Lecture 2: Introduction to Numerical Simulation

Ahmed Kebaier
kebaier@math.univ-paris13.fr

HEC, Paris

Outline of The Talk

1 Simulation of Random variables

Outline

1 Simulation of Random variables

Random Number Generators I

- Computer algorithms for generating random numbers are deterministic algorithms. Although the sequence of numbers produced by a random number generator appears random, the sequence of numbers is completely predictable and for this reason they are often called **pseudo-random**.

Random Number Generators I

- Computer algorithms for generating random numbers are deterministic algorithms. Although the sequence of numbers produced by a random number generator appears random, the sequence of numbers is completely predictable and for this reason they are often called **pseudo-random**.

- Since computers have only a finite number of different states, the sequence of pseudo-random numbers produced by any deterministic algorithm must necessarily repeat itself. The number of random numbers generated before the sequence repeats is called the period of the generator.

Random Number Generators II

A good random number generator should have the following characteristics:

1. **Randomness.** It should pass statistical tests of randomness.
2. **Long Period.** For obvious reasons.
3. **Efficiency.** This is important since simulations often require millions of random variables
4. **Repeatability.** It should produce the same sequence of numbers if started in the same state. This allows the repetition and checking of simulations.

Random Number Generators II

A good random number generator should have the following characteristics:

1. **Randomness.** It should pass statistical tests of randomness.
2. **Long Period.** For obvious reasons.
3. **Efficiency.** This is important since simulations often require millions of random variables
4. **Repeatability.** It should produce the same sequence of numbers if started in the same state. This allows the repetition and checking of simulations.

Almost all random number generators used in practice produce uniform [0, 1] distributed random numbers and from these random numbers with other distributions can be produced if required.

Algorithms

Linear Congruential Generators The simplest are the linear congruential generators. Starting with the seed x_0, these generate a sequence of integers by

$$x_k = (ax_{k-1} + c) \, mod \, M.$$

where a, c and M are given integers. All the x_k are integers between 0 and $M - 1$. In order to produce floating point numbers these are divided by M to give a floating point number in the interval [0, 1).

Algorithms

Linear Congruential Generators The simplest are the linear congruential generators. Starting with the seed x_0, these generate a sequence of integers by

$$x_k = (ax_{k-1} + c) \, mod \, M.$$

where a, c and M are given integers. All the x_k are integers between 0 and $M - 1$. In order to produce floating point numbers these are divided by M to give a floating point number in the interval [0, 1).

```
function [x] = lcg(n, a, c, m, x0)
x = zeros(1,n+1)
x(1) = x0
for i = 2:n+1
x(i) = pmodulo(a*x(i-1)+c, m)
end
x = x/m
endfunction
```

- The following example illustrates a common problem with some random number generators. We will take $a = 1203$, $c = 0$, $m = 2048$. With initial value $x_0 = 1$, this generator has period 512. We will run through a full period of the generator:

```
-->xx = lcg(511, 1203, 0, 2048, 1);
-->xx(1:10)
```

- The following example illustrates a common problem with some random number generators. We will take $a = 1203$, $c = 0$, $m = 2048$. With initial value $x_0 = 1$, this generator has period 512. We will run through a full period of the generator:

```
-->xx = lcg(511, 1203, 0, 2048, 1);
-->xx(1:10)
```

- Now take successive pairs of values as the x and y coordinates of a point in the plane and plot the results.

```
-->x = xx(1:2:511);
-->y = xx(2:2:512);
-->plot2d(x,y,style = -1)
```

- The following example illustrates a common problem with some random number generators. We will take $a = 1203$, $c = 0$, $m = 2048$. With initial value $x_0 = 1$, this generator has period 512. We will run through a full period of the generator:

```
-->xx = lcg(511, 1203, 0, 2048, 1);
-->xx(1:10)
```

- Now take successive pairs of values as the x and y coordinates of a point in the plane and plot the results.

```
-->x = xx(1:2:511);
-->y = xx(2:2:512);
-->plot2d(x,y,style = -1)
```

- This "latticing" is a common problem with random number generators. the points may tend to lie on a lattice of lower dimensional subspaces.

Other Random Number Generators

- random number generators are based on number theory and have quite sophisticated implementations. One in common use is the Mersenne twister which takes a vector of 625 as its seed.

