Universiteit Utrecht

FACULTY OF SCIENCE

Scientific Computing

1st Report

Random Number Generators and Numerical Integration

Handed in	15/11/2011
Advisor	Bas Fagginger Auer
Student	Benjamin F. Maier (F110163)

Contents

Chapter 1

Random Number Generators

Random numbers play a rather important role in statistical sciences, for example for evaluating Markov chains, path integrals, particle simulations or the Monte Carlo evaluation of integrals (as explained in Ch.2). A device which produces random numbers is called a "random number generator" (RNG). There are certain types of RNGs, which can be parted in deterministic and non-deterministic RNGs.
For the need of reproduction of scientific calculations it is often helpful to have deterministic RNGs, since it becomes easier to compare results with previous calculations by using the same RNG with the same settings. In the following I will introduce you to a certain class of deterministic RNGs, called "linear congruential RNGs".

1.1 Draft of Mathematical Background

In the following I will summarize a few formulas, which are important for the methods described in this document. I have to emphasize that this is not a proper introduction and recommend [2] for readers who are unfamiliar with the topic.
Random variables are associated with a probability density function (pdf) $f(x)$, which fulfills the conditions $\lim_{x\to\pm\infty} f(x) = 0$, $\int_{-\infty}^{+\infty} \mathrm{d}x f(x) = 1$ and $f(x) \leq 0$. Furthermore, they are distributed with the cumulative distribution function $F(x)$, which fulfills $F(x) = \int_{-\infty}^{x} \mathrm{d}x' f(x')$, $\lim_{x\to-\infty} F(x) = 0$, $\lim_{x\to+\infty} F(x) = 1$, therefore $F(x) = f'(x)$.
The expectation value of X is then given as

$$E(X) = \int_{-\infty}^{+\infty} \mathrm{d}x\, x f(x), \tag{1.1}$$

associated with the variance

$$\mathrm{Var}(X) = E\Big((X - E(X))^2\Big). \tag{1.2}$$

1.2 Linear Congruential Random Number Generators

Linear congruential RNGs (in the following LRNGs or only RNGs) are random number generators, which use Alg. 1 to generate random numbers.
Respectively one defines the function $f(x) = (ax_{i-1} + c) \mod m$, sets the "seed" $x_0 = s$ and requests the i-th random number according to $x_i = f(x_{i-1})$.

Algorithm 1 Linear congruential random number generator

set $s \in \mathbb{N}$, $a, c, m \in \mathbb{N}$ and $x_0 = s$
to get N random numbers $x_i, i = 1, ..., N$:
for $i = 1 \to N$ **do**
  $x_i = (ax_{i-1} + c) \mod m$
end for

RNG	Parameters		
Park-Miller	$a = 16807$	$c = 0$	$m = 2^{31} - 1$
Noname	$a = 65539$	$c = 0$	$m = 2^{31} - 1$
Bad	$a = 5$	$c = 0$	$m = 2^7$
RANDU	$a = 65539$	$c = 0$	$m = 2^{31}$
Quick	$a = 1664525$	$c = 1013904223$	$m = 2^{32}$
Unix	$a = 1103515245$	$c = 12345$	$m = 2^{31}$

Table 1.1: Definition of certain LRNGs by the parameters a, c and m

The maximum number of such an RNG is $m - 1$ and the maximum period (after a sequence repeats itself) is m. One can map the integer random numbers x_i to be distributed in an interval $[a, b] \in \mathbb{R}$ by

$$y_i = \frac{x_i}{b - a} + a. \tag{1.3}$$

1.2.1 Certain RNGs

The first LRNGs I want to introduce are defined by certain values of a, c and m. Their definition is taken from [2] and can be seen in Tab. 1.1.

Gnu Standard Libary RNG

The Gnu Standard Libary (GSL) RNG is defined with $a = 1103515245$, $c = 12345$ and $m = 2^{31} = 2147483648$, [1]. It is famous for having bit correlations, which will be discussed later.

