4. Computer Generated Random Numbers: The Monte Carlo Method

4.1 Random Numbers

Up to now in this book we have considered the observation of random variables, but not prescriptions for producing them. It is in many applications useful, however, to have a sequence of values of a randomly distributed variable x. Since operations must often be carried out with a large number of such *random numbers*, it is particularly convenient to have them directly available on a computer. The correct procedure to create such random numbers would be to use a statistical process, e.g., the measurement of the time between two decays from a radioactive source, and to transfer the measured results into the computer. In practical applications, however, the random numbers are almost always calculated directly by the computer. Since this works in a strictly deterministic way, the resulting values are not really random, but rather can be exactly predicted. They are therefore called *pseudorandom*.

Computations with random numbers currently make up a large part of all computer calculations in the planning and evaluation of experiments. The statistical behavior which stems either from the nature of the experiment or from the presence of measurement errors can be simulated on the computer. The use of random numbers in computer programs is often called *the Monte Carlo method*.

We begin this chapter with a discussion of the representation of numbers in a computer (Sect. 4.2), which is indispensable for an understanding of what follows. The best studied method for the creation of uniformly distributed random numbers is the subject of Sects. 4.3–4.7. Sections 4.8 and 4.9 cover the creation of random numbers that follow an arbitrary distribution and the especially common case of normally distributed numbers. In the last two sections one finds discussion and examples of the Monte Carlo method in applications of numerical integration and simulation.

S. Brandt, *Data Analysis: Statistical and Computational Methods for Scientists and Engineers*,
DOI 10.1007/978-3-319-03762-2_4, © Springer International Publishing Switzerland 2014

In many examples and exercises we will simulate measurements with the Monte Carlo method and then analyze them. We possess in this way a *computer laboratory*, which allows us to study individually the influence of simulated measurement errors on the results of an analysis.

4.2 Representation of Numbers in a Computer

For most applications the representation of numbers used in a computation is unimportant. It can be of decisive significance, however, for the properties of computer-generated random numbers. We will restrict ourselves to the binary representation, which is used today in practically all computers. The elementary unit of information is the *bit*,* which can assume the values of 0 or 1. This is realized physically by two distinguishably different electric or magnetic states of a component in the computer.

If one has k bits available for the representation of an integer, then 1 bit is sufficient to encode the sign. The remaining $k-1$ bits are used for the binary representation of the absolute value in the form

$$a = a^{(k-2)}2^{k-2} + a^{(k-3)}2^{k-3} + \cdots + a^{(1)}2^1 + a^{(0)}2^0 \quad . \tag{4.2.1}$$

Here each of the coefficients $a^{(j)}$ can assume only the values 0 or 1, and thus can be represented by a single bit.

The binary representation for non-negative integers is

$$\begin{aligned} 00\cdots000 &= 0 \\ 00\cdots001 &= 1 \\ 00\cdots010 &= 2 \\ 00\cdots011 &= 3 \\ &\vdots \end{aligned}$$

One could simply use the first bit to encode the sign and represent the corresponding negative numbers such that in the first bit the 0 is replaced by a 1. That would give, however, two different representations for the number zero, or rather $+0$ and -0. In fact, one uses for negative numbers the "complementary representation"

$$\begin{aligned} 11\cdots111 &= -1 \\ 11\cdots110 &= -2 \\ 11\cdots101 &= -3 \\ &\vdots \end{aligned}$$

*Abbreviation of *binary digit*.

Then using k bits, integers in the interval

$$-2^{k-1} \le x \le 2^{k-1} - 1 \tag{4.2.2}$$

can be represented.

In most computers 8 bit are grouped together into one *byte*. Four bytes are generally used for the representation of integers, i.e., $k = 32$, $2^{k-1} - 1 = 2\,147\,483\,647$. In many small computers only two bytes are available, $k = 16$, $2^{k-1} - 1 = 32\,767$. This constraint (4.2.2) must be taken into consideration when designing a program to generate random numbers.

Before turning to the representation of fractional numbers in a computer, let us consider a finite decimal fraction, which we can write in various ways, e.g.,

$$x = 17.23 = 0.1723 \cdot 10^2$$

or in general

$$x = M \cdot 10^e \quad .$$

The quantities M and e are called the *mantissa* and *exponent*, respectively. One chooses the exponent such that the mantissa's nonzero digits are all to the right of the decimal point, and the first place after the decimal point is not zero. If one has available n decimal places for the representation of the value M, then

$$m = M \cdot 10^n$$

is an integer. In our example, $n = 4$ and $m = 1723$. In this way the decimal fraction d is represented by the two integers m and e.

The representation of fractions in the binary system is done in a completely analogous way. One decomposes a number of the form

$$x = M \cdot 2^e \tag{4.2.3}$$

into a mantissa M and exponent e. If n_m bits are available for the representation of the mantissa (including sign), it can be expressed by the integer

$$m = M \cdot 2^{n_m - 1} \quad , \qquad -2^{n_m - 1} \le m \le 2^{n_m - 1} - 1 \quad . \tag{4.2.4}$$

If the exponent with its sign is represented by n_e bits, then it can cover the interval

$$-2^{n_e} \le e \le 2^{n_e} - 1 \quad . \tag{4.2.5}$$

In our Java classes we use floating-point numbers of the type double with 64 bit, $n_m = 53$ for the mantissa and $n_e = 11$ for the exponent.

For the interval of values in which a floating point number can be represented in a computer, the constraint (4.2.2) no longer applies but one has rather the weaker condition

$$2^{e_{\min}} < |x| < 2^{e_{\max}} \quad . \tag{4.2.6}$$

Here $e_{\min}$ and $e_{\max}$ are given by (4.2.5). If 11 bit are available for representing the exponent (including sign), then one has $e_{\max} = 2^{10} - 1 = 1023$. Therefore, one has the constraint $|x| < 2^{1023} \approx 10^{308}$.

When computing with floating point numbers, the concept of the *relative precision* of the representation is of considerable significance. There are a fixed number of binary digits corresponding to a fixed number of decimal places available for the representation of the mantissa M. If we designate by α the smallest possible mantissa, then two numbers x_1 and x_2 can still be represented as being distinct if

$$x_1 = x = M \cdot 2^e \quad , \qquad x_2 = (M + \alpha) \cdot 2^e \quad .$$

The *absolute precision* in the representation of x is thus

$$\Delta x = x_1 - x_2 = \alpha \cdot 2^e \quad ,$$

which depends on the exponent of x. The relative precision

$$\frac{\Delta x}{x} = \frac{\alpha}{M}$$

is in contrast independent of x. If n binary digits are available for the representation of the mantissa, then one has $M \approx 2^n$, since the exponent is chosen such that all n places for the mantissa are completely used. The smallest possible mantissa is $\alpha = 2^0$, so that the *relative precision* in the representation of x is

$$\frac{\Delta x}{x} = 2^{-n} \quad . \tag{4.2.7}$$

4.3 Linear Congruential Generators

Since, as mentioned, computers work in a strictly deterministic way, all (pseudo)"-random numbers generated in a computer are in the most general case a function of all of the preceding (pseudo)random numbers[†]

$$x_{j+1} = f(x_j, x_{j-1}, \ldots, x_1) \quad . \tag{4.3.1}$$

Programs for creating random numbers are called *random number generators*.

[†]Since the numbers are pseudorandom and not strictly random, we use the notation x in place of x.

The best studied algorithm is based on the following rule,

$$x_{j+1} = (a\,x_j + c) \bmod m \quad . \tag{4.3.2}$$

All of the quantities in (4.3.2) are integer valued. Generators using this rule are called linear congruential generators (LCG). The symbol mod m or modulo m in (4.3.2) means that the expression before the symbol is divided by m and only the remainder of the result is taken, e.g., $6 \bmod 5 = 1$. Each random number made by an LCG according to the rule (4.3.2) depends only on the number immediately preceding it and on the constant a (the *multiplier*), on c (the *increment*), and on m (the *modulus*). When these three constants and one *initial value* x_0 are given, the infinite sequence of random numbers $x_0, x_1, \ldots$ is determined.

The sequence is clearly periodic. The maximum period length is m. Only partial sequences that are short compared to the period length are useful for computations.

Theorem on the maximum period of an LCG with $c \neq 0$:
An LCG defined by the values m, a, c, and x_0 has the period m if and only if

(a) c and m have no common factors;

(b) $b = a - 1$ is a multiple of p for every prime number p that is a factor of m;

(c) b is a multiple of 4 if m is a multiple of 4.

