Simulation

Alberto Ceselli
MSc in Computer Science
Univ. of Milan

Part 2 - Generating Random Variables

Acknowlegment

Lecture adapted from D. Malchiodi's “Simulation Book” (CC BY 3.0 US)

https://github.com/dariomalchiodi/simulation-book/blob/master/lecture-2-pseudorandomness.ipynb

Random Numbers

Dictionary definition of *random*:

> *happening, done, or chosen by chance rather than according to a plan*

Random Numbers

Dictionary definition of *random*:

> *happening, done, or chosen by chance rather than according to a plan*

How to ask a computer to act *outside a plan (i.e. algorithm)*?

Random Generation

Example of random generation:

```
int getRandomNumber() {
return 6; // chosen by random dice roll
}
```

Pseudo-random generation

- The behavior of a computer is always the result of a program execution and thus it is **purely deterministic**;
- computers can be (deterministically) programmed in such a way that they exhibit **random behavior.**

That is, we formally distinguish

- *genuine* randomness (which we naturally observe in the world)
- *artificial* randomness, or **pseudorandomness**, (which we can simulate through computers)

The dream of random generation

- L. H. C. Tippet (Random Number Table, ~1927): 10,400 numbers of four digits taken at random from the British census reports,
- M. G. Kendall (machine producing tables of random digits, ~1938)
- Fisher and Yates Tables
- Kendall and Babington Smiths Tables
- Rand corporation random number tables

Pseudo-random number generators

- Von Neumann's middle square generator
- Congruential generator
- Bit shifter

Von Neumann's middle square generator

Algorithm:

- take a number (seed)
- compute its square
- keep the middle digits as the “random number”
- use it as the “seed” for the subsequent iteration.

See R code implementation

Von Neumann's middle square generator

Algorithm:

- take a number (seed)
- compute its square
- keep the middle digits as the “random number”
- use it as the “seed” for the subsequent iteration.

See R code implementation

Drawbacks?

Von Neumann's middle square generator

Algorithm:

- take a number (seed)
- compute its square
- keep the middle digits as the "random number"
- use it as the "seed" for the subsequent iteration.

See R code implementation

Drawbacks? After a few iterations no more "random"!

Congruential generator

Algorithm: choose three parameters a, c and m, and a seed s

$$x_0 = s; \;\; x_{i+1} = (a \cdot x_i + c) \bmod m;$$

See R code implementation

Congruential generator

Algorithm: choose three parameters a, c and m, and a seed s

$$x_0 = s; \;\; x_{i+1} = (a \cdot x_i + c) \text{ mod } m;$$

See R code implementation Drawbacks?

Congruential generator

Algorithm: choose three parameters a, c and m, and a seed s

$$x_0 = s; \;\; x_{i+1} = (a \cdot x_i + c) \text{ mod } m;$$

See R code implementation Drawbacks? The sequence tends to repeat!

Congruential generator

Key parameter: the *period* of the generator, m *in the best case!*.

- **Knuth 1981.** A mixed congruential generator has *full period* for all seed values if and only if:
  - m and c are relatively prime,
  - $a-1$ is divisible by all prime factors of m,
  - $a-1$ is divisible by 4 if m is divisible by 4.
- **Ripley, 1987.** A congruential generator has period $m-1$
  - only if m is prime
  - when $m-1$ is prime, the period is a divisor of $m-1$, and it is precisely $m-1$ when a is a primitive root of m ($a \neq 0$ and $a^{(m-1)p}$ not congruential to 1 modulo m for each prime factor p of $m-1$).
- **Park and Miller, 1988.** when m is the Mersenne's prime $2^{31}-1$, one of its primitive root is $a=7^5$, thus the recurrence relation $x_{i+1} = 7^5 x_i \mod 2^{31}-1$ will have a full period.

Congruential generator

Is having high period enough?

Congruential generator

Is having high period enough?

$$x_{i+1} = (1 \cdot x_i + 1) \bmod m$$

Congruential generator

Is having high period enough?

$$x_{i+1} = (1 \cdot x_i + 1) \text{ mod } m$$

Very predictable! I would like to *simulate to randomly draw from a uniform distribution, instead!*

How to check *predictability*? (See R code)

- Ripley's test
- Empirical cumulative distribution function (sample r):

$$ECDF(x) = \text{number of elements of } r \text{ having value } \leq x$$

Glivenko-Cantelli thm. if $\hat{F}$ has been computed using a sample of size n drawn from a distribution whose c.d.f. is F, $\hat{F}$ converges in probability to F as n increases.

Expected properties of generators

Expected properties of a random generator are:

Expected properties of generators

Expected properties of a random generator are:

- the set of generated pseudorandom values should be undistinguishable from an analogous sample drawn from a discrete uniform distribution over $\{0, \ldots, m-1\}$;
- its period should be as higher as possible;

Expected properties of generators

Expected properties of a random generator are:

- the set of generated pseudorandom values should be undistinguishable from an analogous sample drawn from a discrete uniform distribution over $\{0, \ldots, m-1\}$;
- its period should be as higher as possible;
- its computer implementation should be efficient (e.g. $m = 2^3 1 - 1$ allows to be encoded with 32 bits).