SUN RNG

The SUN RNG fixes the problem of the GSL RNG while keeping its period fixed. The algorithm obtains the random numbers by

$$x_{i+1} = ((a\tilde{x}_i + c) \gg 16) \,\&\, (2^{15} - 1),$$

where $\gg$ is the bitshift to right operator, a and c are taken from the GSL RNG and $\tilde{x}_i$ is obtained via $\tilde{x}_i = (a\tilde{x}_{i-1} + c) \mod 2^{32}$. In contrast to other RNGs, the seed for the next random number is not the previous output random number, but the number before the bitshift. This means that the random numbers are produced in the way of a linear congruential generator, but are modified before the output. The SUN RNG can therefore be described as a linear congruential generator with modified output.

Standard RNG (STD)

This a standard unix RNG used by the compiler g++. It is considered here to be comparised to the other RNGs.

Horribly Romantic Couple (HRC)

The HRC is a test generator defined by two other RNGS (in the following, I will combine mainly Park-Miller with GSL). It works, s.t. the seed of one generator is set by the last random number of the other RNG. The name was chosen because of the analogy of two people feeding each other. This procedure adds the maximum periods of the two chosen RNGs. However, it provides negative effects, too, which will be discussed later.

Overflow Problems

In case of too large values for the parameters, one has to be careful not to produce an "overflow" during the numerical evaluation. Since the datatypes in CPUs have an upper bound, a multiplication may yield a false result (s.t. it is larger than the upper bound - this would yield a periodically shifted result). Therefore one uses Schrage's trick. With

$$m = aq + r, \qquad q = \lfloor m/a \rfloor, \qquad r = m \mod a, \qquad and r < q, \tag{1.4}$$

the mod operation can be done according to

$$ax \mod m = a(x \mod q) - r\lfloor x \rfloor, \tag{1.5}$$

as stated in [2]. This has been done for the Park-Miller RNG.

1.2.2 Properties and Tests

The RNGs described in the previous section have been implemented and should be tested. A "good" RNG should provide the following attributes.

Large Period

A sequence of random numbers should not repeat itself too fast. A method to test this is to apply a two-dimensional binary walk, where one starts in the two-dimensional plane at $(0,0)$ and follows a path according to the random numbers. If a random number is in $[0,1/4)$ one goes one step up, if it is in $[1/4,1/2)$, one goes one step right, and so forth. After one period, the path will repeat itself.

Small Correlation

Even though a random number sequence $(x_i,...,x_N)$seems to be uncorrelated, one can discover correlations by plotting points in $[0,1]^2$ or $[0,1]^3$ according to $(x_1,x_2),(x_2,x_3),...$ respectively (x_1,x_2,x_2), $(x_2,x_3,x_4),....$ Correlations will result in vectors laying in hyperplanes.
Furthermore, random numbers can be correlated in their bit representation. A one-dimensional random walk can show periods in bits very efficiently. Get the random number in its bit representation. Now, for the n-th random walk, let us start at $(0,n)$. If the next n-th lowest bit is 1, the walker will move by (1,1/20), otherwise he will move by (1,-1/20).
For this, a function is needed, which delivers the last bits in an efficient way. Here, this was done with the bitwise AND-operator `&`. This operator works like a logical AND:

```
0 & 0 = 0
1 & 0 = 0
0 & 1 = 0
1 & 1 = 1
```

RNG	Time to get a Random Number in 10^{-5} s
SUN	4.76 ± 0.02
Quick	5.29 ± 0.03
GSL	5.48 ± 0.03
Unix	5.53 ± 0.03
RANDU	5.57 ± 0.02
Noname	5.75 ± 0.04
Park-Miller	6.26 ± 0.03
Bad	6.48 ± 0.03
STD	6.57 ± 0.03
HRC	9.74 ± 0.04

Table 1.2: Slopes of the curves in Fig. 1.1, respectively the time to get one random number with an Intel Core 2 Duo 2.10 GHz processor, listed from fast to slow. The slopes were obtained via linear fits.

By combining two numbers via the bitwise AND, every bit is combined with its corresponding partner of the other number. Take the following AND-operation as an example: 0b1100 & 0b1010 = 0b1000. Therefore, one can pick single bits of numbers by applying an AND-operation on the number and the corresponding power of 2 (every power 2 equals a binary number with only one 1). The n-th lowest bit of an integer i is obtained via the following algorithm.
Combine i bitwise with 2^{n-1}. If the result is equal to 2^{n-1}, the n-th lowest bit of i is 1. If the result is zero, the corresponding bit is 0. E.g. to pick the third lowest bit of $15 = 0b1111$, one can AND-operate it with $2^2 = 0b0100$, resulting in 0b0100.