The proof of this theorem as well as the theorems of Sect. 4.4 can be found in, e.g., KNUTH [2].

A simple example is $c = 3$, $a = 5$, $m = 16$. One can easily compute that $x_0 = 0$ results in the sequence

$$0, 3, 2, 13, 4, 7, 6, 1, 8, 11, 10, 5, 12, 15, 14, 9, 0, \ldots \quad .$$

Since the period m can only be attained when all m possible values are actually assumed, the choice of the initial value x_0 is unimportant.

4.4 Multiplicative Linear Congruential Generators

If one chooses $c = 0$ in (4.3.2), then the algorithm simplifies to

$$x_{j+1} = (a x_j) \bmod m \quad . \tag{4.4.1}$$

Generators based on this rule are called multiplicative linear congruential generators (MLCG). The computation becomes somewhat shorter and thus

faster. The exact value zero, however, can no longer be produced (except for the unusable sequence 0, 0, ...). In addition the period becomes shorter. Before giving the theorem on the maximum period length for this case, we introduce the concept of the primitive element modulo m.

Let a be an integer having no common factors (except unity) with m. We consider all a for which $a^{\lambda} \bmod m = 1$ for integer λ. The smallest value of λ for which this relation is valid is called the *order* of a modulo m. All values a having the same largest possible order $\lambda(m)$ are called *primitive elements* modulo m.

Theorem on the order $\lambda(m)$ of a primitive element modulo m: For every integer e and prime number p

$$\begin{aligned} &\lambda(2) = 1 \quad ; \\ &\lambda(4) = 2 \quad ; \\ &\lambda(2^e) = 2^{e-2} \quad , \qquad e > 2 \quad ; \\ &\lambda(p^e) = p^{e-1}(p-1) \quad , \qquad p > 2 \quad . \end{aligned} \tag{4.4.2}$$

Theorem on primitive elements modulo p^e: The number a is a primitive element modulo p^e if and only if

$$\begin{aligned} &a \text{ odd} \ , \quad p^e = 2 \ ; \\ &a \bmod 4 = 3 \ , \quad p^e = 4 \ ; \\ &a \bmod 8 = 3, 5, 7 \ , \quad p^e = 8 \ ; \\ &a \bmod 8 = 3, 5 \ , \quad p = 2 \ , \quad e > 3 \ ; \\ &a \bmod p \neq 0 \ , \quad a^{(p-1)/q} \bmod p \neq 1 \ , \quad p > 2 \ , \quad e = 1 \ , \\ &\qquad q \text{ every prime factor of } p-1 \ ; \\ &a \bmod p \neq 0 \ , \quad a^{p-1} \bmod p^2 \neq 1 \ , \quad a^{(p-1)/q} \bmod p \neq 1 \ , \\ &\qquad p > 2 \ , \quad e > 1 \ , \quad q \text{ every prime factor of } p-1 \ . \end{aligned} \tag{4.4.3}$$

For large values of p the primitive elements must be determined with computer programs with the aid of this theorem.

Theorem on the maximum period of an MLCG: The maximum period of an MLCG defined by the quantities m, a, $c = 0$, x_0 is equal to the order $\lambda(m)$. This is attained if the multiplier a is a primitive element modulo m and when the initial value x_0 and the multiplier m have no common factors (except unity).

In fact, MLC generators with $c = 0$ are frequently used in practice. There are two cases of practical significance in choosing the multiplier m.

(i) $m = 2^e$: Here $m-1$ can be the largest integer that can be represented on the computer. According to (4.4.2) the maximum attainable period length is $m/4$.

(ii) $m = p$: If m is a prime number, the period of $m-1$ can be attained according to (4.4.2).

4.5 Quality of an MLCG: Spectral Test

When producing random numbers, the main goal is naturally not just to attain the longest possible period. This could be achieved very simply with the sequence 0, 1, 2, …, $m-1$, 0, 1, …. Much more importantly, the individual elements within a period should follow each other "randomly". First the modulus m is chosen, and then one chooses various multipliers a corresponding to (4.4.3) that guaranty a maximum period. One then constructs generators with the constants a, m, and $c = 0$ in the form of computer programs and checks with statistical tests the randomness of the resulting numbers. General tests, also applicable to this particular question, will be discussed in Sect. 8. The spectral test was especially developed for investigating random numbers, in particular for detecting non-random dependencies between neighboring elements in a sequence.

In a simple example we first consider the case $a = 3, m = 7, c = 0, x_0 = 1$ and obtain the sequence

$$1, 3, 2, 6, 4, 5, 1, \ldots \quad .$$

We now form pairs of neighboring numbers

$$(x_j, x_{j+1}) \quad , \qquad j = 0, 1, \ldots, n-1 \quad . \tag{4.5.1}$$

Here n is the period, which in our example is $n = m - 1 = 6$. In Fig. 4.1 the number pairs (4.5.1) are represented as points in a two-dimensional Cartesian coordinate system. We note – possibly with surprise – that they form a regular lattice. The surprise is somewhat less, however, when we consider two features of the algorithm (4.3.2):

(i) All coordinate values x_j are integers. In the accessible range of values $1 \le x_j \le n$ there are, however, only n^2 number pairs (4.5.1) for which both elements are integer. They lie on a lattice of horizontal and vertical lines. Two neighboring lines have a separation of one.

(ii) There are only n different pairs (4.5.1), so that only a fraction of the n^2 points mentioned in (i) are actually occupied.

We now go from integer numbers x_j to transformed numbers

$$u_j = x_j/m \tag{4.5.2}$$

with the property

$$0 < u_j < 1 \quad . \tag{4.5.3}$$

For simplicity we assume that the sequence $x_0, x_1, \ldots$ has the maximum possible period m for an MLC generator. The pairs

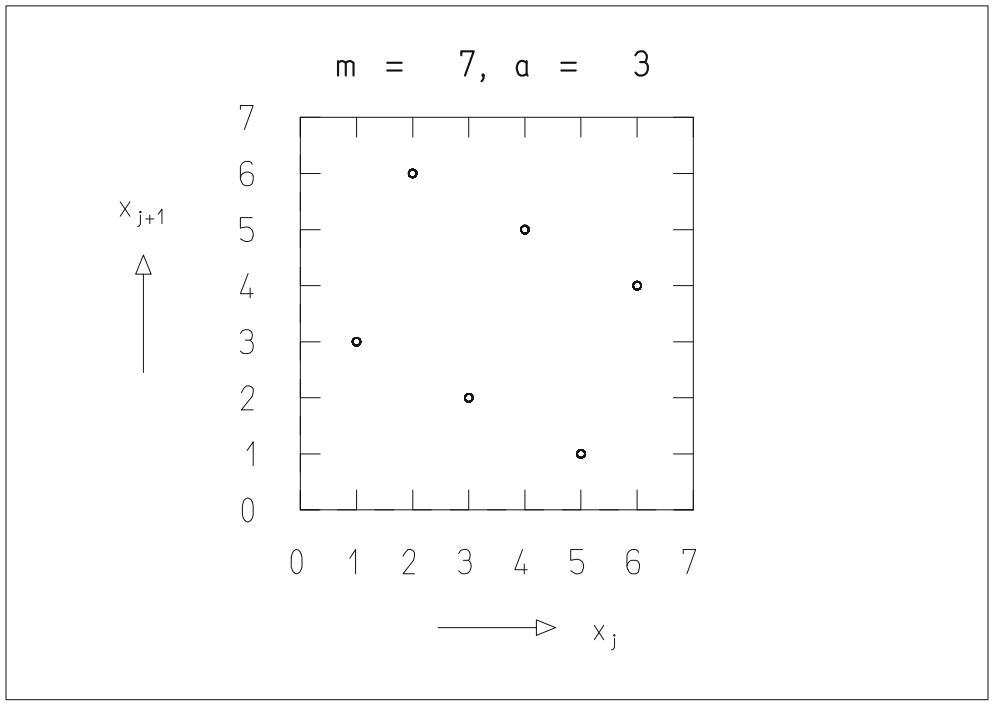

Fig. 4.1: Diagram of number pairs (4.5.1) for $a = 3$, $m = 7$.

$$(u_j, u_{j+1}) \quad , \qquad j = 0, 1, \ldots, m-1 \quad , \tag{4.5.4}$$

lie in a square whose side has unit length. Because the x_j are integers, the spacing between the horizontal or vertical lattice lines on which the points (4.5.4) must lie is $1/m$. By far not all of these points, however, are occupied. A finite family of lines can be constructed which pass through those points that are actually occupied. We consider now the spacing of neighboring lines within a family, look for the family for which this distance is a maximum, and call this d_2.

