Mathematical Tools

S. Robin, C. Doursat
INA-PG, Biometry

June 6, 2003

Contents

1 Simulating random numbers 3
  1.1 System-supplied random number generators ............... 3
  1.2 Initialization ........................................ 3
  1.3 Algorithm aspects .................................. 3
  1.4 Pseudo-random sequence .............................. 4
  1.5 Examples .......................................... 4
  1.6 Improvements ...................................... 4

2 Simulating classical distributions 6
  2.1 Discrete distributions ................................ 6
    2.1.1 Coin tossing .................................... 6
    2.1.2 Bernoulli distribution ............................ 6
    2.1.3 General discrete finite distribution ............ 7
    2.1.4 Binomial distribution ............................ 9
    2.1.5 Poisson distribution ............................. 10
  2.2 Continuous distributions ............................... 10
    2.2.1 Uniform distribution ......................... 10
    2.2.2 Inversion ...................................... 11
    2.2.3 Rejection method ............................... 13
    2.2.4 General rejection .............................. 14

3 The normal distribution 16
  3.1 Univariate ......................................... 16
    3.1.1 Central Limit Theorem (CLT) .................... 16
    3.1.2 Inversion method ............................... 17
    3.1.3 Rejection method ............................... 17
    3.1.4 Box-Müller .................................... 18
    3.1.5 Comparison of the methods .................... 19
  3.2 Multivariate case .................................. 19
3.2.1 Definition ......................................................... 19
3.2.2 Standard multivariate normal distribution .................. 20
3.2.3 General multivariate normal distribution ................... 20
3.2.4 Application to the bivariate case ............................. 21

A Appendix ................................................................. 23
A.1 Simulating normal variables .................................... 23
  A.1.1 Rejection method ............................................ 23
  A.1.2 Box and Müller method .................................... 23
A.2 MatLab functions for some classical distributions .......... 25
  A.2.1 General discrete distribution .............................. 25
  A.2.2 Poisson: inversion method .................................. 25
  A.2.3 Poisson: sum of exponential variables .................... 25
  A.2.4 Exponential: inversion method ............................ 26
  A.2.5 Rejection method ............................................. 26
  A.2.6 General rejection method: application to the normal distribution . 26
A.3 Cholesky factorization ........................................... 27

List of Figures

1 Simulating a Bernoulli variable .................................. 7
2 Simulating a discrete variable .................................... 8
3 Cumulative distribution function a discrete variable ........... 8
4 Inversion method ................................................... 11
5 Rejection method ................................................... 13
6 Application of the generalized rejection method to the normal distribution 18
1 Simulating random numbers

The result of any program on a computer is entirely predictable. So it seems to be strange to use a deterministic machine to produce “random” numbers. In fact with all languages we can find random number generators called “ran”, “ran1”, “rand” (Matlab), ... The computer doesn’t use a coin tossing to generate these numbers. We must understand that real random process comes from physic, biology, ... In fact the computer, to simulate a random variable, generates a sequence of numbers termed *pseudo-random*.

Uniform deviates are random numbers that lie within a specified range (such as [0, 1]). Other sorts of deviates are generally generated with operations on one or more uniform deviates. We study here only the generation of uniform deviates.

1.1 System-supplied random number generators

In each language, like on computers, you can find “rand” function which is almost always linear congruential generator. It generates a sequence of integers $i_1, i_2, i_3, ...$, each between 0 and $m - 1$ (where $m$ is a “large” number) using the recurrence relation:

$$i_{n+1} = a \cdot i_n + b \mod m$$

where $[m]$ means *modulo* $m$. Here $m$, $a$ and $b$ are positive integers and respectively called the modulus, the multiplier and the increment.

Each call of the function “rand” calculates the next term of the sequence.

1.2 Initialization

To define the sequence it is necessary to define the first term. Generally when beginning the execution of a program (or when starting Matlab) the computer resets the state of the uniform generator so it defines the same first term each time; thus, one should always obtain the same random sequence.

Generally, to define a new random sequence one uses the clock of the computer to initialize the sequence. For example, with Matlab:

```matlab
rand('state', sum(100 * clock))
```

resets it to a different initial state each time.

To resume, when we want identical pseudo-random sequences, we have to initialize the random-generator at the same state each time. On the opposite, when we want different sequences, it is necessary to initialize them with different states (so with a random generator!).

1.3 Algorithm aspects

Generally, random generators work with integers which are stored in memory on 4 octets (32 bits). If these integers are positive, the largest integer will be $2^{32} - 1$ and the integer
$2^{32}$ is, for the computer, equal to 0. If we take $m = 2^{32}$, the addition of two 32-bit integers is naturally computed modulo $2^{32}$, because of the storage of the numbers. In fact, the 32 bits include a sign bit and the multiplication of two 32-bit integers needs to use intermediates larger than 32 bits. So special algorithms (such as Schrage, Press & al. [8]) are generally used to compute the recurrence relation.

An advantage of the linear congruential method is that it uses few operations. So these generators are very fast, which is a very important aspect when the “rand” function is often called and can be the most consuming time operations in your simulations.

The last aspect is that most of the languages, such as Matlab, return a random real between 0 and 1. In that case the result of “rand” is simply divided by $m$ (in general) and is strictly less than 1, but occasionally exactly equal to zero.

### 1.4 Pseudo-random sequence

All the terms of the sequence are between 0 and $m - 1$. The order of the recurrence being 1, it is obvious that the sequence is always periodic, and the period not greater than $m$. If $m$, $a$ and $b$ are properly chosen, the period can be $m$, and in that case all possible integers between 0 and $m - 1$ will appear once. So, all the sequences are equivalent to only one periodic sequence. It is fundamental to have an estimation of the period of the random generator used before making a simulation.

