overline{X} = \mu) = 1$$

The LLN is important because it "guarantees" stable long-term results for random events.

UPPSALA UNIVERSITET

Buffon-Laplace-needle problem

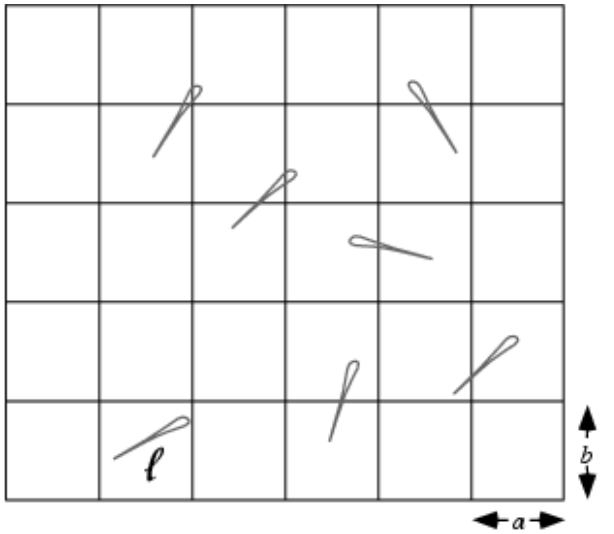
Problem Statement:
More than 200 years before Metropolis coined the name 'Monte Carlo' method, George Louis Leclerc, Comte de Buffon, proposed the following problem.
'If a needle of length ℓ is dropped at random on the middle of a horizontal surface ruled with parallel lines a distance $d > \ell$ apart, what is the probability that the needle will cross one of the lines?'
This problem was first solved by Buffon (1777, pp. 100-104), but his derivation contained an error.
A correct solution was given by Laplace (1812, pp. 359-362; Laplace 1820, pp. 365-369).

`http://mathworld.wolfram.com/Buffon-LaplaceNeedleProblem.html`

UPPSALA UNIVERSITET

Buffon-Laplace-needle problem

We reformulate somewhat the original problem in the following way. Imagine that a needle of length ℓ is dropped onto a floor with a grid of equally spaced parallel lines distances a and b apart, where ℓ is less than a and b.



The probability that the needle will land on at least one line is given by

$$P(\ell, a, b) = \frac{2\ell(a+b) - \ell^2}{\pi\, ab}$$

(Uspensky 1937, p. 256; Solomon 1978, p. 4).

Buffon-Laplace-needle problem

The idea: keep dropping this needle over and over on the table, and record the statistics. Namely, we want to keep track of both the total number of times that the needle is randomly dropped on the table (call this N), and the number of times that it crosses a line (call this C).

If you keep dropping the needle, eventually you will find that the number $\frac{N(2\ell(a+b)-\ell^2)}{Cab}$ approaches the value of π.
(Note: for large N the quantity C/N approaches the probability $P(\ell, a, b)$.)

UPPSALA
UNIVERSITET

In order to get a reasonably accurate approximation of π we need to perform a number of trials of order $10^6 - 10^8$. Since the separate trials are completely independent, we can perform those in parallel and sum up the result. This problem is an example of a trivial parallelism.

BUFFON_LAPLACE - Master process:

```
FORTRAN version
MPI program to estimate PI in the Buffon-Laplace needle experiment.
On a grid of cells of width A and height B, a needle of
length L is dropped. We count the number of times it crosses at
least one grid line, and use this to estimate the value of PI.
pi = 3.14159265358979323846264 3D+00

The number of processes is   4

Cell width A =  1.000000
Cell height B =  1.000000
Needle length L =  1.000000

Trials    Hits    Estimated PDF       Estimated Pi       Error

400000    382200   .955500000000      3.13971742543      .187522815808E
```

Buffon needle: Matlab, on Maya's laptop

Number of trials:	Error	Time
1000	0.053300	0.000365
1000000	0.000231	0.277653
100000000	0.000034	27.75966
1000000000	0.000017	277.2686