

## Chapter 2

# Numerical Generation of Random Numbers

The modelling of financial processes often requires also to take into account stochastic influences, e.g., the seemingly random development of a stock price in the future. In order to simulate a stochastic behaviour within a numerical simulation, one has to realize randomness on a computer, i.e., the generation of random numbers.

Possible applications (among others) include:

- Numerical realization and simulation of stochastic processes. This field in fact has a huge area of applications far beyond financial mathematics. Let us just mention traffic simulation, medicine, science and engineering.
- Monte–Carlo–Methods. We will come to these methods for a specific application later, they require in particular the availability of random numbers.

The main problem is that a computer is a *deterministic* calculating machine, i.e., any algorithm, any process on the computer is deterministic. Thus, the nature of a computer is in contrast to the generation of *random* numbers. Because of this one usually talks of *pseudo random numbers*, i.e., numbers that are generated in a deterministic way but that reflect a random behaviour in a ‘good’ way. Moreover, often random numbers mimicing a given distribution are required. First we analyze the generators of pseudo random numbers for uniformly distributed numbers. Other distributions will then be realized with the aid of suitable transformations.

## 2.1 Congruence Methods

We start with the maybe most simple family of methods, the so-called *congruence methods*.

**Definition 2.1.1** For  $M \in \mathbb{N}$  set  $\mathbb{Z}_M := \{0, \dots, M - 1\}$ . A congruence method of first order constructed by an initial value  $y_0 \in \mathbb{Z}_M$  with a function

$$f : \mathbb{Z}_M \rightarrow \mathbb{Z} \quad (2.1.1)$$

is a sequence  $(y_n)_{n \in \mathbb{N}} \subset \mathbb{Z}_M$  defined by the rule

$$y_{n+1} := f(y_n) \pmod{M} . \quad (2.1.2)$$

This method is called *linear*, if  $f$  is affine-linear, i.e., if there exist  $a, b \in \mathbb{Z}$  such that  $f(x) = ax + b$ .  $\square$

For the congruence method we can now easily prove the following properties.

**Theorem 2.1.2** Let the sequence  $(y_n)_{n \in \mathbb{N}}$  be generated by the congruence method. Then, the following statements hold:

- (a) The created sequence  $(y_n)_{n \in \mathbb{N}}$  has a period with the maximal length  $M$ .
- (b) For the linear congruence method with  $b = 0$  (the so called Prime-Modulo-Generator)  $y_m = 0$  must be excluded for all  $m \in \mathbb{N}$ .

**Proof:**

- (a) Because of  $\#\mathbb{Z}_M = M$  there exist at least two identical elements in  $\{y_0, \dots, y_M\}$ , i.e.  $\exists 0 \leq i \leq M - 1, \exists 1 \leq p \leq M$  such that  $y_i, y_{i+p} \in \{y_0, \dots, y_M\}$  and therefore  $y_i = y_{i+np}$  for all  $n \in \mathbb{N}$ .
- (b) For  $y_m = 0$  (for some  $m \in \mathbb{N}$ ) and  $b = 0$  we obtain

$$f(y_m) = ay_m + b = 0 = y_m ,$$

so that  $y_m = y_n$  for all  $n \geq m$ , i.e., we obtain a constant sequence, which of course is non-random.  $\square$

The periodicity of the generated sequence is of course a serious drawback of the congruence method. Thus, in practice  $M$  should be chosen as large as possible in order to obtain a maximal length of the period. However, Theorem 2.1.2 (a) gives only an *upper bound* for the length of the period, in practice it could even be much smaller as we have seen in (b). The next result gives a precise statement for the length of the period of the Prime-Modulo-Generator.

**Theorem 2.1.3** *Let  $M$  be a prime number. Then, the Prime-Modulo-Generator  $y_{n+1} = ay_n \pmod{M}$  has the smallest period  $M - 1$  if  $a$  is a primitive root of  $M$ , i.e., if*

$$(a^i - 1) \begin{cases} \not\equiv 0 \pmod{M}, & \text{if } 1 \leq i < M - 1, \\ \equiv 0 \pmod{M}, & \text{if } i = M - 1. \end{cases}$$

**Proof:** By Theorem 2.1.2 (b) we can assume  $y_0 \neq 0$ . For the sequence  $(z_n)_{n \in \mathbb{N}}$  with  $z_0 := y_0$ ,  $z_n := f(z_{n-1}) = az_{n-1}$  we obviously have  $z_n = a^n z_0$ . Thus  $y_n = z_n \pmod{M} = y_0$  holds if and only if  $a^n \equiv 1 \pmod{M}$ . Thus, by assumption on  $a$  we have  $n = M - 1$  which is the smallest period.  $\square$

**Example 2.1.4** *We consider the case  $M = 11$  with the choices of the parameters  $a = y_0 = 5$ . Note that in this case we have  $a^5 = 3125 = 11 \times 284 + 1$ , which implies  $a^5 \pmod{11} \equiv 1$ , i.e., we expect that the periodic length is equal to 5. In fact:*

$$\begin{aligned} y_1 &= 25 \pmod{11} = 3 \\ y_2 &= 15 \pmod{11} = 4 \\ y_3 &= 20 \pmod{11} = 9 \\ y_4 &= 45 \pmod{11} = 1 \\ y_5 &= 5 \pmod{11} = 5 = y_0 \end{aligned}$$

**Example 2.1.5** *The generator RANDU, which still is often used in mathematical software packages is a Prime-Modulo-Generator with  $a = 2^{16} + 3$  and  $M = 2^{31}$  (see exercises).*

**Example 2.1.6** *An example of a non-linear congruence method is the inverse congruence method, where we have*

$$f(x) = a\bar{x} + b, \quad a, b \in \mathbb{Z}_M, M \text{ prime,}$$

is used and  $\bar{x}$  is defined for a given  $x \in \mathbb{Z}_M$  as

$$\begin{cases} \bar{x} = 0, & \text{if } x = 0, \\ x\bar{x} \equiv 1 \pmod{M}, & \text{else.} \end{cases}$$

Obviously, the calculation of  $\bar{x}$  is the most expensive part from the numerical point of view. This can e.g. be done with the euclidean algorithm.