One of the disadvantages of the linear congruential method is that often the low-order bits are much less random than their high-order bits (think about what happens if $a$ is even). Another disadvantage is to be not free of sequential correlation on successive calls.

### 1.5 Examples

We denote the linear congruential generator by $LGC(m, a, b, i_1)$. You can find here some examples of the LGC definition of some softwares, languages, etc. :

- **ANSIC** (language): $LGC(2^{31}, 1103515245, 12345, 12345)$
- **APPLE** (computer): $LGC(2^{35}, 5^{13}, 0, 1)$
- **CRAY** (computer): $LGC(2^{59}, 13^{13}, 0, 123456789(2^{32} + 1))$
- **MAPLE** (software): $LGC(2^{25}, 5^{13}, 0, 1)$
- **NAG** (Fortran library): $LGC(10^{12} - 11, 427419669081, 0, 1)$

### 1.6 Improvements

Several methods exist to improve LGC, but only two ideas (Press & al. [8]) are given here:
• A random deviate derived from the \( n \)-th value in the sequence, \( i_n \), is output not on the \( n \)-th call, but on a randomized later call (Park & Miller [9]). In that case it is necessary to store some values of the sequence during the simulation.

• When longer random sequences are needed, a good way is to combine two sequences with different periods. The new period obtained is the least common multiple (L’Ecuyer [6]).
2 Simulating classical distributions

In the following section, we shall consider the problem of simulating a random variable $X$ with a given distribution. A general review of this problem can be found in Devroye [4]; Evan & al. [5] give the main properties of many probabilistic distributions, including simulation formulae.

Most of the methods that we are to present use the definition of the cumulative distribution function (cdf) of the uniform distribution $U [0, 1]$

$$\forall u \in [0, 1], \quad \Pr \{U \leq u\} = u. \quad (1)$$

In the following we shall assume that we are able to simulate a set $\{U_i\}_{i \geq 1}$ of independent variables uniformly distributed on $[0, 1]$ (see section 1).

MatLab functions corresponding to some of the methods presented are given in appendix A.2.

2.1 Discrete distributions

2.1.1 Coin tossing

The simplest random value corresponds to the result $X$ of a coin tossing. $X$ is equal to one with probability $\frac{1}{2}$ and to zero with the same probability. Using equation (1), we have $\Pr \{X = 1\} = \Pr \{U \leq \frac{1}{2}\}$, so $X$ can be simulated as follows:

$$X = \begin{cases} 1 & \text{if } U \leq \frac{1}{2}, \\ 0 & \text{otherwise.} \end{cases} \quad (2)$$

2.1.2 Bernoulli distribution

The Bernoulli $B(p)$ distribution generalizes the coin tossing. A Bernoulli variable $X$ is equal to one with probability $p$ and to zero with probability $1 - p$:

$$\Pr \{X = 1\} = p, \quad \Pr \{X = 0\} = 1 - p.$$

The formula (2) can easily be generalized to any value of $p$:

$$X = \begin{cases} 1 & \text{if } U \leq p, \\ 0 & \text{otherwise.} \end{cases} \quad (3)$$

Figure 1 gives a geometrical representation of this formula.

**Simple random sampling.** This method can be applied to draw a random sample of $n$ individuals from a population of size $N$. The simple random sampling assumes that each of the $\binom{N}{n}$ possible samples\(^1\) have the same probability $\binom{N}{n}^{-1}$ to occur. A classical

\(^1\binom{n}{p}\) denotes the binomial coefficient: $\binom{n}{p} = n! [p! \times (n - p)!]^{-1}$. 

6
sampling technique consists in selecting each individual \( i = 1, \ldots, N \) with an adaptive probability:

\[
\Pr \{ \text{select } \#1 \} = \frac{n}{N};
\]

\[
\Pr \{ \text{select } \#2 \} = \begin{cases} \frac{n - 1}{N - 1} & \text{if } \#1 \text{ has been selected,} \\ \frac{n}{N - 1} & \text{otherwise;} \end{cases}
\]

\[
\Pr \{ \text{select } \#i \} = \frac{n - k(i)}{N - i + 1} \quad \text{where } k(i) \text{ is the number of individuals already selected.}
\]

Each selection can be made using formula (3).

### 2.1.3 General discrete finite distribution

In the general case of a random variable \( X \) belonging to a set of values \( \{1, 2, \ldots, k\} \) with respective probabilities \( \{p_1, p_2, \ldots, p_k\} \):

\[
p_x = \Pr \{ X = x \},
\]

the formula defined for the Bernoulli variable can be generalized using the cumulated distribution function \( F(x) \):

\[
F(x) = \Pr \{ X \leq x \} = \sum_{i=1}^{x} p_i = \Pr \{ X \leq x \}
\]

which are represented in figure 2. The formula is

\[
\begin{array}{l}
X = 1 \quad \text{if } U \leq F(1), \\
    = 2 \quad \text{if } F(1) < U \leq F(2), \\
    \vdots \\
    = x \quad \text{if } F(x - 1) < U \leq F(x), \\
    \vdots
\end{array}
\]

(4)
The corresponding MatLab function is given in appendix A.2.1, page 25.

As an introduction to the inverse function method (see section 2.2.2), this method can be explained considering figure 3: the random variable $U$ is drawn on the vertical axis, so it simulates a value of $F(X)$. If it belongs to the interval $[F(x - 1); F(x)]$, then $X$ takes the value $x$.

