RIGA TECHNICAL UNIVERSITY

Monte Carlo Methods in Financial Engineering

2. Topic

Generation of random variables

Introduction

- Algorithms at the core of Monte Carlo simulations:
  - Generating uniformly distributed random variables;
  - Methods for transforming uniformly distributed variables into other distributions.
- To efficiently implement simulations, these algorithms can be repeated millions of times within a single simulation framework.

Introduction

- In the analysis of Monte Carlo methods, it is necessary to use sequences of random variables that exhibit an occasional nature or are random.
- This is convenient “invention” that allows the use of probability theory and statistical tools in the analysis of Monte Carlo calculations:
  - “Convenient”, because modern pseudo-random number generators are good enough at simulating true randomness, making this analysis informative;
  - Pseudo-random number generators use entirely deterministic algorithms.

Introduction

- True random number generator: a mechanism that creates a sequence of random variables $U_1, U_2, \ldots$, possessing following properties:
  1. each U_i is uniformly distributed between 0 and 1;
  2. U_i are mutually independent.
- The first property is the accepted normalization; but you could also take, for example, the interval between 0 and ½.
- Random variables that are uniformly distributed within the unit interval can be transformed into essentially any other distribution.

Efficient generator

- A random number generator creates a finite sequence of numbers $u_1, u_2, \ldots, u_K$ within the unit interval.
- Usually, these values depend on user-defined input parameters.
- Each such sequence forms a possible outcome set independent of the random variable sequence $U_1, U_2, \ldots, U_K$.
- An efficient generator produces values that possess both aforementioned properties (slide 4).

Efficient generator

- Period length
- Reproducibility
- Speed
- Portability
- Randomness

Linear Congruential generator

Linear congruential generator = LCG

$$x_{i+1} = (ax_i + c) mod\ m$$
$$u_{i+1} = x_i / m$$

Parameters a, c, m – integer numbers

- If $c = 0$, then for the generated sequence x_i:
  - $x_0 = 1, a = 6\ and\ m = 11$: 1, 6, 3, 7, 9, 10, 5, 8, 4, 2, 1, 6, …
    (all ten different integers are between 1 and $m - 1$);
  - $x_0 = 1, a = 3\ and\ m = 11$: 1, 3, 9, 5, 4, 1, …
  - $x_0 = 2, a = 3\ and\ m = 11$: 2, 6, 7, 10, 8, 2, …
  - All possible values {1, …, 10} are divided into two cycles.

Linear Congruential generator

$$x_{i+1} = (ax_i + c) mod\ m$$
$$u_{i+1} = x_i / m$$

- If $c \neq 0$, then the following conditions guarantee that the generator has a full period (the number of different values generated with any initial values x_0, is $m - 1$):
  1. c and m are relatively prime, meaning their only common divisor is 1;
  2. every prime number that divides m also divides $a - 1$;
  3. $a - 1$ is divisible by 4 if m is divisible by 4.

Linear Congruential generator

- Conclusion: If m is a power of 2, then the generator has a full period, if c is an odd number and $a = 4n + 1$ for any integer n.
- If $c = 0$ and m is a prime number, a full period is achieved starting from any $x_0 \neq 0$, if
  1. $a^{m-1} - 1$ is divisible by m without residual;
  2. $a^j - 1$ is not divisible by m without residual, where $j = 1, \ldots, m - 2$.

  A number a that satisfies both properties is called a *primitive root* of the number m.

Linear Congruential generator

If $c = 0$, then the sequence $\{x_i\}$ is as follows:

$$x_0, ax_0, a^2x_0, a^3x_0, \ldots (mod\ m)$$

- The sequence returns to the value x_0 for the first time at the smallest k, where $a^k x_0\ mod\ m = x_0$.
- This is the smallest k for which $a^k mod\ m = 1$ holds, that is, the smallest k for which $a^k - 1$ is divisible by m without a residual.
- So, the definition of a primitive root precisely corresponds to the requirement that the sequence does not return to the value x_0 until the value of the sequence is $a^{m-1}x_0$.
- In the case where a is a primitive root of m, all x_i are nonzero, if x_0 is nonzero. This is important because if any of the x_i were 0, then all subsequent generated values would also be zeros.