Time

Requesting a sequence of random numbers should not take too long. Therefore it can be helpful to see a comparison of the time needed to get N random numbers. A time measuring has been done for 6 different N. For every number the time has been measured ten times and the average was evaluated.

Uniformity

For a sequence of N random numbers, we demand the numbers to be equally distributed on the interval $[a,b]$ they are defined in. One way to test this, is to draw a histogramm of the numbers. The bins' heights h_i should be nearly equal with $h_i = (b-a)/n$, where n is the number of bins. A sophisticated way to test this, is to perform a χ^2-test which measures the total deviation of the bins from their expected height according to

$$\chi^2 = \sum_{i=1}^{n} \frac{(h_i - E_i)^2}{E_i} = \frac{n}{b-a} \sum_{i=1}^{n} \left(h_i - \frac{b-a}{n} \right)^2 \tag{1.6}$$

$$= \frac{1}{n} \sum_{i=1}^{n} (n h_i - 1)^2, \tag{1.7}$$

where we will only concentrate on the interval $[0,1]$. According to [2], this quantity follows the χ^2 distribution, which means that if one calculates χ^2 for a number of random number series' and sets a confidence interval of the distribution, one is able to count the numbers of χ^2 exceeding this interval, and compare them with the theoretical probability of a χ^2 exceeding it (I recommend [2] for a deeper explanation).

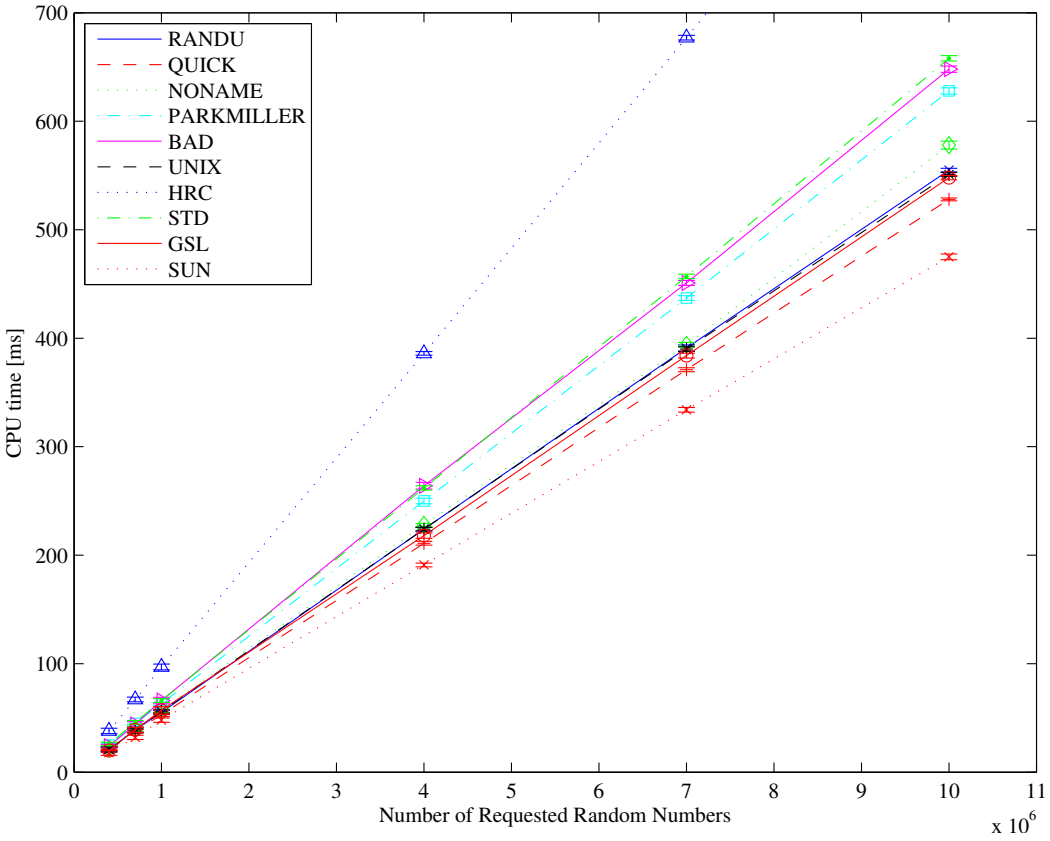

Figure 1.1: For certain values of N, the time to obtain that number of random number was measured (10 times) for a comparison of the RNGs' speed. A list of the slopes of the curves can be seen in Tab. 1.2