Uniform distribution on $\{1, \ldots, k\}$. Formula (4) can be applied to a variable $X$ uniformly distributed on $\{1, \ldots, k\}: p_x = \frac{1}{k}$. It is equivalent to the following formula:

$$X = \lfloor kU \rfloor$$
where \([a]\) denotes smallest integer superior or equal to \(a\) : \([a]\) \(- 1 < a \leq [a]\). This is obvious since

\[
\Pr \{X = x\} = \Pr \{[kU] = x\} = \Pr \{x - 1 < kU \leq x\} = \Pr \left\{ \frac{x - 1}{k} < U \leq \frac{x}{k} \right\} = \frac{1}{k}.
\]

**Markov chain.** The Markov processes are widely used in genetics because the Markov property is satisfied by genetic processes; the celebrated Fisher-Wright model is one of their historical applications.

The method described in this section can be easily applied to simulate such processes: consider a \(s\)-states Markov chain \(\{X(t)\}_{t \geq 0} \) with transition matrix

\[
\Pi = [\pi(i, j)]_{i,j=1,...s}
\]

where \(\pi(i, j)\) denotes the one-step transition probability from state \(i\) to state \(j\):

\[
\pi(i, j) = \Pr \{X(t + 1) = j \mid X(t) = i\}.
\]

If, at time \(t\), the process is in state \(X(t) = i\), the state \(X(t + 1)\) is distributed according to the \(i\)-th row of the matrix \(\Pi\). Thus \(X(t + 1)\) can be simulated using formula (4) with the probabilities

\[
p_x = \pi(i, x).
\]

**2.1.4 Binomial distribution**

The binomial distribution \(\mathcal{B}(n, p)\) is defined by

\[
\Pr \{X = x\} = \binom{n}{x} p^x (1 - p)^{n-x} \quad \text{if } x \in \{0, 1, \ldots, n\}, \quad = 0 \quad \text{otherwise}.
\]

It can be simulated using two methods:

1. Since a binomial variable \(X\) is distributed as the sum of \(n\) Bernoulli variables \(\{Y_i\}_{i=1,\ldots,n}\) of parameter \(p\), \(X\) can be simulated by calculating the sum of \(n\) independent \(\mathcal{B}(p)\) variables obtained with the formula (3).

2. Since the binomial distribution \(\mathcal{B}(n, p)\) is a discrete distribution, it can be simulated using the formula (4) with

\[
p_x = \binom{n}{x} p^x (1 - p)^{n-x}.
\]

For large \(n\), both of these methods may induce long computational time, so the normal approximation can be used: \(X\) can be defined as the truncated value of a normal variable \(\mathcal{N}(\mu, \sigma^2)\) with

\[
\mu = np, \quad \sigma^2 = np(1 - p).
\]

Several methods for simulating normal variables shall be presented in section 3.
2.1.5 Poisson distribution

The Poisson distribution $P(\lambda)$ if defined by

$$\Pr \{X = x\} = \frac{\lambda^x}{x!} e^{-\lambda} \quad \text{if } x \in \{0, 1, 2, \ldots\},$$

$$= 0 \quad \text{otherwise.}$$

Two methods for simulating this distribution are presented here.

1. The method (2) can be generalized to the case of an infinity of possible values $(k = +\infty)$. In this case, the following recurrence formula can be used to calculate the probabilities $p_x$’s and the cdf:

$$p_x = \frac{\lambda p_{x-1}}{x},$$

$$F(x) = F(x - 1) + p_x = F(x - 1) + \frac{\lambda p_{x-1}}{x}.$$  

The corresponding MatLab function is given in appendix A.2.2, page 25.

2. Another method for simulating Poisson variables uses the following general property of homogeneous Poisson processes (see for example Taylor & Karlin [11], pp175-176): if events occur at dates separated by independent exponential durations $\mathcal{E}(\lambda)$, the total number of events observed in a time unit is distributed as a Poisson variable with parameter $\lambda$. $X$ can hence be obtained by cumulating independent exponential variables $\{Y_i\}_{i \geq 1}$ until the sum exceeds one:

$$X = x : \sum_{i=1}^{x} Y_i \leq 1 < \sum_{i=1}^{x+1} Y_i.$$  

The corresponding MatLab function is given in appendix A.2.3, page 25. A method for simulating exponential variables is presented in section 2.2.2, page 12.

2.2 Continuous distributions

2.2.1 Uniform distribution

The simplest continuous is the uniform distribution $\mathcal{U}_{[0,1]}$. It can easily be generalized to a variable $X$ uniformly distributed on any interval $[a; b]$ using the following formula:

$$X = a + (b - a) U.$$  

The fact that $X$ is uniformly distributed on $[a; b]$ comes straightforward since, for any $a \leq x \leq b$, one has

$$\Pr \{X \leq x\} = \Pr \left\{ U \leq \frac{x - a}{b - a} \right\} = \frac{x - a}{b - a}$$  

which is the distribution function $\mathcal{U}_{[a;b]}$.  

2.2.2 Inversion

A simple way of simulating a continuous random variable $X$ is given by the following result which is based on a general property of the cdf $F$:

$$F(x) = \Pr \{X \leq x\}.$$  

Figure 4 presents the principle of the inversion method which is based on the graph of the cdf function $F$. The two small regions around $x_1$ and $x_2$ have the same width but the slope of $F$ (i.e. the probability density $f$) is higher in $x_1$ than in $x_2$. Therefore, if $U$ is uniformly distributed on the vertical axis between 0 and 1, its reciprocal $F^{-1}(U)$ has more chance to be near $x_1$ than near $x_2$.