Linear Congruential generator

- Since generators with a nonzero c are slower than those using $c = 0$, generators with $c = 0$ are often preferred.

m	*a*
$2^{31} - 1 = 2\,147\,483\,647$	$16\,807$
	$39\,373$
	$742\,938\,285$
	$950\,706\,376$
	$1\,226\,874\,159$
$2\,147\,483\,399$	$40\,692$
$2\,147\,483\,563$	$40\,014$

LCG implementation

- If the multiplier a is large, even in double-precision number format, it may be insufficient to accurately represent each product ax_i.

⇒ The generator can be introduced by initially representing the multiplier as $a = 2^a a_1 + a_2$, where $a_1, a_2 < 2^a$, and then using

$$ax_i mod\, m = (a_1(2^a x_i \, mod\, m) + a_2 x_i \, mod\, m) mod\, m$$

For example, if $a = 16$ and $m = 2^{31} - 1$, in this approach, no value will be as large as 2^{47}, even if ax_i could be close to the number 2^{62}.

LCG implementation

- Sometimes it is useful to be able to divide the sequence of random numbers into mutually unrelated sub-sequences.

⇒ This can be done by generating the same random number with two or more different initial values.

By choosing initial values arbitrarily, there is a possibility that seemingly unrelated sub-sequences will have a significant overlap.

LCG implementation

- When using an LCG, it's easy to "jump" forward through the sequence without generating intermediate values.

If $x_{i+1} = ax_i \ mod \ m$, when

$$x_{i+k} = a^k x_i \ mod \ m$$

This, in turn, is equivalent

$$x_{i+k} = \left(\left(a^k \ mod \ m \right) x_i \right) mod \ m$$

Combined generators

- One way to move further from the basic LCG is to combine 2 or more such LCG generators, summing them up:
  - By summing the values from the unit interval, that is, after mod division;
  - By first summing and then dividing.
- The goal is to meet requirements related to speed, uniformity, and period length.

General methods of random sampling

- Let's assume that there exists an independent sequence of random variables $U_1, U_2, \ldots$, where

$$P(U_i \leq u) = \begin{cases} 0, u < 0 \\ u, 0 \leq u \leq 1 \\ 1, u > 1 \end{cases}$$

- The simulation algorithm transforms the independent uniformly distributed values in the interval [0; 1] into trajectories of a stochastic process.
  1. Inverse transform method;
  2. Acceptance-rejection method.

Inverse transform method

- It is necessary to generate a random variable X with properties

$P(X \leq x) = F(x), \forall x.$

- Inverse transform method:

$$X = F^{-1}(U), \qquad U \sim Unif[0,1]$$

F^{-1} - The inverse function of the function F

Inverse transform method

Example 1. The exponential distribution
(such a distribution applies to the time between jumps in a Poisson process)

The exponential distribution with mean θ has a cumulative distribution function:

$$F(x) = 1 - e^{-\frac{x}{\theta}}, \qquad x \geq 0$$

Taking the inverse function, we get the algorithm:

$$X = F^{-1}(U) = -\theta \ln(1 - U)$$

$$X = -\theta \log(U),$$

Where U and $1 - U$ follow the same distribution.

Inverse transform method

Example 2. *The Arcsine law*
(a distribution for the time at which standard Brownian motion reaches its maximum within the time Interval [0;1])

$$F(x) = \frac{2}{\pi}\arcsin(\sqrt{x}), \qquad 0 \le x \le 1$$

Inverse transform method: $X = sin^2(U\pi/2)$, $\mathrm{U} \sim Unif[0,1]$.

By using the identity $2sin^2(t) = 1 - \cos(2t), 0 \le t \le \pi/2$,

The transformation can be simplified as:

$$X = \frac{1}{2} - \frac{1}{2}\cos(U\pi), \qquad U \sim Unif[0,1]$$

Acceptance-Rejection Method

- It is necessary to generate samples with a function f defined in a set D (which could be a real line or $\Re^d$, or even more general sets).
- Choose g, which is a density function defined in set D, and for which it is known how to generate samples. It should also have the property that $\exists c, f(x) \leq cg(x), \forall x \in D$.
- Using the method, a sample X with density function g is generated, and it is accepted with a probability $f(X)/cg(X)$.
- This can be done by generating $U \sim Unif[0,1]$ and accepting X if $U \leq f(X)/cg(X)$ holds.