Therefore I will apply the following test. For each RNG one chooses 13 to 16 confidence intervals, produces 500 random number sequences, where one chooses N randomly from $\{10^4, 5\times10^4, 10^5, 3\times10^5, 8\times10^5\}$, bins the numbers to n bins (n randomly chosen from $\{10, 20, 30, 40, 50\}$) and calculates χ^2 for every series. One then counts the χ^2-values exceeding the confidence interval. If this number is lower or equal the expected number, the RNG passes the test for this confidence interval.

The results of the χ^2-tests can be seen in Tab. 1.3. As one can see, all generators except Noname and Bad pass at least a few of the χ^2-tests. Furthermore, in case of failure, the number of exceeding χ^2-values is near the border. That indicates an acceptable equal distribution of the random numbers.

The results of the period, uniformity and correlation tests are shown in App. B, Fig. B.1-Fig. B.10. As one can see, the Bad RNG lacks of everything. RANDU has correlations which can be seen from the 3D correlation plot, as well as the Noname RNG. UNIX and GSL show that they are implemented in the same way as they yield the same plots.

As one can see in Fig. B.11, the combination of GSL with SUN for the HRC RNG yields a rather bad RNG. Therefore one has to be careful with this one.

The test of the bit correlations is shown inApp. B, Fig. B.12-Fig. B.21. Bit correlations can be seen for RANDU , Quick , Bad , Unix , GSL and the last bit of HRC . Furthermore one notices the fix of the bit correlation of the SUN RNG.

The time measurement is shown in Fig. 1.1 and Tab. 1.2. Since the implementation of most of the RNGs does not really differ, there is no big difference in the time to get a random number. Small time differences (e.g. between Quick and SUN) may come from a slight difference in the implementation. Park-Miller , the RNG with Schrage's trick, shows a slower behavior which is probably caused by the additional calculations for r and q. The fastest RNG is SUN , the slowest is HRC, caused by the use of two RNGs.

RNG	Failures	In case of failure, numbers of exceeds is close to the boundary
RANDU	5 of 16	yes
Quick	11 of 16	yes
Noname	13 of 13	no
Park-Miller	11 of 13	yes
Bad	13 of 13	no
Unix	9 of 13	yes
HRC	10 of 13	yes
STD	7 of 13	yes
GSL	11 of 16	yes
SUN	9 of 16	yes

Table 1.3: Results of the χ^2-tests. The detailed results can be seen in App. A.

1.2.3 Conclusion

As one can see from the tests, the RNGs differ in their properties - therefore one always has to investigate a problem structure and needs before choosing an RNG to apply it on the problem. But as one could see from the investigations, the SUN RNG seems to provide rather good test results in all tests. Its only negative aspect is, that it only provides a range of 2^{15} random numbers, which may not be enough for certain problems.

Furthermore one should not use the HRC RNG, since it nearly doubles the time to obtain a random number and shows bad behavior for certain combinations.

1.3 Non-Uniform Random Numbers

Sometimes it might be useful to obtain random numbers which are not equally distributed but follow a certain distribution $g(x)$. There are two main methods to yield non-uniform random numbers from equally distributed random numbers.

1.3.1 Inversion Method

The first method is called inversion method. Let X be a uniformly distributed variable on $[0,1]$. Then the variable $Y = F^{-1}(X)$ is distributed with the cdf $F(X)$. For an explanation, see [2].