- Scilab has two random number generators **rand** and **grand**. Uniform random numbers are the default. We have already seen how **rand** works:
  - `rand(m, n)` - gives a $m \times n$ matrix of random numbers.
  - `rand(m, n, 'normal')` - gives a $m \times n$ matrix of normally distributed random numbers.
  - `rand(a)` or `rand(a, 'normal')` - for matrix a gives a matrix of ran- dom numbers the same size as a.
  - `rand('seed', 0)` - resets the random number generator to its original state. This is handy if you want to repeat an experiment using the same random numbers.

Some Common Distributions

Uniform distribution

- If x is a uniform [0, 1] distribution, then the change of variable

$$y = (b - a)x + a$$

- Here is an example generating a uniform [-5, 5] distribution:

```
-->x =rand(1,10000);
-->y = 10*x - 5;
-->histplot(100,y)
```

Some Common Distributions

Normal distribution

- If x has a normal $\mu = 0$, $\sigma = 1$ distribution, then the change of variable

$$y = \mu + \sigma x$$

- Here is an example generating a normal $\mu = 100$, $\sigma = 10$ distribution:

```
-->x = rand(1,10000,'normal');
-->y = 10*x + 100;
-->histplot(100,y)
```

Non-Uniform Random Numbers

Theorem 1

Let Y be a random variable with distribution function F_y. We define for all $x \in [0,1]$

$$F_Y^{-1}(y) := \inf\{x \in \mathbb{R} : F_Y(x) \geq y\}.$$

If $U \sim U_{[0,1]}$ then $F_Y^{-1}(U)$ has the same distribution as Y.

Remark:

1. Note that by construction we always have

$$\forall y \in \mathbb{R} \quad y \leq F_Y\left(F_Y^{-1}(y)\right).$$

2. If in addition F is continuous then F^{-1} coincides with the classical inverse function and we get

$$\forall y \in \mathbb{R} \quad y = F_Y\left(F_Y^{-1}(y)\right).$$

Exercise

Simulation of the Exponential Distribution

- Let Y be random variable with Exponential distribution $\mathcal{E}(\lambda)$, $\lambda > 0$. That is Y have a density f

$$f(x) = \begin{cases} 0 & x < 0 \\ \lambda e^{-\lambda x} & x \geq 0 \end{cases}$$

1. Compute the associated distribution function F and its inverse G.
2. For $\lambda = 2$, simulate a sample of size $N = 10000$ of random variables with distribution $\mathcal{E}(2)$

Solution

```
-->y = rand(1,10000);
-->x = - log(1-y)/2;
-->histplot(100,x)
```

- We can compare the histogram to exact density

```
-->xx = 0:0.01:5;
-->plot2d(xx, 2*exp(-2*xx))
```

Solution

```
-->y = rand(1,10000);
-->x = - log(1-y)/2;
-->histplot(100,x)
```

- We can compare the histogram to exact density

```
-->xx = 0:0.01:5;
-->plot2d(xx, 2*exp(-2*xx))
```

Remark: One drawback of the inverse transform method is that we need to know the inverse of the distribution function. For most distributions there is no explicit formula for this inverse and we must resort to approximations or seek another method.

Simulation of a discrete distributions

- Let Y be a random varibale taking values in $\{y_k,\ k \in \mathbb{N}\}$ and such that

$$\mathbb{P}(Y = y_k) = p_k.$$

If U is a uniform random variable on $[0, 1]$, then the random variable X given by

$$X := y_0 \mathbf{1}_{\{U \leq p_0\}} + \sum_{k=1} y_k \mathbf{1}_{\{\sum_{i=0}^{k-1} p_i < U \leq \sum_{i=0}^{k} p_i\}}$$

has the same distribution as Y.

Simulation of a discrete distributions

- Let Y be a random varibale taking values in $\{y_k,\ k \in \mathbb{N}\}$ and such that

$$\mathbb{P}(Y = y_k) = p_k.$$

If U is a uniform random variable on $[0, 1]$, then the random variable X given by

$$X := y_0 \mathbf{1}_{\{U \leq p_0\}} + \sum_{k=1} y_k \mathbf{1}_{\{\sum_{i=0}^{k-1} p_i < U \leq \sum_{i=0}^{k} p_i\}}$$

has the same distribution as Y.

Exercise

Generate uniformly distributed random integers in the range 1 to 10 (inclusive).