Acceptance-Rejection Method

1. Generate X with the density function g.
2. Generate U from $Unif[0,1]$.
3. If $U \leq f(X)/cg(X)$
   accept X
   Otherwise,
   go back to step 1.

Acceptance-Rejection Method

Example. *Normal from double exponential*

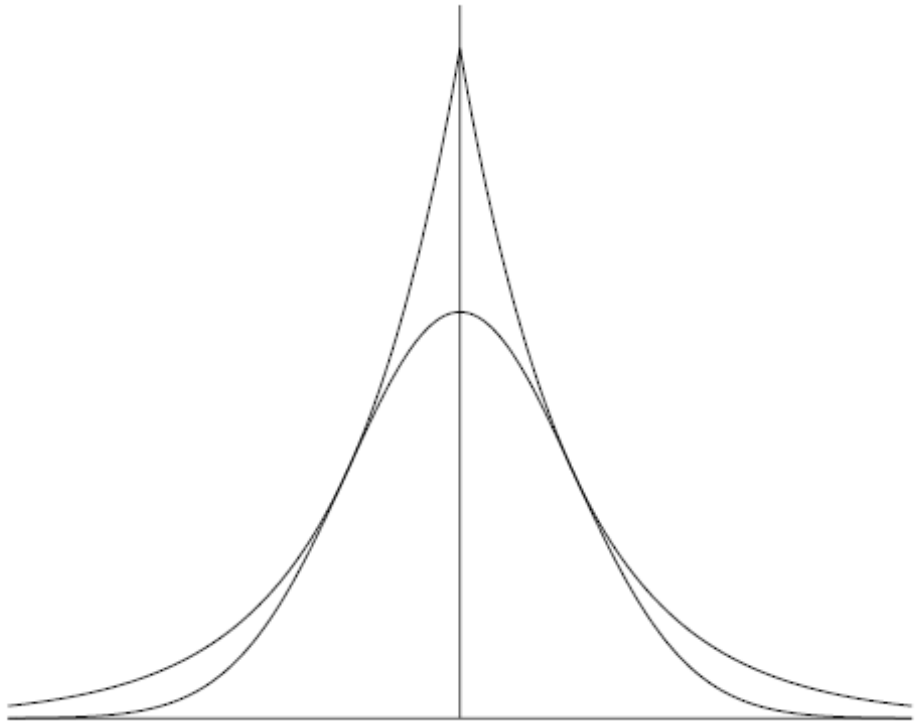
The density function of the double exponential distribution of $(-\infty, \infty)$ is $g(x) = e^{-|x|}/2$

The density function of normal distribution is $f(x) = e^{-\frac{x^2}{2}}/\sqrt{2\pi}$

The relationship is $\frac{f(x)}{g(x)} = \sqrt{\frac{2}{\pi}} e^{-\frac{1}{2}x^2+|x|} \leq \sqrt{\frac{2e}{\pi}} \approx 1.3155 \equiv c$

Thus, the weighted double exponential density $\mathrm{cg}(x)$ dominates the normal density (see the image on the next slide).

Acceptance-Rejection Method



The probability density function of the normal distribution and the weighted double exponential distribution.

Acceptance-Rejection Method

Example. *Normal from double exponential*

An ensemble with a double exponential density can be generated using $\mathrm{X} = -\theta\log(U)$ (inverse transform method) to obtain a standard exponential random variable and then assign the sign in a randomized manner.

The rejection test $u > f(X)/cg(X)$ can be implemented as

$$u > \exp\left(-\frac{1}{2}x^2 + |x| - \frac{1}{2}\right) = \exp\left(-\frac{1}{2}(|x| - 1)^2\right)$$

Acceptance-Rejection Method

Example. *Normal from double exponential*

1. Generate U_1, U_2, U_3 from $Unif\,[0,1]$
2. $X \leftarrow -\log(U_1)$
3. If $U_2 > \exp(-0.5(X-1)^2)$
   go to step 1
4. If $U_3 \leq 0.5$
   $X \leftarrow -X$.
5. Accept X

1 Task

Generate 500 random variables uniformly distributed in the interval [0,1] using a linear congruential generator.

2 Task

Using the inverse transformation method, obtain an exponential distribution from the uniform distribution, $\theta = 1$.

3 Task

Using the acceptance-rejection method, obtain a normal distribution from the double exponential distribution, $\theta = 1$.

Thank You!