If the distances between neighboring lattice lines for all families are approximately equal, we can then be certain of having a maximally uniform distribution of the occupied lattice points on the unit square. Since this distance is $1/m$ for a completely occupied lattice (m^2 points), we obtain for a uniformly occupied lattice with m points a distance of $d_2 \approx m^{-1/2}$. With a very nonuniform lattice one obtains the considerably larger value $d_2 \gg m^{-1/2}$.

If one now considers not only pairs (4.5.4), but t-tuples of numbers

$$(u_j, u_{j+1}, \ldots, u_{j+t-1}) \quad , \tag{4.5.5}$$

one sees that the corresponding points lie on families of $(t-1)$-dimensional hyperplanes in a t-dimensional cube whose side has unit length. Let us investigate as before the distance between neighboring hyperplanes of a family. We determine the family with the largest spacing and designate this by d_t. One expects for a uniform distribution of points (4.5.5) a distance

$$d_t \approx m^{-1/t} \quad . \tag{4.5.6}$$

If the lattice is nonuniform, however, we expect

$$d_t \gg m^{-1/t} \quad . \tag{4.5.7}$$

The situations (4.5.6) and (4.5.7) are shown in Fig. 4.2. Naturally one tries to achieve as uniform a lattice as possible. One should note that there is at least a distance (4.5.6) between the lattice points. The lowest decimal places of random numbers are therefore not random, but rather reflect the structure of the lattice.

Theoretical considerations give an upper limit on the smallest possible lattice spacing,

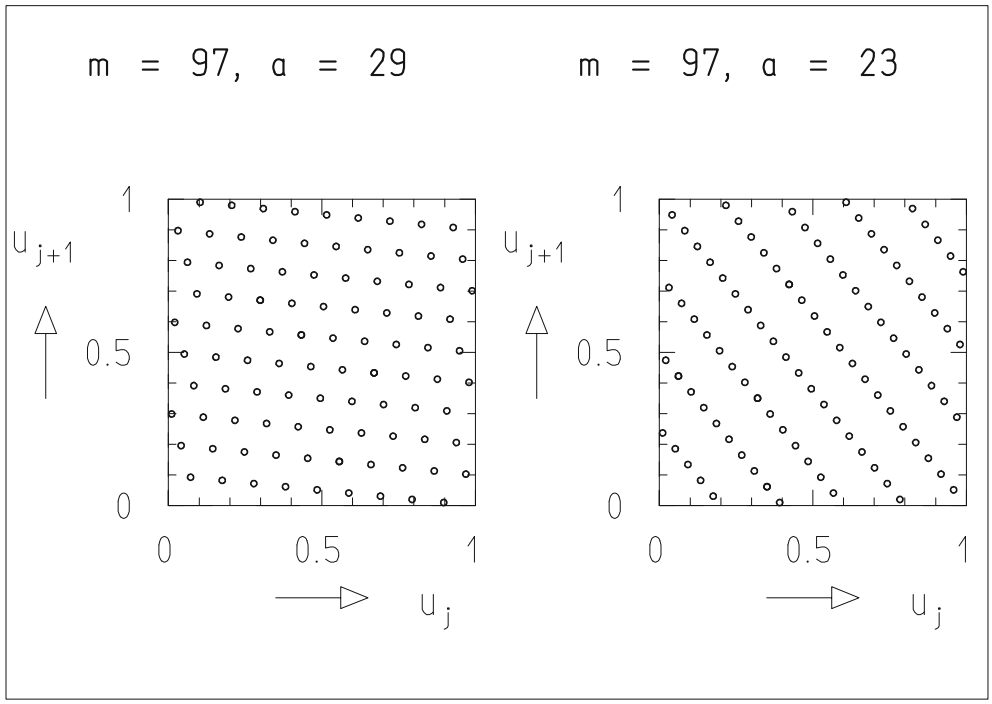

Fig. 4.2: Diagram of number pairs (4.5.4) for various small values of a and m.

Table 4.1: Suitable moduli m and multipliers a for portable MLC generators for computers with 32-bit (16-bit) integer arithmetic.

32 bit		16 bit	
m	a	m	a
2 147 483 647	39 373	32 749	162
2 147 483 563	40 014	32 363	157
2 147 483 399	40 692	32 143	160
2 147 482 811	41 546	32 119	172
2 147 482 801	42 024	31 727	146
2 147 482 739	45 742	31 657	142

$$d_t \geq d_t^* = c_t m^{-1/t} \quad . \tag{4.5.8}$$

The constants c_t are of order unity. They have the numerical values [2]

$$\begin{aligned} &c_2 = (4/3)^{-1/4} \quad , \quad c_3 = 2^{-1/6} \quad , \quad c_4 = 2^{-1/4} \quad , \quad c_5 = 2^{-3/10} \quad , \\ &c_6 = (64/3)^{-1/12} \quad , \quad c_7 = 2^{-3/7} \quad , \quad c_8 = 2^{-1/2} \quad . \end{aligned} \tag{4.5.9}$$

The *spectral test* can now be carried out as follows. For given values (m, a) of the modulus and multiplier of an MLCG one determines with a computer algorithm [2] the values $d_t(m, a)$ for small t, e.g., $t = 2, 3, \ldots, 6$. One constructs the test quantities

$$S_t(m, a) = \frac{d_t^*(m)}{d_t(m, a)} \tag{4.5.10}$$

and accepts the generator as usable if the $S_t(m, a)$ do not exceed a given limit. Table 4.1 gives the results of extensive investigations by L'ECUYER [3]. The moduli m are prime numbers close to the maximum integer values representable by 16 or 32 bit. The multipliers are primitive elements modulo m. They fulfill the requirement $a < \sqrt{m}$ (see Sect. 4.6). The prime numbers were chosen such that a does not have to be much smaller than $\sqrt{m}$, but the condition (m, a) in Table 4.1 $S_t(m, a) > 0.65$, $t = 2, 3, \ldots, 6$, still applies.

4.6 Implementation and Portability of an MLCG

By *implementation* of an algorithm one means its realization as a computer program for a specific type of computer. If the program can be easily transferred to other computer types and gives there (essentially) the same results, then the program is said to be *portable*. In this section we will give a portable implementation of an MLCG, as realized by WICHMANN and HILL [4] and L'ECUYER [3].

A program that implements the rule (4.4.1) is certain to be portable if the computations are carried out exclusively with integers. If the computer has k bits for the representation of an integer, then all numbers between $-m-1$ and m for $m < 2^{k-1}$ are available.

We now choose a multiplier a with

$$a^2 < m \tag{4.6.1}$$

and define

$$q = m \text{ div } a \quad , \qquad r = m \bmod a \quad , \tag{4.6.2}$$

so that

$$m = aq + r \quad . \tag{4.6.3}$$

The expression m div a defined by (4.6.2) and (4.6.3) is the integer part of the quotient m/a. We now compute the right-hand side of (4.4.1), where we leave off the index j and note that $[(x \text{div } q)m] \bmod m = 0$, since x div q is an integer:

$$\begin{aligned} [ax] \bmod m &= [ax - (x \text{ div } q)m] \bmod m \\ &= [ax - (x \text{ div } q)(aq + r)] \bmod m \\ &= [a\{x - (x \text{ div } q)q\} - (x \text{ div } q)r] \bmod m \\ &= [a(x \bmod q) - (x \text{ div } q)r] \bmod m \quad . \end{aligned} \tag{4.6.4}$$

Since one always has $0 < x < m$, it follows that

$$a(x \bmod q) < aq \leq m \quad , \tag{4.6.5}$$

$$(x \text{ div } q)r < [(aq + r) \text{ div } q]r = ar < a^2 < m \quad . \tag{4.6.6}$$

In this way both terms in square brackets in the last line of (4.6.4) are less than m, so that the bracketed expression remains in the interval between $-m$ and m.

In the Java class DatanRandom we have implemented the expression (4.6.4) in the following three lines, in which all variables are integer:

```
k = x / Q;
x = A * (x - k * Q) - k * R;
if(x < 0) x = x + M;
```

One should note that division of two integer variables results directly in the integer part of the quotient. The first line therefore yields x div q and the last line $ax \bmod m$, respectively.