![Figure 4: Inversion method](image)

This method is summarized by proposition 1.

**Proposition 1** If $F$ is a continuous and strictly increasing cdf and if $U$ is uniformly distributed on the $[0, 1]$ interval, then $F^{-1}(U)$ has distribution $F$:

$$F^{-1}(U) \sim F.$$  

11
Proof. Since $F$ is continuous and strictly increasing, its inverse $F^{-1}$ exists and is defined for any $u \in [0, 1]$. For the particular value $u = F(x)$ equation (1) gives

$$F(x) = \Pr \{ U \leq F(x) \}.$$ 

Applying the inverse $F^{-1}$ to both hand sides of $U \leq F(x)$, we get

$$F(x) = \Pr \{ F^{-1}(U) \leq x \}$$

which proves that the random variable $X = F^{-1}(U)$ has distribution $F$. 

Application to the exponential distribution. The rejection method can be applied to the exponential distribution $\mathcal{E}(\lambda)$ which is defined by

$$F(x) = 1 - e^{-\lambda x}.$$ 

In this case, the inverse function is

$$F^{-1}(u) = -\frac{1}{\lambda} \ln(1 - u).$$ 

Using the simple following property:

$$U \sim \mathcal{U}[0, 1] \iff (1 - U) \sim \mathcal{U}[0, 1],$$

we obtain an exponential variable by calculating

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

The corresponding MatLab function is given in appendix A.2.4, page 26.

The rejection method can be used to simulate any distribution for which the inverse function $F^{-1}$ is known. It can also be applied for the normal distribution, as shown in section 3.1.2.

Generalization. As seen in section 2.1.3, this method can be applied to discrete distributions. Given its general principle (presented in figure 4), it is clear that it can be applied to any distribution:

- if $F$ is not continuous in $x_0$, $F^{-1}(U)$ shall have a probability $p_0 > 0$ to be equal to $x_0$;
- if $F$ is not strictly increasing, $F^{-1}(U)$ will never fall in the regions where $F(x)$ is constant, i.e. in the regions where the probability density is null.
2.2.3 Rejection method

Another general way of simulating continuous distribution is the rejection method. We introduce first for a distribution defined on a compact interval $[a, b]$ and having a bounded density:

$$
0 \leq f (x) \leq m \quad \text{if } x \in [a, b],
$$
$$
f (x) = 0 \quad \text{otherwise}.
$$

Such a density is presented in figure 5.

A random variable of distribution $F (x) = \int_{a}^{x} f (u)\,du$ can be simulated with the following algorithm. (The notations are those used in figure 5.)

Rejection algorithm.

Step 1. Simulate a point of coordinates $(X, Y)$ uniformly on the rectangle $(abcd)$:

$$
X \leftarrow a + (b - a) U_1, \quad Y \leftarrow m U_2;
$$

Step 2. If $f (X) \leq Y$ then keep $X^* \leftarrow X$, else goto step 1.
Proposition 2 The random variable $X^*$ simulated according to the rejection algorithm has distribution $F$.

Proof. For a given value $x_0$, one has

$$\Pr \{X^* \leq x_0\} = \Pr \{X \leq x_0 \mid Y \leq f(X)\} = \frac{\Pr \{X \leq x_0, Y \leq f(X)\}}{\Pr \{Y \leq f(X)\}}.$$ 

The numerator equals the proportion of the grey region of figure 5 in the surface of the whole rectangle $(abcd)$ so $\Pr \{X \leq x_0, Y \leq f(X)\} = F(x_0)[m(b-a)]^{-1}$ and the denominator is equal to the proportion of the surface under the density curve (which is 1) in $(abcd) : \Pr \{Y \leq f(X)\} = \left[\int_a^b f(x) \, dx\right] \times [m(b-a)]^{-1} = 1 \times [m(b-a)]^{-1}$. We hence get $\Pr \{X^* \leq x_0\} = F(x_0)$ and the proposition is proved. □

It is clear that the majoring constant $m$ must be as low as possible to avoid numerous rejections in the step 2 of the rejection algorithm.

The corresponding MatLab function is given in appendix A.2.5, page 26.

2.2.4 General rejection

The rejection method can be extended to distributions defined on an infinite domain. In this case, the rectangle $(abcd)$ can not be defined because its surface would necessarily be infinite. We use therefore a majoring distribution $G$.

In this section we consider the simulation of a distribution $F$ (of density $F$) and we assume that we are able to simulate a majoring distribution $G$ (of density $g$) such as

$$\exists m > 1, \quad \forall x : m \times g(x) \geq f(x).$$

$m$ is called the majoring constant.

For example, if the inverse function $G^{-1}$ is known, a variable $Y$ of distribution $G$ is obtained with $Y = G^{-1}(u)$.

Generalized rejection algorithm.

Step 1. Simulate a point with coordinates $(X,Y)$ such as

$$X \sim G, \quad Y \sim \mathcal{U}[0, mg(X)];$$

Step 2. If $Y \leq f(X)$ then keep $X^* \leftarrow X$, else goto step 1.

Proposition 3 The random variable $X^*$ simulated according to the generalized rejection algorithm has distribution $F$.