Simulation of a discrete distributions

- Let Y be a random varibale taking values in $\{y_k,\ k \in \mathbb{N}\}$ and such that

$$\mathbb{P}(Y = y_k) = p_k.$$

If U is a uniform random variable on $[0, 1]$, then the random variable X given by

$$X := y_0 \mathbf{1}_{\{U \leq p_0\}} + \sum_{k=1} y_k \mathbf{1}_{\{\sum_{i=0}^{k-1} p_i < U \leq \sum_{i=0}^{k} p_i\}}$$

has the same distribution as Y.

Exercise

Generate uniformly distributed random integers in the range 1 to 10 (inclusive).

Solution

```
-->x = rand(1,30);
-->y = floor(10*x+1)
```

Exercise

Simulate a Binomial distribution with parameter N and P. ($N = 5$ and $p = 0.25$).

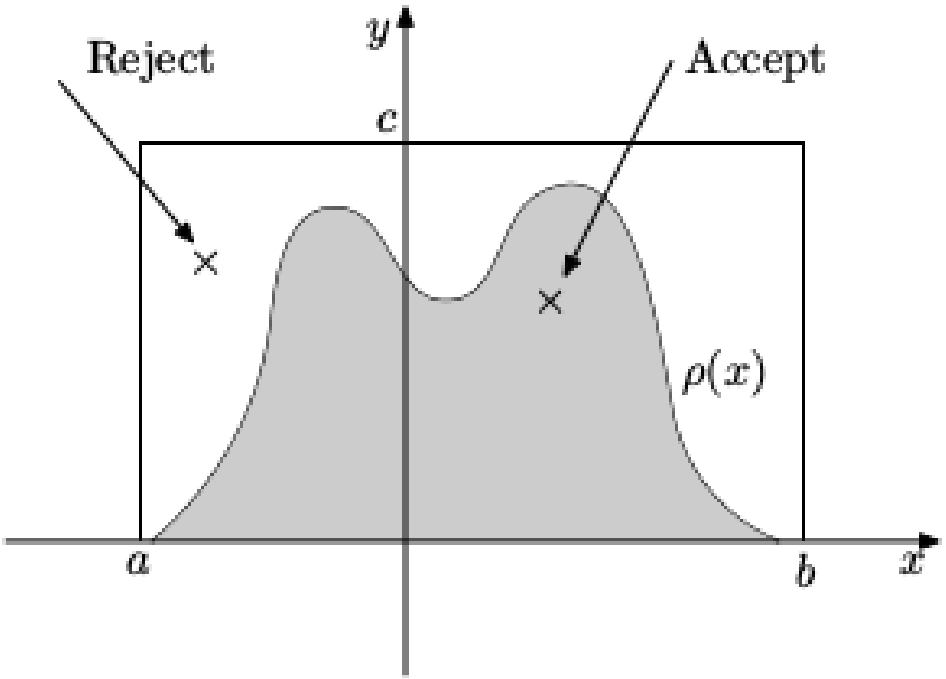
Exercise

Simulate a Binomial distribution with parameter N and P. ($N = 5$ and $p = 0.25$). **Solution**

```
function [x]=s_binomial(s,N,p)
// simulation_binomial
// s = Sample size
// N,p = parameters
//-------------------------------
x=0*ones(1:s);
y=rand(s,N);
for i=1:s do ;
for j=1:N do ;
if y(i,j)< p then x(i)=x(i)+1;
elseif y(i,j) >= p then x(i)=x(i);
end;
end;
end;
enfunction  -->exec('s_binomial.sci',-1)
-->x=s_binomial(1000,5,0.25)
```

Acceptance/Rejection Method

- Suppose that $\rho(x)$ is zero outside of the interval $[a, b]$ and furthermore that $\rho(x)$ is bounded above by c.