The method DatanRandom.mlcg yields a partial sequence of random numbers of length N. Each time the subroutine is called, an additional partial sequence is produced. The period of the entire sequence is

$m - 1 = 2\,147\,483\,562$. The computation is carried out entirely with integer arithmetic, ensuring portability. The output values are, however, floating point valued because of the division by m, and therefore correspond to a uniform distribution between 0 and 1.

Often one would like to interrupt a computation requiring many random numbers and continue it later starting from the same place. In this case one can read out and store the last computed (integer) random number directly before the interruption, and use it later for producing the next random number. In the technical terminology one calls such a number the *seed* of the generator.

It is sometimes desirable to be able to produce non-overlapping partial sequences of random numbers not one after the other but rather independently. In this way one can, for example, carry out parts of larger simulation problems simultaneously on several computers. As seeds for such partial sequences one uses elements of the total sequence separated by an amount greater than the length of each partial sequence. Such seeds can be determined without having to run through the entire sequence. From (4.4.1) it follows that

$$x_{j+n} = (a^n x_j) \bmod m = [(a^n \bmod m) x_j] \bmod m \quad . \tag{4.6.7}$$

L'ECUYER [3] suggests setting $n = 2^d$ and choosing some seed x_0. The expression $a^{2^d} \bmod m$ can be computed by beginning with a and squaring it d times modulo m. Then one computes x_n using (4.6.7) and obtains correspondingly $x_{2n}, x_{3n}, \ldots$.

4.7 Combination of Several MLCGs

Since the period of an MLCG is at most $m - 1$, and since m is restricted to $m < 2^{k-1} - 1$ where k is the number of bits available in the computer for the representation of an integer, only a relatively short period can be attained with a single MLCG. WICHMANN and HILL [4] and L'ECUYER [3] have given a procedure for combining several MLCGs, which allows for very long periods. The technique is based on the following two theorems.

Theorem on the sum of discrete random variables, one of which comes from a discrete uniform distribution: If $\mathsf{x}_1, \ldots, \mathsf{x}_\ell$ are independent random variables that can only assume integer values, and if x_1 follows a discrete uniform distribution, so that

$$P(\mathsf{x}_1 = n) = \frac{1}{d} \quad , \qquad n = 0, 1, \ldots, d-1 \quad ,$$

then

$$\mathsf{x} = \left(\sum_{j=1}^{\ell} \mathsf{x}_j \right) \bmod d \tag{4.7.1}$$

also follows this distribution.

We first demonstrate the proof for $\ell = 2$, using the abbreviations $\min(\mathsf{x}_2) = a$, $\max(\mathsf{x}_2) = b$. One has

$$\begin{aligned} P(\mathsf{x} = n) &= \sum_{k=0}^{\infty} P(\mathsf{x}_1 + \mathsf{x}_2 = n + kd) \\ &= \sum_{i=a}^{b} P(\mathsf{x}_2 = i) P(\mathsf{x}_1 = (n-i) \bmod d) \\ &= \frac{1}{d} \sum_{i=a}^{b} P(\mathsf{x}_2 = i) = \frac{1}{d} \quad . \end{aligned}$$

For $\ell = 3$ we first construct the variable $\mathsf{x}'_1 = \mathsf{x}_1 + \mathsf{x}_2$, which follows a discrete uniform distribution between 0 and $d-1$, and then the sum $\mathsf{x}'_1 + \mathsf{x}_3$, which has only two terms and therefore possesses the same property. The generalization for $\ell > 3$ is obvious.

Theorem on the period of a family of generators: Consider the random variables $\mathsf{x}_{j,i}$ coming from a generator j with a period p_j, so that the generator gives a sequence $\mathsf{x}_{j,0}, \mathsf{x}_{j,1}, \ldots, \mathsf{x}_{j,p_j-1}$. We consider now ℓ generators $j = 1, 2, \ldots, \ell$ and the sequence of ℓ-tuples

$$\mathbf{x}_i = \{\mathsf{x}_{1,i}, \mathsf{x}_{2,i}, \ldots, \mathsf{x}_{\ell,i}\} \quad , \qquad i = 0, 1, \ldots \quad . \tag{4.7.2}$$

Its period p is the smallest common multiple of the periods $p_1, p_2, \ldots, p_\ell$ of the individual generators. The proof is obtained directly from the fact that p is clearly a multiple of each p_j.

We now determine the maximum value of the period p. If the ℓ individual MLCGs have prime numbers m_j as moduli, then their periods are $p_j = m_j - 1$ and are therefore even. Therefore one has

$$p \leq \frac{\prod_{j=1}^{\ell} (m_j - 1)}{2^{\ell-1}} \quad . \tag{4.7.3}$$

Equality results if the quantities $(m_j - 1)/2$ possess no common factors.

The first theorem of this section can now be used to construct a sequence with period given by (4.7.3). One forms first the integer quantity

$$z_i = \left(\sum_{j=1}^{\ell} (-1)^{j-1} x_{j,i} \right) \bmod (m_1 - 1) \quad . \tag{4.7.4}$$

The alternating sign in (4.7.4), which simplifies the construction of the modulus function, does not contradict the prescription of (4.7.1), since one could also use in place of $x_2, x_4, \ldots$, the variables $x_2' = -x_2$, $x_4' = -x_4$, ... The quantity z_i can take on the values

$$z_i \in \{0, 1, \ldots, m_1 - 2\} \quad . \tag{4.7.5}$$

The transformation to floating point numbers

$$u_i = \begin{cases} z_i / m_1 , & z_i > 0 \\ (m_1 - 1)/m_1 , & z_i = 0 \end{cases} \tag{4.7.6}$$

gives values in the range $0 < u_i < 1$.

In the method DatanRandom.ecuy we use the techniques, assembled above, to produce uniformly distributed random numbers with a long period. We combine two MLCGs with $m_1 = 2\,147\,483\,563$, $a_1 = 40\,014$, $m_2 = 2\,147\,483\,399$, $a_2 = 40\,692$. The numbers $(m_1 - 1)/2$ and $(m_2 - 1)/2$ have no common factor. Therefore the period of the combined generator is, according to (4.7.3),

$$p = (m_1 - 1)(m_2 - 1)/2 \approx 2.3 \cdot 10^{18} \quad .$$

The absolute values of all integers occurring during the computation remain in the range $\leq 2^{31} - 85$. The resulting floating point values u are in the range $0 < u < 1$. One does not obtain the values 0 or 1, at least if 23 or more bits are available for the mantissa, which is almost always the case when representing floating point numbers with 32 bit. The program with the given values of m_1, m_2, a_1, a_2 has been subjected to the spectral test and to many other tests by L'ECUYER [3], who has provided a PASCAL version. He determined that it satisfied all of the requirements of the tests.

Figure 4.3 illustrates the difference between the simple MLCG and the combined generator. For the simple MLCG one can still recognize a structure in a scatter plot of the number pairs (4.5.4), although with an expansion of the abscissa by a factor of 1000. The corresponding diagram for the combined generator appears, in contrast, to be completely without structure. For each diagram one million pairs of random numbers were generated. The plots correspond only to a narrow strip on the left-hand edge of the unit square.

In order to initialize non-overlapping partial sequences one can use two methods:

(i) One applies the procedure discussed in connection with (4.6.7) to both MLCGs, naturally with the same value n, in order to construct pairs of seeds for each partial sequence.

(ii) It is considerably easier to use the same seed for the first MLCG for every partial sequence. For the second MLCG one uses an arbitrary seed for the first partial sequence, the following random number from the second MLCG for the second partial sequence, etc. In this way one obtains partial sequences that can reach a length of $(m_1 - 1)$ without overlapping.