Proof. The proof is similar to this of proposition 2: one has

$$\Pr \{X^* \leq x_0\} = [\Pr \{Y \leq f(X)\}]^{-1} \times \{X \leq x_0, Y \leq f(X)\}$$

14
where

\[
\Pr \{X \leq x_0, Y \leq f (X)\} = \int_{-\infty}^{x_0} g(x) \Pr \{mg(x) U \leq f (x)\} \, dx
\]

\[
= \int_{-\infty}^{x_0} g(x) \Pr \{U \leq f (x) [mg(x)]^{-1}\} \, dx
\]

\[
= \int_{-\infty}^{x_0} g(x) \frac{f(x)}{mg(x)} \, dx = \frac{F(x_0)}{m},
\]

and \( \Pr \{Y \leq f (X)\} = F(+\infty) / m = m^{-1} \) so the proposition is proved. ■

An application of this method to the normal distribution is presented in section 3.1.3.
3 The normal distribution

The normal distribution is one of the most widely used distribution, especially in biological problems. We present here several method to simulate normal variables in both univariate and multivariate cases. In this section we shall denote \( \phi \) the density of the standard normal distribution \( \mathcal{N}(0, 1) \):

\[
\phi(x) = \frac{1}{\sqrt{2\pi}} \exp \left( -\frac{x^2}{2} \right)
\]

and \( \Phi \) its cdf:

\[
\Phi(x) = \int_{-\infty}^{x} \phi(t) \, dt.
\]

3.1 Univariate

We consider here the problem of simulating a standard normal variable \( X \). To simulate a normal variable \( Y \) with mean \( \mu \) and variance \( \sigma^2 \), it suffices to apply the simple transformation

\[
Y = \mu + \sigma X.
\]

3.1.1 Central Limit Theorem (CLT)

The simplest way of generating normal variables is an application of the central limit theorem.

If \( U \) is uniformly distributed over \([0, 1]\), then its mean and variance are

\[
\mathbb{E}(U) = \int_{0}^{1} u \, du = \frac{1}{2}, \quad \mathbb{V}(U) = \int_{0}^{1} (u - \mu)^2 \, du = \frac{1}{12}
\]

So, if \( \{U_1, U_2, \ldots, U_n\} \) are independent and uniformly distributed over \([0, 1]\), then their sum \( X = \sum_{i=1}^{n} U_i \) has mean \( n\mathbb{E}(U) = n/2 \) and variance \( n\mathbb{V}(U) = n/12 \) and, according to the central limit theorem, its has an asymptotically normal distribution:

\[
\frac{X - n\mu}{\sqrt{n\sigma}} \xrightarrow{n \to \infty} \mathcal{N}(0, 1).
\]

To simulate an approximately normal variable, one often use \( n = 12 \) so that \( \sqrt{n}\sigma = 1 \). We hence have the following formula:

\[
X = \left( \sum_{i=1}^{12} U_i \right) - 6.
\]

This variable has not exactly a normal distribution (for example, it never exceeds 6), but the greatest difference between its exact cdf and \( \Phi \) is about \( 10^{-4} \).
3.1.2 Inversion method

Although the inverse function $\Phi^{-1}$ has no analytical expression, the inversion method can be applied to the normal distribution using an approximation of $\Phi^{-1}$. As an example, we present here the rational approximation given in Bratley et al. [2]:

$$\Phi^{-1}(u) \simeq y = \frac{p_0 + p_1 y + p_2 y^2 + p_3 y^3 + p_4 y^4}{q_0 + q_1 y + q_2 y^2 + q_3 y^3 + q_4 y^4}$$

with

$$y = -\sqrt{2 \log (1 - u)}$$

and, with only five decimals,

$$p_0 = 3.2223 \times 10^{-1}, \quad p_1 = 1, \quad p_2 = 3.4224 \times 10^{-1}, \quad p_3 = 2.0423 \times 10^{-2}, \quad p_4 = 4.5364 \times 10^{-5},$$
$$q_0 = 9.9348 \times 10^{-2}, \quad q_1 = 5.8858 \times 10^{-1}, \quad q_2 = 5.3110 \times 10^{-1}, \quad q_3 = 1.0354 \times 10^{-1}, \quad q_4 = 3.8561 \times 10^{-3}.$$  

Other approximations can be found in Abramowitz and Stegun [1] or in Patel and Read [10].

Another way to get an approximation of $\Phi^{-1}$ is to use the erf function define for any $x \geq 0$:

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x \exp \left(-t^2\right) \, dt$$

which is related to $\Phi(x)$ by the following formulae:

$$\Phi(x) = \frac{1}{2} \left[1 + \text{erf} \left( \frac{x}{\sqrt{2}} \right) \right] \quad \text{if } x \geq 0,$$
$$= \frac{1}{2} \left[1 - \text{erf} \left( \frac{x}{\sqrt{2}} \right) \right] \quad \text{otherwise.}$$

Many software (such as MatLab) and numerical libraries give approximations of the erf function and of its inverse erf$^{-1}$ which is related to $\Phi^{-1}$ by the formula:

$$\Phi^{-1}(u) = +\sqrt{2} \text{erf}^{-1} (2u - 1) \quad \text{if } u \geq \frac{1}{2},$$
$$= -\sqrt{2} \text{erf}^{-1} (1 - 2u) \quad \text{otherwise.}$$

3.1.3 Rejection method

The generalized rejection method can be applied to the normal distribution. In this case a possible majoring distribution is the Laplace (or double exponential) one. A Laplace variable defined as an exponential variable multiply by either +1 or -1 with equal probabilities. Such a variable $Y$ is easily simulated with the following formula:

$$Z = -\ln(U_1) \sim \mathcal{E}(1), \quad \begin{cases} Y = +Z & \text{if } U_2 \leq \frac{1}{2}, \\ Y = -Z & \text{otherwise.} \end{cases}$$

The density of this distribution is

$$g(x) = \frac{1}{2} e^{-|x|}.$$
Figure 6 presents the rejection graph with two majoring constant $m$. The optimal one is

$$m = \sqrt{2e/\pi} \approx 1.3155.$$  

(This optimality is proved in appendix A.1.1.)