If the length of the period is  $M$ , the calculated pseudo random numbers are obviously uniformly distributed. If they are normalized through  $\frac{y_i}{M}$  on the unit interval  $[0, 1]$ , they can be subjected to statistical tests in order to check if the desired distribution is in fact matched. We will describe this in the next section.

## 2.2 Frequency and Gap Tests

Once a sequence of random numbers is generated, one of course wants to check if this sequence is of the desired distribution. For the uniform distribution one may look at a graphical visualization, where each random number within an interval is plotted with a different vertical coordinate. Such a visualization is shown in Figure 2.1. Even though this graphical visualization

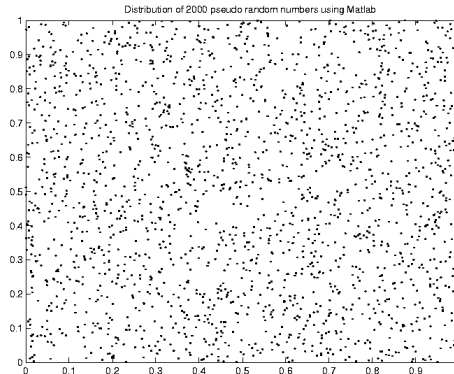


Figure 2.1: Visualization for the random number generator of MATLAB.

gives a first idea, it clearly has a number of serious drawbacks. First of all, it is more or less restricted to the uniform distribution. Moreover, and more seriously, it does not give any quantitative information. Thus, we describe in

this section how standard statistical tests can be used in order to investigate the quality of generated sequences of pseudo random numbers.

### 2.2.1 The $\chi^2$ -Test

Let us briefly recall the well-known  $\chi^2$ -test from statistics:

Divide  $[0, 1]$  into  $m + 1$  subintervals  $J_i = [x_i, x_{i+1})$ ,  $0 = x_0 < x_1 < \dots < x_{m+1} = 1$  and define the quantity

$$B_i := \#t_\nu \text{ in the interval } J_i$$

( $t_\nu$  are the pseudo random number,  $\nu = 1, \dots, n$ ). For the test to be meaningful every  $B_i$  should be at least of the size 5 to 10. Further let

$$E_i := \frac{n}{m + 1}$$

be the expected number of  $t_\nu$ 's in  $J_i$  in case of an equal distribution. Then, we define

$$\chi_{(n)}^2 := \sum_{i=0}^m \frac{(B_i - E_i)^2}{E_i}$$

and we have

$$\chi_{(n)}^2 \xrightarrow[n \rightarrow \infty]{d} \chi_m^2$$

where  $\chi_m^2$  is the chi-square distribution with density

$$f_m(x) := \begin{cases} \left(\frac{x}{2}\right)^{\frac{m}{2}} \frac{e^{-\frac{x}{2}}}{x \Gamma(\frac{m}{2})}, & x > 0, \\ 0, & x \leq 0, \end{cases}$$

and  $\Gamma(x) := \int_0^\infty t^{x-1} e^{-t} dt$  is the *Gamma function*.

A significant test results, if this test is realized for a large number (say  $N$ ) of realizations of a random number generator. Then, the quantity  $p_i$  of the calculated  $\chi^2$ -values in the intervals

$$\left[i - \frac{1}{2}, i + \frac{1}{2}\right), \quad i = 1, 2, \dots$$

are counted. If the points  $(i, p_i/N)$  are “close” to the probability density-function  $f_m$ , the random number generator has passed the  $\chi^2$  test. A possible quantitative measure could be the  $L_2$ -norm.

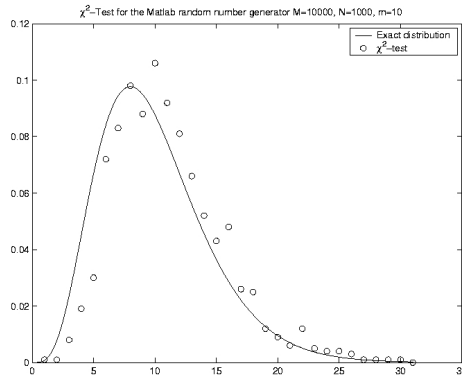


Figure 2.2: A  $\chi^2$ -test for the random number generator of MATLAB. Obviously, the test was succesful.

**Example 2.2.1** *Figure 2.2 shows the result of a  $\chi^2$ -test for the random number generator of MATLAB. The code for the  $\chi^2$ -test is also written in MATLAB.*

## 2.2.2 Gaps

**Definition 2.2.2** *For a given interval  $J \subset [0, 1]$  a sequence  $(t_n)_{n \in \mathbb{N}_0}$  is said to have a gap of length  $k$ , if there exists some  $n \in \mathbb{N}_0$  such that  $t_n, \dots, t_{n+k-1} \notin J$ , but  $t_{n+k} \in J$ .  $\square$*

For a corresponding test, choose  $h \in \mathbb{N}$ , and count the number of gaps of length  $0, 1, \dots, h-1, h$ . On this sequence of pseudo random numbers, the above  $\chi^2$ -test is applied.

For further information on random number generation and corresponding tests, we refer to [10].

## 2.3 Discrepancy

We have seen statistical tests to check the distribution of pseudo random numbers. We have concentrated on the uniform distribution. So far, we do not have a *measure* how good a uniform distribution is matched. We will now introduce such a measure.