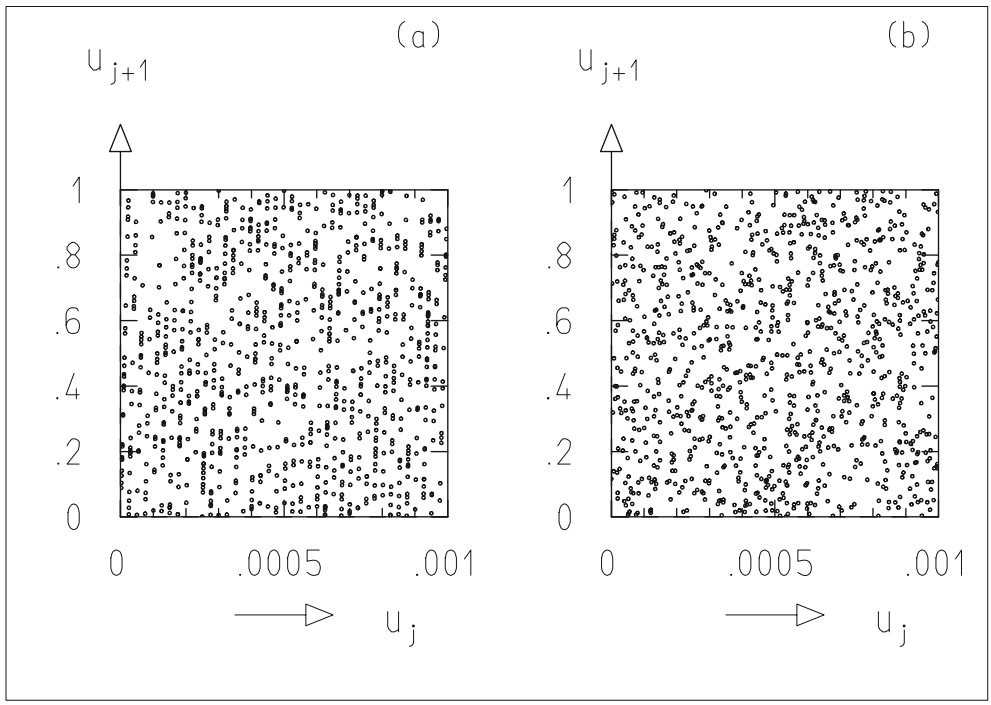

Fig. 4.3: Scatter plots of number pairs (4.5.4) from (**a**) a MLC generator and (**b**) a combined generator. The methods DatanRandom.mclg and DatanRandom.ecuy, respectively, were used in the generation.

4.8 Generation of Arbitrarily Distributed Random Numbers

4.8.1 Generation by Transformation of the Uniform Distribution

If x is a random variable following the uniform distribution,

$$f(x) = 1 \quad , \quad 0 \le x < 1 \quad ; \qquad f(x) = 0 \quad , \quad x < 0 \quad , \quad x \ge 1 \quad , \tag{4.8.1}$$

and y is a random variable described by the probability density $g(y)$, the transformation (3.7.1) simplifies to

$$g(y)\,\mathrm{d}y = \mathrm{d}x \quad . \tag{4.8.2}$$

We use the distribution function $G(y)$, which is related to $g(y)$ through $\mathrm{d}G(y)/\mathrm{d}y = g(y)$, and write (4.8.2) in the form

$$\mathrm{d}x = g(y)\,\mathrm{d}y = \mathrm{d}G(y) \quad , \tag{4.8.3}$$

or after integration,

$$x = G(y) = \int_{-\infty}^{y} g(t)\,\mathrm{d}t \quad . \tag{4.8.4}$$

This equation has the following meaning. If a random number x is taken from a uniform distribution between 0 and 1 and the function $\mathsf{x} = G(\mathsf{y})$ is inverted,

$$\mathsf{y} = G^{-1}(\mathsf{x}) \quad , \tag{4.8.5}$$

then one obtains a random number y described by the probability density $g(y)$. The relationship is depicted in Fig. 4.4a. The probability to obtain a random number x between x and $x + \mathrm{d}x$ is equal to the probability to have a value $\mathsf{y}(\mathsf{x})$ between y and $y + \mathrm{d}y$.

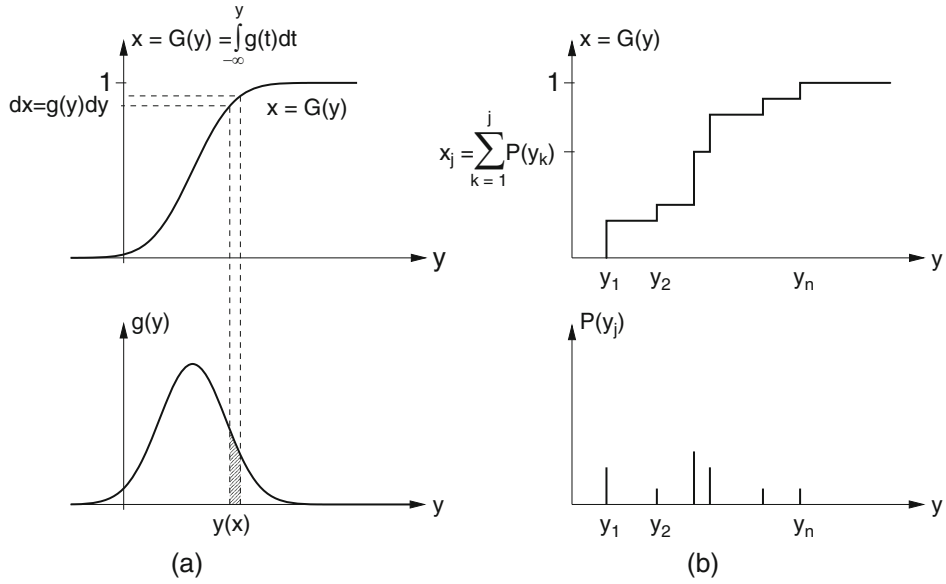

Fig. 4.4: Transformation from a uniformly distributed variable x to a variable y with the distribution function $G(y)$. The variable y can be continuous (**a**) or discrete (**b**).

The relationship (4.8.4) can be also be used to produce discrete probability distributions. An example is shown in Fig. 4.4b. The random variable y can take on the values $y_1, y_2, \ldots, y_n$ with the probabilities $P(y_1), P(y_2), \ldots, P(y_n)$. The distribution function as given by (3.2.1) is $G(y) = P(\mathsf{y} < y)$. The construction of a step function $x = G(y)$ according to this equation gives the values

$$x_j = G(y_j) = \sum_{k=1}^{i} P(y_k) \quad , \tag{4.8.6}$$

which lie in the range between 0 and 1. From this one can produce random numbers according to a discrete distribution $G(y)$ by first producing random numbers x uniformly distributed between 0 and 1. Depending on the interval j in which x falls, $x_{j-1} < \mathsf{x} < x_j$, the number y_j is then produced.

Example 4.1: Exponentially distributed random numbers

We would like to generate random numbers according to the probability density

$$g(t) = \begin{cases} \dfrac{1}{\tau} \mathrm{e}^{-t/\tau}\,, & t \geq 0 \quad , \\ \\ 0\,, & t < 0 \quad . \end{cases} \tag{4.8.7}$$

This is the probability density describing the time t of the decay of a radioactive nucleus that exists at time $t = 0$ and has a mean lifetime τ. The distribution function is

$$x = G(t) = \frac{1}{\tau} \int_{t'=0}^{t} g(t')\,\mathrm{d}t' = 1 - \mathrm{e}^{-t/\tau} \quad . \tag{4.8.8}$$

According to (4.8.4) and (4.8.5) we can obtain exponentially distributed random numbers t by first generating random numbers uniformly distributed between 0 and 1 and then finding the inverse function $\mathsf{t} = G^{-1}(\mathsf{x})$, i.e.,

$$\mathsf{t} = -\tau \ln(1 - \mathsf{x}) \quad .$$

Since $1 - \mathsf{x}$ is also uniformly distributed between 0 and 1, it is sufficient to compute

$$\mathsf{t} = -\tau \ln \mathsf{x} \quad . \blacksquare \tag{4.8.9}$$

Example 4.2: Generation of random numbers following a Breit–Wigner distribution

To generate random numbers y which follow a Breit–Wigner distribution (3.3.32),

$$g(y) = \frac{2}{\pi\,\Gamma} \frac{\Gamma^2}{4(y-a)^2 + \Gamma^2} \quad ,$$

we proceed as discussed in Sect. 4.8.1. We form the distribution function

$$x = G(y) = \int_{-\infty}^{y} g(y)\,\mathrm{d}y = \frac{2}{\pi\,\Gamma} \int_{-\infty}^{y} \frac{\Gamma^2}{4(y-a)^2 + \Gamma^2}\,\mathrm{d}y$$

and perform the integration using the substitution

$$u = \frac{2(y-a)}{\Gamma} \quad , \qquad \mathrm{d}u = \frac{2}{\Gamma}\,\mathrm{d}y \quad .$$

Thus we obtain

$$x = G(y) = \frac{1}{\pi}\int_{\theta=-\infty}^{\theta=2(y-a)/\Gamma} \frac{1}{1+u^2}\,\mathrm{d}u = \frac{1}{\pi}\left[\arctan u\right]_{-\infty}^{2(y-a)/\Gamma}$$
$$= \frac{\arctan 2(y-a)/\Gamma}{\pi} + \frac{1}{2} \quad .$$