It is clear that if we use a non-optimal value of $m$ (such as $m = 2$), the generalized rejection algorithm (see section 2.2.4, page 14) will often fall in the rejection region (between the solid and the dotted lines) and the computational time will therefore be longer.

![Graph showing rejection method](image)

Figure 6: Application of the generalized rejection method to the normal distribution (solid line); the majoring distribution is the proportional to the Laplace one (dotted line)

The corresponding MatLab function is given in appendix A.2.6, page 26.

### 3.1.4 Box-Müller

Last (but not least), we present an exact way of simulating a couple $(X_1, X_2)$ of independent standard normal variables.

**Proposition 4** If $U_1$ and $U_2$ are independent uniform $U[0,1]$ variables, then

\[
X_1 = \sqrt{-2\ln U_1} \cos (2\pi U_2) \\
and \quad X_2 = \sqrt{-2\ln U_1} \sin (2\pi U_2)
\]
are independent normal standard variables.

**Proof.** The proof of this proposition is based on the expression of the density of a couple of normal variables in terms of polar coordinates. It is given in appendix A.1.2.

### 3.1.5 Comparison of the methods

Table 1 gives a comparison of the four different methods of simulating \( \mathcal{N}(0, 1) \) variables. In terms of means and standard deviations: the results are all coherent. All the means obtained are acceptable since the theoretical standard deviation of the mean is \( \sigma/n = 0.001 \).

The greatest difference concerns the computational times. The inversion method appears to be very long (this is essentially due to the calculation of approximate function \( \Phi^{-1} \)). The rejection method (because of the rejections) is more time-consuming than both CLT and Box-Müller methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Time</th>
<th>Mean</th>
<th>Variance</th>
<th>Skewness</th>
<th>Kurtosis</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLT</td>
<td>0.9910</td>
<td>+0.0072</td>
<td>1.0043</td>
<td>+0.0400</td>
<td>2.8856</td>
<td>−3.6380</td>
<td>3.7504</td>
</tr>
<tr>
<td>Inversion</td>
<td>8.4420</td>
<td>−0.0046</td>
<td>0.9698</td>
<td>+0.0407</td>
<td>2.9065</td>
<td>−3.0597</td>
<td>3.5390</td>
</tr>
<tr>
<td>Rejection</td>
<td>2.5440</td>
<td>−0.0055</td>
<td>0.9560</td>
<td>−0.0183</td>
<td>3.0500</td>
<td>−3.7620</td>
<td>3.9641</td>
</tr>
<tr>
<td>Box-Müller</td>
<td>1.1720</td>
<td>−0.0069</td>
<td>1.0176</td>
<td>−0.0138</td>
<td>2.9748</td>
<td>−3.7193</td>
<td>4.0410</td>
</tr>
</tbody>
</table>

Table 1: Comparison of four different methods for simulating standard normal variables: this results are obtained after \( n = 10000 \) simulations (the theoretical values of the skewness and kurtosis of the \( \mathcal{N}(0:1) \) distribution are respectively 0 and 3). This computations have been performed with the MatLab software on a Pentium II 450 MHz (RAM: 256 Mo).

### 3.2 Multivariate case

#### 3.2.1 Definition

The (finite) multivariate normal distribution \( \mathcal{N}_p(\mu, \Sigma) \) is the distribution of a \( p \)-dimensional random vector \( \mathbf{X} \) where

\[
\mathbf{X} = \begin{bmatrix} X_1 \\ \vdots \\ X_p \end{bmatrix}, \quad \mu = \begin{bmatrix} \mu_1 \\ \vdots \\ \mu_p \end{bmatrix}, \quad \Sigma = \begin{bmatrix} \sigma_{11} & \cdots & \sigma_{1p} \\ \vdots & \ddots & \vdots \\ \sigma_{p1} & \cdots & \sigma_{pp} \end{bmatrix}
\]

and, for any \( 1 \leq i, j \leq p \):

\[
\mu_i = \mathbb{E}(X_i), \quad \sigma_{ii} = \mathbb{V}(X_i), \quad \sigma_{ij} = \mathbb{Cov}(X_i, X_j).
\]

\( \Sigma \) is obviously symmetrical, it is furthermore generally positive definite.
3.2.2 Standard multivariate normal distribution

The standard multivariate normal distribution $N_p(0, I)$ generalizes the standard distribution $N(0, 1)$. With this distribution, all the Gaussian coordinates $X_i$ have mean 0 and variance 1 and are independent from each other.

This multivariate distribution can be easily simulated using $p$ times one of the methods described in section 3.1.

3.2.3 General multivariate normal distribution

A general method to simulate a $N_p(\mu, \Sigma)$ distribution through a standard $N_p(0, I)$ is based on a general property of the multivariate normal distribution (proposition 5) and on the Cholesky factorization (proposition 6).

**Proposition 5** If the random vector $X$ has distribution $N_p(\mu, \Sigma)$, then $Y = AX$ (where $A$ is any $(q \times p)$ matrix) has distribution $N_q(\mu, A \Sigma A')$.

