Random Number Generation

732A90
Computational Statistics

Maryna Prus
(maryna.prus@liu.se)

Slides originally by Krzysztof Bartoszek

November 15, 2021

Department of Computer and Information Science
Linköping University

Pseudorandom Numbers

- Computer is deterministic machine

  $\rightarrow$ Creates (pseudo) random numbers using mathematical algorithms

  $\rightarrow$ The numbers are not truly random

- Classical approach: *linear congruential generator*

- (Pseudo) random numbers - realizations of some distribution

First Step: Generating Unif$[0, 1]$

Linear congruential generator

Define a sequence of integers according to

$$x_{k+1} = (a \cdot x_k + c) \mod m, \quad k \geq 0$$

- $\mod m$ - remainder after division by m
- x_0 - *seed* or *start value*, integer
- $a, c \in [0, m)$, integer
- $x_k \in \{0, \ldots, m-1\}$, integer
- x_k/m from Unif$[0, 1]$

First Step: Generating Unif$[0, 1]$

Generated numbers will get into a "loop" with a certain *period*

$$x_{k+1} = (a \cdot x_k + c) \mod m, \quad k \geq 0$$

Example: $x_0 = a = c = 7$, $m = 10$

1. $x_1 = (7 \cdot 7 + 7) \mod 10 = 56 \mod 10 = 6$
2. $x_2 = (7 \cdot 6 + 7) \mod 10 = 49 \mod 10 = 9$
3. $x_3 = (7 \cdot 9 + 7) \mod 10 = 70 \mod 10 = 0$
4. $x_4 = (7 \cdot 0 + 7) \mod 10 = 7 \mod 10 = 7$
5. $x_5 = (7 \cdot 7 + 7) \mod 10 = 56 \mod 10 = 6$
6. $\ldots$

$\rightarrow$ period is 4

First Step: Generating Unif$[0, 1]$

- Period is $\leq m$
- a, c, m have to be chosen carefully
- m typically very large
- Seed (x_0) defines the random sequence:

  same seed $\rightarrow$ same sequence
- In R: function `runif(n)`

  n - number of values to be generated
- Other methods - not in this course

Second Step: Generating Unif$[a, b]$ and Discrete Uniform Distribution

- $U \sim \text{Unif}[0, 1]$ can be transformed into $X \sim \text{Unif}[a, b]$:

$$X = a + U \cdot (b - a)$$

  In R: function `runif(n,a,b)`

- *discrete* uniform distribution on $\{1, \ldots, n\}$:

$$x = [nu] + 1,$$

  where u from Unif$[0, 1]$, $[\cdot]$ - integer part

  Why we add 1?

Inverse CDF Method

$U \sim \text{Unif}(0, 1)$

- *Cumulative distribution function* (CDF) of U:

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

- *Density function* of U:

$$f_U(u) = \begin{cases} 1, & 0 < u < 1 \\ 0, & otherwise \end{cases}$$

Inverse CDF Method

- X - random variable with CDF F_X
- F_X^{-1} - inverse of F_X
- $U \sim \text{Unif}(0, 1)$
- Consider $Y = F_X^{-1}(U)$:

$$
\begin{aligned}
F_Y(y) &= P(Y \leq y) \\
&= P(F_X^{-1}(U) \leq y) \\
&= P(F_X(F_X^{-1}(U)) \leq F_X(y)) \\
&= P(U \leq F_X(y)) \\
&= F_U(F_X(y)) = F_X(y)
\end{aligned}
$$

as $0 \leq F_X(y) \leq 1$ and $F_U(u) = u$ for $0 \leq u \leq 1$

$\rightarrow Y$ has same probability distribution as X

Inverse CDF Method

- X - random variable with CDF F_X
- $U \sim \text{Unif}(0,1)$
- We can generate u (realization of U)

  $\rightarrow$ we can generate x (realization of X) as

  $$x = F_X^{-1}(u)$$

  if F_X^{-1} exists

Inverse CDF Method

Example:

- $X \sim \exp(\lambda)$
- Density function of X:

$$f_X(x) = \begin{cases} \lambda e^{-\lambda x}, & x \geq 0 \\ 0, & otherwise \end{cases}$$

- CDF of X:

$$\begin{aligned} F_X(x) &= \int_{-\infty}^{x} f_X(s) \mathrm{d}s \\ &= \begin{cases} 1 - e^{-\lambda x}, & x \geq 0 \\ 0, & otherwise \end{cases} \end{aligned}$$