By inversion we obtain

$$2(y-a)/\Gamma = \tan\left\{\pi\left(x-\frac{1}{2}\right)\right\} \quad ,$$
$$y = a + \frac{\Gamma}{2}\tan\left\{\pi\left(x-\frac{1}{2}\right)\right\} \quad . \qquad (4.8.10)$$

If x are random numbers uniformly distributed in the interval $0 < \mathsf{x} < 1$, then y follows a Breit–Wigner distribution. ■

Example 4.3: Generation of random numbers with a triangular distribution
In order to generate random numbers y following a triangular distribution as in Problem 3.2 we form the distribution function

$$F(y) = \begin{cases} 0\,, & y < a \quad , \\ \dfrac{(y-a)^2}{(b-a)(c-a)}\,, & a \le y < c \quad , \\ 1 - \dfrac{(y-b)^2}{(b-a)(b-c)}\,, & c \le y < b \quad , \\ 1\,, & b \le y \quad . \end{cases}$$

In particular we have

$$F(c) = \frac{c-a}{b-a} \quad .$$

Inverting $x = F(y)$ gives

$$y = a + \sqrt{(b-a)(c-a)x} \quad , \qquad x < (c-a)/(b-a) \quad ,$$
$$y = b - \sqrt{(b-a)(b-c)(1-x)} \quad , \qquad x \ge (c-a)/(b-a) \quad .$$

If x is uniformly distributed with $0 < \mathsf{x} < 1$, then y follows a triangular distribution. ■

4.8.2 Generation with the von Neumann Acceptance–Rejection Technique

The elegant technique of the previous section requires that the distribution function $x = G(y)$ be known and that the inverse function $y = G^{-1}(x)$ exists and be known as well.

Often one only knows the probability density $g(y)$. One can then use the VON NEUMANN acceptance–rejection technique, which we introduce with a simple example before discussing it in its general form.

Example 4.4: Semicircle distribution with the simple acceptance–rejection method

As a simple example we generate random numbers following a semicircular probability density,

$$g(y) = \begin{cases} (2/\pi R^2)\sqrt{R^2 - y^2}, & |y| \le R \\ 0, & |y| > R \end{cases} , \tag{4.8.11}$$

Instead of trying to find and invert the distribution function $G(y)$, we generate pairs of random numbers $(\mathsf{y}_i, \mathsf{u}_i)$. Here y_i is uniformly distributed in the interval available to y, $-R \le y \le R$, and u_i is uniformly distributed in the range of values assumed by the function $g(y)$, $0 \le u \le R$. For each pair we test if

$$\mathsf{u}_i \ge g(\mathsf{y}_i) \quad . \tag{4.8.12}$$

If this inequality is fulfilled, we reject the random number y_i. The set of random numbers y_i that are not rejected then follow a probability density $g(y)$, since each was accepted with a probability proportional to $g(\mathsf{y}_i)$. ■

The technique of Example 4.4 can easily be described geometrically. To generate random numbers in the interval $a \le y \le b$ according to the probability density $g(y)$, one must consider in the region $a \le y \le b$ the curve

$$u = g(y) \tag{4.8.13}$$

and a constant

$$u = d \quad , \qquad d \ge g_{\max} \quad , \tag{4.8.14}$$

which is greater than or equal to the maximum value of $g(y)$ in that region. In the (y, u) plane this constant is described by the line $u = d$. Pairs of random numbers $(\mathsf{y}_i, \mathsf{u}_i)$ uniformly distributed in the interval $a \le \mathsf{y}_i \le b$, $0 \le \mathsf{u}_i \le d$ correspond to a uniform distribution of points in the corresponding rectangle of the (y, u)-plane. If all of the points for which (4.8.12) holds are rejected, then only points under the curve $u = g(y)$ remain. Figure 4.5 shows this situation for the Example 4.4. [It is clear that the technique also gives meaningful results if the function is not normalized to one. In Fig. 4.5 we have simply set $g(y) = \sqrt{R^2 - y^2}$ and $R = 1$.]

For the transformation technique of Sect. 4.8.1, each random number y_i required only that exactly one random number x_i be generated from a uniform

distribution and that it be transformed according to (4.8.5). In the acceptance–rejection technique, pairs $\mathsf{y}_i, \mathsf{u}_i$ must always be generated, and a considerable fraction of the numbers y_i – depending on the value of u_i according to (4.8.12) – are rejected. The probability for y_i to be accepted is

$$E = \frac{\int_a^b g(y)\,\mathrm{d}y}{(b-a)d} \quad . \tag{4.8.15}$$

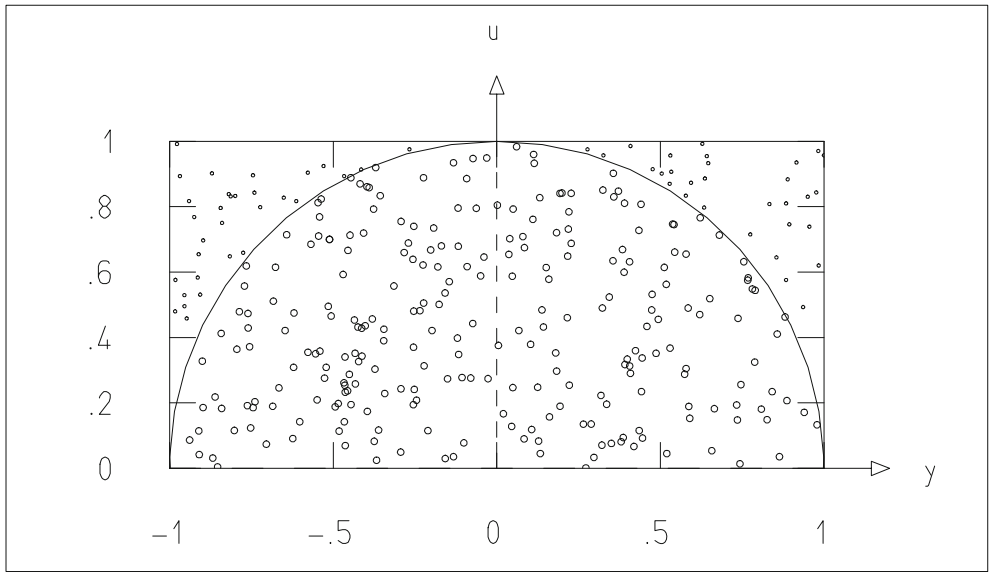

Fig. 4.5: All the pairs $(\mathsf{y}_i, \mathsf{u}_i)$ produced are marked as points in the (y, u)-plane. Points above the curve $u = g(y)$ (*small points*) are rejected.

We can call E the *efficiency* of the procedure. If the interval $a \le y \le b$ includes the entire allowed range of y, then the numerator of (4.8.15) is equal to unity, and one obtains

$$E = \frac{1}{(b-a)d} \quad . \tag{4.8.16}$$

The numerator and denominator of (4.8.15) are simply the areas contained in the region $a \le y \le b$ under the curves (4.8.13) and (4.8.14), respectively. One distributes points $(\mathsf{y}_i, \mathsf{u}_i)$ uniformly under the curve (4.8.14) and rejects the random numbers y_i if the inequality (4.8.12) holds. The efficiency of the procedure is certainly higher if one uses as the upper curve not the constant (4.8.14) but rather a curve that is closer to $g(y)$.

With this in mind the acceptance–rejection technique can be stated in its general form:

(i) One finds a probability density $s(y)$ that is sufficiently simple that random numbers can be generated from it using the transformation method, and a constant c such that

$$g(y) \le c \cdot s(y) \quad , \qquad a < y < b \quad , \tag{4.8.17}$$

holds.

(ii) One generates one random number y uniformly distributed in the interval $a < \mathsf{y} < b$ and a second random number u uniformly distributed in the interval $0 < \mathsf{u} < 1$.