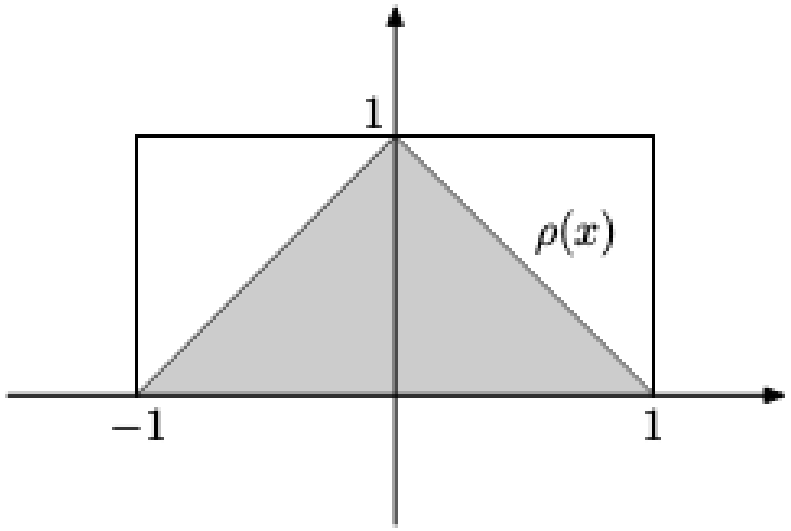


- Now generate points (x_i, y_i) with x_i uniformly distributed in $[a, b]$ and y_i uniformly distributed in $[0, c]$.
- If $y_i \leq \rho(x_i)$ then we accept the value x_i, if $y_i \geq \rho(x_i)$ then we reject the value x_i. The values x_i which are accepted will have the probability density $\rho(x)$.

Exercise

- Consider the hat-shaped probability density defined on $[-1, 1]$ by

$$\rho(x) = \begin{cases} x + 1 & -1 \leq x \leq 0 \\ 1 - x & 0 \leq x \leq 1 \end{cases}$$

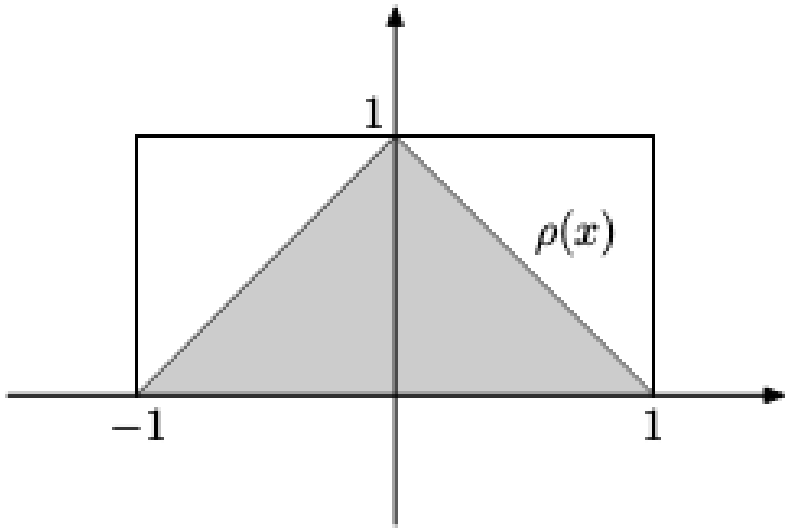


- Write a Scilab function to generate n random numbers with this density using the acceptance/rejection method.

Exercise

- Consider the hat-shaped probability density defined on $[-1, 1]$ by

$$\rho(x) = \begin{cases} x + 1 & -1 \leq x \leq 0 \\ 1 - x & 0 \leq x \leq 1 \end{cases}$$



- Write a Scilab function to generate n random numbers with this density using the acceptance/rejection method.

Remark. One thing to keep in mind when using the acceptance/rejection is that we do not know before hand how many random numbers we need to generate.

Solution

```
function [y] = rhohat(x)// First we need the density
function
if (x < 0) then
y = x + 1
else
y = 1 - x end
endfunction
```

Solution

```
function [y] = rhohat(x)// First we need the density
function
if (x < 0) then
y = x + 1
else
y = 1 - x end
endfunction
function [x] = randhat(n)// Now the random number
generator
x = zeros(1,n)
k = 0 // keep count of numbers generated
while (k < n)
xx = -1 + 2*rand(1,1) // uniform on [-1,1]
yy = rand(1,1)
```

Solution

```
function [y] = rhohat(x)// First we need the density
function
if (x < 0) then
y = x + 1
else
y = 1 - x end
endfunction
function [x] = randhat(n)// Now the random number
generator
x = zeros(1,n)
k = 0 // keep count of numbers generated
while (k < n)
xx = -1 + 2*rand(1,1) // uniform on [-1,1]
yy = rand(1,1)
if (yy <= rhohat(xx))// accept xx
k = k + 1
x(k) = xx
end
end
endfunction
```