Inverse CDF Method

Example (cont.):

To determine F_X^{-1} solve the equation

$$y = 1 - e^{-\lambda x}$$

$$\begin{aligned}
\Rightarrow \quad & e^{-\lambda x} = 1 - y \\
\Rightarrow \quad & x = -\frac{1}{\lambda}\ln(1-y) \\
\Rightarrow \quad & F_X^{-1}(y) = -\frac{1}{\lambda}\ln(1-y)
\end{aligned}$$

Then for $U \sim \mathrm{Unif}(0,1)$

$$X = -\frac{1}{\lambda}\ln(1-U) \sim \exp(\lambda)$$

Inverse CDF Method

1. If F_X^{-1} can be derived

   $\rightarrow$ works well

2. If not

   $\rightarrow$ numerical solutions

   $\rightarrow$ time

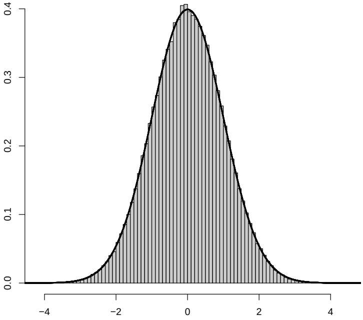
   $\rightarrow$ numerical errors

   Example: normal distribution

Generating Normal Distribution

N(0, 1)

- $\theta \in \mathrm{Unif}(0, 2\pi)$
- $D \in \mathrm{Unif}(0, 1)$



1: Generate θ, D
2: Generate X_1 and X_2 as

$$X_1 = \sqrt{-2\ln D}\cos\theta$$
$$X_2 = \sqrt{-2\ln D}\sin\theta$$

X_1 and X_2 are independent and normally distributed

In R: function `rnorm(n)`

Generating Normal Distribution

- $X \sim \mathrm{N}(0,1)$ can be transformed into $Y \sim \mathrm{N}(\mu,\sigma^2)$:

$$Y = \mu + \sigma X$$

In R: function `rnorm(n,`μ`,`σ`)`

- Multivariate normal distribution: $\vec{Y} \sim \mathrm{N}(\vec{\mu}, \mathbf{\Sigma})$

1: Generate $X_1, \ldots, X_n$, $X_i \sim \mathrm{N}(0,1)$

$\rightarrow \vec{X} = (X_1, \ldots, X_n)$

2: Compute $\mathbf{A}$ with $\mathbf{A}\mathbf{A}^\top = \mathbf{\Sigma}$

$\rightarrow$ Use Cholesky decomposition of $\mathbf{\Sigma}$

$\rightarrow$ in `R`: fucntion `chol()`

3: $\vec{Y} = \vec{\mu} + \mathbf{A}\vec{X}$

Acceptance/Rejection Method

- X - random variable with density function f_X
- f_X similar to f_Y - density of some *known* distribution
- Generate X using f_Y
- Requirement: there exists constant c with

$$cf_Y(x) \geq f_X(x),\ for\, all\, x$$

- f_Y - majorizing density, proposal density
- f_X - target density
- c - majorizing constant

Acceptance/Rejection Method

```
while X not generated do
    Generate Y from distribution with density f_Y
    Generate U from Unif(0,1)
    if U ≤ f_X(Y)/(c f_Y(Y)) then
        X = Y
        Set X is generated
    end if
end while
```

- X is *"really"* from distribution with density f_X
- Larger c leads to larger rejection rate

  $\rightarrow$ c should be as small as possible
- Problems:
  - Difficult to find majorizing density
  - High rejection rate

Example: Generate beta(2,7)

```
y<-dbeta(seq(0,2,0.0001),2,7)
c<-max(y);c
[1] 3.172554
```

1: **while** X not generated **do**
2: Generate $Y \sim \text{Unif}(0,1)$
3: Generate $U \sim \text{Unif}(0,1)$
4: **if** $U \le dbeta(Y,2,7)/(c \cdot 1)$ **then**
5: $X = Y$
6: Set X is generated
7: **end if**
8: **end while**

Random Numbers in R

1. `d`distribution name`()`: density of distribution
2. `p`distribution name`()`: CDF of distribution
3. `q`distribution name`()`: quantiles of distribution
4. `r`distribution name`()`: simulate from distribution

See also

- `?RNGversion`
- `?RNGkind`
- `https://bugs.r-project.org/bugzilla/show_bug.cgi?id=17494`

Thank you for attention!