(iii) One rejects y

$$\mathsf{u} \ge \frac{g(\mathsf{y})}{c \cdot s(\mathsf{y})} \quad . \tag{4.8.18}$$

After the points (ii) and (iii) have been repeated enough times, the resulting set of accepted random numbers y follows the probability density $g(y)$, since

$$P(\mathsf{y} < y) = \int_a^y s(t) \frac{g(t)}{c \cdot s(t)} \, \mathrm{d}t = \frac{1}{c} \int_a^y g(t) \, \mathrm{d}t = \frac{1}{c} [G(y) - G(a)] \quad .$$

If the interval $a \le y \le b$ includes the entire range of y for both $g(y)$ as well as for $s(y)$, then one obtains an efficiency

$$E = \frac{1}{c} \quad . \tag{4.8.19}$$

Example 4.5: Semicircle distribution with the general acceptance–rejection method

One chooses for $c \cdot s(y)$ the polygon

$$c \cdot s(y) = \begin{cases} 0 \, , & y < -R \quad , \\ 3R/2 + y \, , & -R \le y < -R/2 \quad , \\ R \, , & -R/2 \le y < R/2 \quad , \\ 3R/2 - y \, , & R/2 \le y < R \quad , \\ 0 \, , & R \le y \quad . \end{cases}$$

The efficiency is clearly

$$E = \frac{\pi R^2}{2} \cdot \frac{1}{2R^2 - R^2/4} = \frac{2\pi}{7}$$

in comparison to

$$E = \frac{\pi R^2}{2} \cdot \frac{1}{2R^2} = \frac{\pi}{4}$$

as in Example 4.4. ■

4.9 Generation of Normally Distributed Random Numbers

By far the most important distribution for data analysis is the normal distribution, which we will discuss in Sect. 5.7. We present here a program that can produce random numbers x_i following the standard normal distribution with the probability density

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2} \quad . \tag{4.9.1}$$

The corresponding distribution function $F(x)$ can only be computed and inverted numerically (Appendix C). Therefore the simple transformation method of Sect. 4.8.1 cannot be used. The *polar method* by BOX and MULLER [5] described here combines in an elegant way acceptance–rejection with transformation. The algorithm consists of the following steps:

(i) Generate two independent random numbers u_1, u_2 from a uniform distribution between 0 and 1. Transform $v_1 = 2u_1 - 1$, $v_2 = 2u_2 - 1$.

(ii) Compute $s = v_1^2 + v_2^2$.

(iii) If $s \geq 1$, return to step (i).

(iv) $x_1 = v_1\sqrt{-(2/s)\ln s}$ and $x_2 = v_2\sqrt{-(2/s)\ln s}$ are two independent random numbers following the standard normal distribution.

The number pairs (v_1, v_2) obtained from step (i) are the Cartesian coordinates of a set of points uniformly distributed inside the unit circle. We can write them as $v_1 = r\cos\theta$, $v_2 = r\sin\theta$ using the polar coordinates $r = \sqrt{s}$, $\theta = \arctan(v_2/v_1)$. The point (x_1, x_2) then has the Cartesian coordinates

$$x_1 = \cos\theta\sqrt{-2\ln s} \quad , \qquad x_2 = \sin\theta\sqrt{-2\ln s} \quad .$$

We now ask for the probability

$$\begin{aligned} F(r) &= P(\sqrt{-2\ln s} \leq r) = P(-2\ln s \leq r^2) \\ &= P(s > e^{-r^2/2}) \quad . \end{aligned}$$

Since $s = r^2$ is by construction uniformly distributed between 0 and 1, one has

$$F(r) = P(s > e^{-r^2/2}) = 1 - e^{-r^2/2} \quad .$$

The probability density of r is

$$f(r) = \frac{dF(r)}{dr} = re^{-r^2/2} \quad .$$

The joint distribution function of x_1 and x_2,

$$\begin{aligned} F(x_1,x_2) &= P(\mathsf{x}_1 \le x_1,\, \mathsf{x}_2 \le x_2) = P(\mathsf{r}\cos\theta \le x_1,\, \mathsf{r}\sin\theta \le x_2) \\ &= \frac{1}{2\pi}\int\int_{(\mathsf{x}_1<x_1,\mathsf{x}_2<x_2)} r\mathrm{e}^{-r^2/2}\,\mathrm{d}r\,\mathrm{d}\varphi \\ &= \frac{1}{2\pi}\int\int_{(\mathsf{x}_1<x_1,\mathsf{x}_2<x_2)} \mathrm{e}^{-(x_1^2+x_2^2)/2}\,\mathrm{d}x\,\mathrm{d}y \\ &= \left(\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x_1}\mathrm{e}^{-x_1^2/2}\,\mathrm{d}x_1\right)\left(\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x_2}\mathrm{e}^{-x_2^2/2}\,\mathrm{d}x_2\right) \quad , \end{aligned}$$

is the product of two distribution functions of the standard normal distribution. The procedure is implemented in the method DatanRandom.standard-Normal and illustrated in Fig. 4.6.

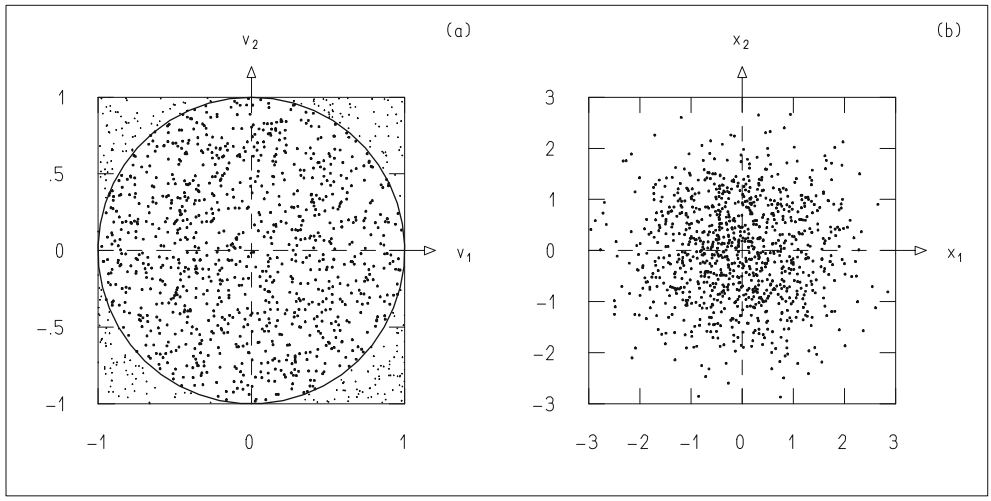

Fig. 4.6: Illustration of the Box–Muller procedure. (**a**) Number pairs $(\mathsf{v}_1, \mathsf{v}_2)$ are generated that uniformly populate the square. Those pairs are then rejected that do not lie inside the unit circle (marked by *small points*). (**b**) This is followed by the transformation $(\mathsf{v}_1, \mathsf{v}_2) \to (\mathsf{x}_1, \mathsf{x}_2)$.

Many other procedures are described in the literature for the generation of normally distributed random numbers. They are to a certain extent more efficient, but are generally more difficult to program than the BOX–MULLER procedure.

4.10 Generation of Random Numbers According to a Multivariate Normal Distribution

The probability density of a multivariate normal distribution of n variables $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ is according to (5.10.1)

$$\phi(\mathbf{x}) = k \exp\left\{-\frac{1}{2}(\mathbf{x}-\mathbf{a})^{\mathrm{T}} B(\mathbf{x}-\mathbf{a})\right\} \quad .$$

Here $\mathbf{a}$ is the vector of expectation values and $B = C^{-1}$ is the inverse of the positive-definite symmetric covariance matrix. With the Cholesky decomposition $B = D^{\mathrm{T}}D$ and the substitution $\mathbf{u} = D(\mathbf{x}-\mathbf{a})$ the exponent takes on the simple form

$$-\frac{1}{2}\mathbf{u}^{\mathrm{T}}\mathbf{u} = -\frac{1}{2}(u_1^2 + u_2^2 + \cdots + u_n^2) \quad .$$

Thus the elements u_i of the vectors $\mathbf{u}$ follow independent standard normal distributions [cf. (5.10.9)]. One obtains vectors $\mathbf{x}$ of random numbers by first forming a vector $\mathbf{u}$ of elements u_i which follow the standard normal distribution and then performing the transformation

$$\mathbf{x} = D^{-1}\mathbf{u} + \mathbf{a} \quad .$$

This procedure is implemented in the method DatanRandom.multivariate Normal.