**Proof.** The proof can be found, for example, in Patel & Read [10]. ■

**Proposition 6** If $A$ is a symmetric and positive definite matrix, then there exist a (lower) triangular matrix $B$ such as

$$A = BB'.$$

Furthermore, this factorization is unique if $B$ is required to have all its diagonal terms non-negative.

**Proof.** The proof can be found in Ciarlet [3], in Lascaux & Théodor [7], or in any work dedicated to matrices’ properties. ■

An algorithm for the Cholesky factorization is given in appendix A.3, page 27.

A combination of this two propositions can be made since any variance matrix $\Sigma$ is symmetric and (semi-)definite positive. Further more, the variance matrix $\Sigma$ of a random vector $Y$ is only semi-definite when one the (random) coordinates of $Y$ is a linear combination of the other coordinates. In this case, we suggest to simulate this coordinate separately.

Given this two results, we can propose the following algorithm to simulate any $N_p(\mu, \Sigma)$ distribution with $\Sigma$ definite.
Multivariate normal distribution algorithm.

Step 1. Simulate a random $N_p(0_p, I_p)$ vector $X$ using $p$ times any of the methods proposed in section 3.1 (page 16).

Step 2. Calculate the Cholevsky factorization $\Sigma = PP'$ using algorithm A.3 (page 27).

Step 3. Calculate $Y = \mu + PX$.

**Proposition 7**  The random vector $Y$ simulated according to the preceding algorithm has distribution $N_p(\mu, \Sigma)$.

**Proof.** According to proposition 5, $PX$ has distribution $N_p(0_p, P^tP)$ so $Y = \mu + PX$ has distribution $N_p(\mu, \Sigma)$.

### 3.2.4 Application to the bivariate case

Suppose we want to simulate a random normal vector $Y$ with mean $\mu$ and variance $\Sigma$:

$$
\mu = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \quad \Sigma = \begin{bmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2 \end{bmatrix}.
$$

$\rho$ is the correlation coefficient between $Y_1$ and $Y_2$. The Cholesky factorization of $\Sigma$ satisfies $PP' = \Sigma$ where

$$
P = \begin{bmatrix} a & 0 \\ b & c \end{bmatrix}
$$

and $a, c > 0$. Thus, one has

$$
PP' = \begin{bmatrix} a & 0 \\ b & c \end{bmatrix} \begin{bmatrix} a & b \\ b & c \end{bmatrix} = \begin{bmatrix} a^2 & ab \\ ab & b^2 + c^2 \end{bmatrix}.
$$

the equation $PP' = \Sigma$ gives the following system

$$
\begin{cases}
a^2 = \sigma_1^2 \\
b\rho = \rho\sigma_1\sigma_2 \\
b^2 + c^2 = \sigma_2^2
\end{cases} \quad \Leftrightarrow \quad \begin{cases}
a = \sigma_1 \\
b = \rho\sigma_2 \\
c = \sqrt{1 - \rho^2}\sigma_2
\end{cases}.
$$

If $X_1$ and $X_2$ are independent $N(0, 1)$ variables, then

$$
Y = \mu + PX = \begin{bmatrix} \mu_1 + \sigma_1X_1 \\ \mu_2 + \sigma_2\left(\rho X_1 + \sqrt{1 - \rho^2}X_2\right) \end{bmatrix}
$$

has distribution $N_2(\mu, \Sigma)$. 


References


A Appendix

A.1 Simulating normal variables

A.1.1 Rejection method.

Proof of the optimality of the constant $m = \sqrt{2e/\pi}$. For any real $x$, one has

$$\exp\left(-\frac{x^2}{2}\right) \leq \exp\left(\frac{1}{2} - |x|\right) \iff \frac{x^2}{2} - |x| + \frac{1}{2} \geq 0 \iff \frac{(|x| - 1)^2}{2} \geq 0$$

which always holds (the equality is obtained for $x = \pm 1$), so

$$\phi(x) = \frac{\exp\left(-\frac{x^2}{2}\right)}{\sqrt{2\pi}} \leq \frac{\exp\left(\frac{1}{2} - |x|\right)}{\sqrt{2\pi}} = \sqrt{\frac{2e}{\pi}} \times \frac{1}{2}e^{-|x|}$$

$m = \sqrt{2e/\pi}$ is therefore the optimal majoring constant.

A.1.2 Box and Müller method.

Proof of proposition 4. The joint distribution of two independent normal variables is

$$\phi(x, y) = \frac{1}{2\pi} \exp\left(-\frac{x^2 + y^2}{2}\right).$$

Applying the polar transform $x = \rho \cos \theta$ and $y = \rho \sin \theta$, one get

$$\phi(x, y) \, dxdy = \frac{1}{2\pi} \exp\left(-\frac{x^2 + y^2}{2}\right) \, dxdy = \frac{1}{2\pi} \exp\left(-\frac{\rho^2}{2}\right) \rho d\rho d\theta.$$

The function

$$f_{T,R}(\rho, \theta) = \frac{1}{2\pi} \times \rho \exp\left(-\frac{\rho^2}{2}\right)$$

can be viewed as the joint density of two independent variables $R$ and $T$ with respective density functions

$$f_R(\rho) = \rho \exp\left(-\frac{\rho^2}{2}\right) \quad \text{and} \quad f_T(\theta) = \frac{1}{2\pi}.$$

- The cdf associated to $f_R$ is

$$F_R(\rho) = \int_0^\rho r \exp\left(-\frac{r^2}{2}\right) \, dr = 1 - \exp\left(-\frac{\rho^2}{2}\right)$$

and the inverse function $F_R^{-1}$ is

$$F_R^{-1}(u) = \sqrt{-2 \ln(1 - u)},$$

so $R$ can be simulated using the inversion method with the following formula

$$R = \sqrt{-2 \ln(U_1)};$$
• $f_T$ is the density of the uniform distribution over $[0, 2\pi]$, so $T$ is obtained with

$$T = 2\pi U_2.$$ 

The proposition is proved thanks to the polar transform. ■
A.2 MatLab functions for some classical distributions

All the MatLab function presented here are based on the \texttt{rand(n,m)} function which returns an \((n \times m)\) matrix of independent uniform \(\mathcal{U}[0,1]\) variables.