4.11 The Monte Carlo Method for Integration

It follows directly from its construction that the acceptance–rejection technique, Sect. 4.8.2, provides a very simple method for numerical integration. If N pairs of random numbers $(\mathsf{y}_1, \mathsf{u}_i)$, $i = 1, 2, \ldots, N$ are generated according to the prescription of the general acceptance–rejection technique, and if $N - n$ of them are rejected because they fulfill condition (4.8.18), then the numbers N (or n) are proportional to the areas under the curves $c \cdot s(y)$ (or $g(y)$), at least in the limit of large N, i.e.,

$$\frac{\int_a^b g(y)\,\mathrm{d}y}{c\int_a^b s(y)\,\mathrm{d}y} = \lim_{N\to\infty}\frac{n}{N} \quad . \tag{4.11.1}$$

Since the function $s(y)$ is chosen to be particularly simple [in the simplest case one has $s(y) = 1/(b-a)$], the ratio n/N is a direct measure of the value of the integral

$$I = \int_a^b g(y)\,\mathrm{d}y = \left(\lim_{N\to\infty}\frac{n}{N}\right) c\int_a^b s(y)\,\mathrm{d}y \quad . \tag{4.11.2}$$

Here the integrand $g(y)$ does not necessarily have to be normalized, i.e., one does not need to require

$$\int_{-\infty}^{\infty} g(y)\,\mathrm{d}y = 1$$

as long as c is chosen such that (4.8.17) is fulfilled.

Example 4.6: Computation of π

Referring to Example 4.4 we compute the integral using (4.8.11) with $R = 1$:

$$I = \int_0^1 g(y)\,\mathrm{d}y = \pi/4 \quad .$$

Choosing $s(y) = 1$ and $c = 1$ we obtain

$$I = \lim_{N\to\infty} \frac{n}{N} \quad .$$

We expect that when N points are distributed according to a uniform distribution in the square $0 \le y \le 1$, $0 \le u \le 1$, and when n of them lie inside the unit circle, then the ratio n/N approaches the value $I = \pi/4$ in the limit $N \to \infty$. Table 4.2 shows the results for various values of n and for various sequences of random numbers. The exact value of n/N clearly depends on the particular sequence. In Sect. 6.8 we will determine that the typical fluctuations of the number n are approximately $\Delta n = \sqrt{n}$. Therefore one has for the relative precision for the determination of the integral (4.11.2)

$$\frac{\Delta I}{I} = \frac{\Delta n}{n} = \frac{1}{\sqrt{n}} \quad . \tag{4.11.3}$$

We expect therefore in the columns of Table 4.2 to find the value of π with precisions of 10, 1, and 0.1 %. We find in fact in the three columns fluctuations in the first, second, and third places after the decimal point. ■

Table 4.2: Numerical values of $4n/N$ for various values of n. The entries in the columns correspond to various sequences of random numbers.

$4n/N$		
$n = 10^2$	$n = 10^4$	$n = 10^6$
3.419	3.122	3.141
3.150	3.145	3.143
3.279	3.159	3.144
3.419	3.130	3.143

The Monte Carlo method of integration can now be implemented by a very simple program. For integration of single variable functions it is usually better to use other numerical techniques for reasons of computing time. For integrals with many variables, however, the Monte Carlo method is more straightforward and often faster as well.

4.12 The Monte Carlo Method for Simulation

Many real situations that are determined by statistical processes can be simulated in a computer with the aid of random numbers. Examples are automobile traffic in a given system of streets or the behavior of neutrons in a nuclear reactor. The Monte Carlo method was originally developed for the latter problem by VON NEUMANN and ULAM. A change of the parameters of the distributions corresponds then to a change in the actual situation. In this way the effect of additional streets or changes in the reactor can be investigated without having to undertake costly and time consuming changes in the real system. Not only processes of interest following statistical laws can be simulated with the Monte Carlo method, but also the measurement errors which occur in every measurement.

Example 4.7: Simulation of measurement errors of points on a line
We consider a line in the (t, y)-plane. It is described by the equation

$$y = at + b \quad . \tag{4.12.1}$$

If we choose discrete values of t

$$t_0 \quad , \qquad t_1 = t_0 + \Delta t \quad , \qquad t_2 = t_0 + 2\Delta t \quad , \qquad \ldots \quad , \tag{4.12.2}$$

then they correspond to values of y

$$y_i = at_i + b \quad , \qquad i = 0, 1, \ldots, n-1 \quad . \tag{4.12.3}$$

We assume that the values $t_0, t_1, \ldots$ of the "controlled variable" t can be set without error. Because of measurement errors, however, instead of the values y_i, one obtains different values

$$y_i' = y_i + \varepsilon_i \quad . \tag{4.12.4}$$

Here ε_i are the measurement errors, which follow a normal distribution with mean of zero and standard deviation σ_y (cf. Sect. 5.7). The method Datan-Random.line generates number pairs (t_i, y_i'). Figure 4.7 as an example displays 10 simulated points. ■

Example 4.8: Generation of decay times for a mixture of two different radioactive substances
At time $t = 0$ a source consists of N radioactive nuclei of which aN decay with a lifetime τ_1 and $(a-1)N$ with a mean lifetime τ_2, with $0 \le a \le 1$. Random numbers for two different problems must be used in the simulation the decay times occurring: for the choice of the type of nucleus and for the determination of the decay time of the nucleus chosen, cf. (4.8.9). The method DatanRandom.radio implements this example. ■

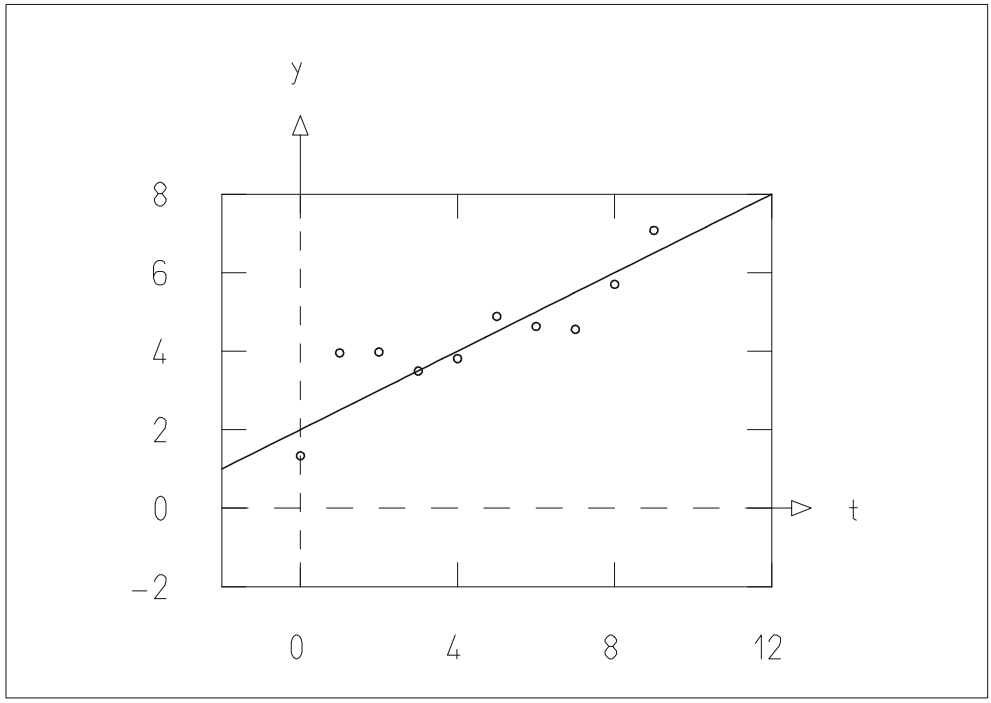

Fig. 4.7: Line in the (t, y)-plane and simulated measured values with errors in y.

4.13 Java Classes and Example Programs

Java Class for the Generation of Random Numbers

DatanRandom contains methods for the generation of random numbers following various distributions, in particular DatanRandom.ecuy for the uniform, DatanRandom.standard-Normal for the standard normal, and DatanRandom.multivariateNormal for the multivariate normal Distribution. Further methods are used to illustrate a simple MLC generator or to demonstrate the following examples.

Example Program 4.1: The class E1Random demonstrates the generation of random numbers

One can choose interactively between three generators. After clicking on Go 100 random numbers are generated and displayed. The seeds before and after generation are shown and can be changed interactively.

Example Program 4.2: The class E2Random demonstrates the generation of measurement points, scattering about a straight line

Example 4.7 is realized. Parameter input is interactive, output both numerical and graphical.

Example Program 4.3: The class E3Random demonstrates the simulation of decay times

Example 4.8 is realized. Parameter input is interactive, output in form of a histogram.

Example Program 4.4: The class E4Random demonstrates the generation of random numbers from a multivariate normal distribution

The procedure of Sect. 4.10 is realized for the case of two variables. Parameter input is interactive. The generated number pairs are displayed numerically.