A.2.1 General discrete distribution

The input parameter \(p\) is the vector of probabilities \([p_1, \ldots, p_k]\) such as \(\sum_{i=1}^{k} p_i = 1\) and where \(p_i = \Pr\{X = i\}\).

\[
\begin{align*}
\text{function } x &= \text{disc}(p) \\
& \quad U = \text{rand}(1); \\
& \quad x = 1; \\
& \quad pcum = p(1); \\
& \quad \text{while } U > pcum \\
& \quad \quad x = x + 1; \\
& \quad \quad pcum = pcum + p(x); \\
& \quad \text{end}
\end{align*}
\]

A.2.2 Poisson: inversion method

We present the function corresponding to the first method presented in section 2.1.5, page 10. The input parameter \texttt{lambda} is the parameter of the Poisson distribution \(\mathcal{P}(\lambda)\).

\[
\begin{align*}
\text{function } x &= \text{pois_inv}(\text{lambda}) \\
& \quad U = \text{rand}(1); \\
& \quad x = 0; \\
& \quad p = \exp(-\text{lambda}); \\
& \quad pcum = p; \\
& \quad \text{while } U > pcum \\
& \quad \quad x = x + 1; \\
& \quad \quad p = p*\text{lambda}/x; \\
& \quad \quad pcum = pcum + p; \\
& \quad \text{end}
\end{align*}
\]

A.2.3 Poisson: sum of exponential variables

We present the function corresponding to the second method presented in section 2.1.5, page 10. The input parameter is the same as the preceding function. This function uses the \texttt{exponent} presented in section A.2.4.

\[
\begin{align*}
\text{function } x &= \text{pois_exp}(\text{lambda}) \\
& \quad x = 0; \\
& \quad \text{Cum} = \text{exponent}(\text{lambda}); \\
& \quad \text{while } \text{Cum} < 1 \\
& \quad \quad x = x + 1;
\end{align*}
\]

25
Cum = Cum + exponen(lambda);
end

A.2.4 Exponential: inversion method
The input parameter lambda is the parameter of the exponential distribution \( E(\lambda) \).

function X = exponen(lambda)
    X = -log(rand(1))/lambda;
end

A.2.5 Rejection method
This function applies the method described in section 2.2.3, page 13. The parameters are:
- \( f \): the name of the MatLab function calculating the density \( f \),
- a, b: the bounds of the definition domain of \( f \),
- m: the majoring constant.

function X = reject('f',a,b,m)
    X = a + (b-a)*rand(1);
    Y = m*rand(1);
    while Y > f(X)
        X = a + (b-a)*rand(1);
        Y = m*rand(1);
    end
end

A.2.6 General rejection method: application to the normal distribution
This function applies the method described in section 6, page 18 to the standard normal distribution. No parameter is needed.

Laplace distribution. The majoring distribution is the Laplace which simulated with the following function. No parameter is needed for this function.

function X = laplace()
    X = -ln(rand(1));
    if rand(1) < 0.5
        X = -X;
    end
end

Standard normal distribution. The parameters mu and sigma are respectively the mean and the standard deviation: the majoring constant \( m = \sqrt{2e/\pi} \) is given in the function.

function X = norm_rej(mu,sigma)
    m = sqrt(2*exp(1)/pi);
\[ X = \text{laplace}(); \]
\[ Y = \text{rand}(1) \times m \times \exp(-\text{abs}(X))/2; \]
while \( Y > m \times \exp(-X \times X/2)/\text{sqrt}(2\times\pi) \)
\[ X = \text{laplace}(); \]
\[ Y = m \times \text{rand}(1) \times m \times \exp(-\text{abs}(X))/2; \]
end
\[ X = \mu + \text{sigma} \times X; \]

A.3 Cholevsky factorization

Bratley and al. [2] give an algorithm for the Cholevsky factorization \( B \cdot B' \) of a \((m \times m)\) matrix \( A \) where \( B \) is lower triangular. Denoting \( A = [a_{ij}]_{i,j=1...m} \) and \( B = [b_{ij}]_{i,j=1...m} \), the matrix \( B \) is calculated as follows.

**Cholevsky factorization algorithm.**

\[
\begin{align*}
\text{for } i \text{ from } 1 \text{ to } m, \\
& \quad \text{for } j \text{ from } 1 \text{ to } i - 1, \\
& \quad \quad b_{ij} \leftarrow \left( a_{ij} - \sum_{k=1}^{j-1} b_{ik}b_{jk} \right) / b_{jj}; \\
& \quad \quad b_{ji} \leftarrow 0; \\
& \quad \text{end}; \\
& \quad b_{ii} \leftarrow \left( a_{ii} - \sum_{k=1}^{i-1} b_{ik}^2 \right)^{1/2}; \\
\text{end.}
\end{align*}
\]