E.g. for a variable which is asked to be associated with a linear density function on [0,1], i.e. $f(x) = 2x$, one has to calculate the cdf

$$F(x) = \begin{cases} x^2, & x \in [0,1] \\ 1, & x > 1, \\ 0, & x < 1, \end{cases} \tag{1.8}$$

and to invert it, which gives

$$F^{-1}(y) = \begin{cases} \sqrt{y}, & y \in [0,1] \\ 0, & \text{other.} \end{cases} \tag{1.9}$$

An exponentially density function $f(x) = C\exp(-x)$ which is only defined in an interval $[a,b]$ would have to be normalized, s.t. $\int_a^b f(x)\mathrm{d}x = 1$, which means that $F(a) = 0, F(b) = 1$. Integration of f

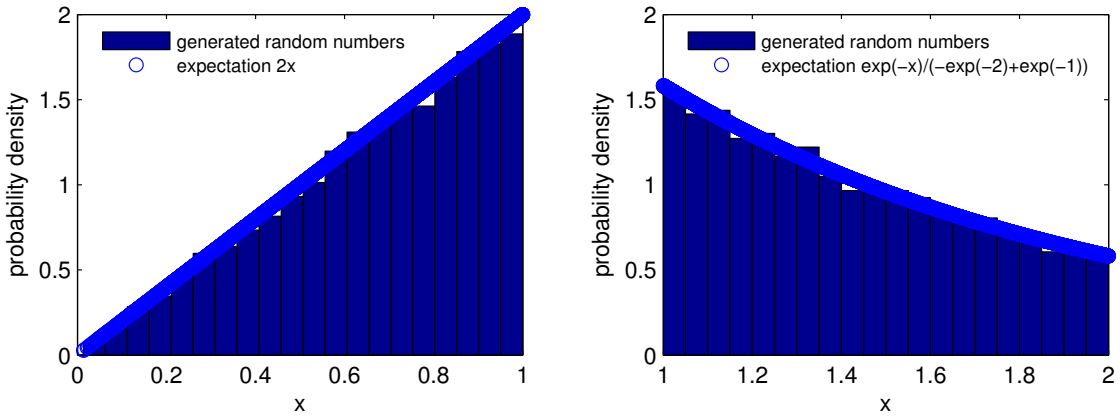

Figure 1.2: Inversion method: both described cases yield the expected behavior ($N = 10^4$).

yields

$$\int_a^x C\exp(-x')\mathrm{d}x' = C\left(-\exp(-x)+\exp(-a)\right)+D \overset{!}{=} F(x) \tag{1.10}$$

$$\Rightarrow \qquad F(x) = \frac{-\exp(-x)+\exp(-a)}{-\exp(-b)+\exp(-a)}, \tag{1.11}$$

respectively

$$F(x) = \begin{cases} \frac{-\exp(-x)+\exp(-a)}{-\exp(-b)+\exp(-a)}, & x \in [a,b] \\ 1, & x > b \\ 0, & x < a. \end{cases} \tag{1.12}$$

The inverse gives

$$F^{-1}(y) = \begin{cases} -\log\left(\exp(-a) - y(-\exp(-b)+\exp(-a))\right), & y \in [0,1] \\ 0, & \text{other.} \end{cases} \tag{1.13}$$

The method was tested for these cases, by generating $N = 10^4$ random numbers y_i and computing $x_i = F^{-1}(y_i)$. The positive results can be seen in Fig. 1.2.

1.3.2 Rejection Method

Another way to get a random variable with an arbitrary density function $p(x)$ on $[a,b]$ is the rejection method. Take two random variables X,Y which are uniformly distributed on [0,1]. In case $Y\ (\max_{[a,b]} p) \leq p(X)$, accept X, otherwise reject it. For a growing number of accepted random variables, these will have the density function $p(x)$. Note that because auf the arbitrariness $p(X)$ is then not anymore a pdf and therefore the result of the outcoming density function is only right with a scaling factor.

The method has been tested for the functions $f_1(x) = 3x^2$, $f_2(x) = e^{-x^2}$, $f_3(x) = exp(-x)$, $f_4(x) = 1/(1+x^2)$, where the positive result can be seen in Fig. 1.3.

A More Sophisticated Rejection Method

In case the density function is concentrated in a specific region and small in others, it may reduce the number of rejections when one does the comparison between function and random number in a region

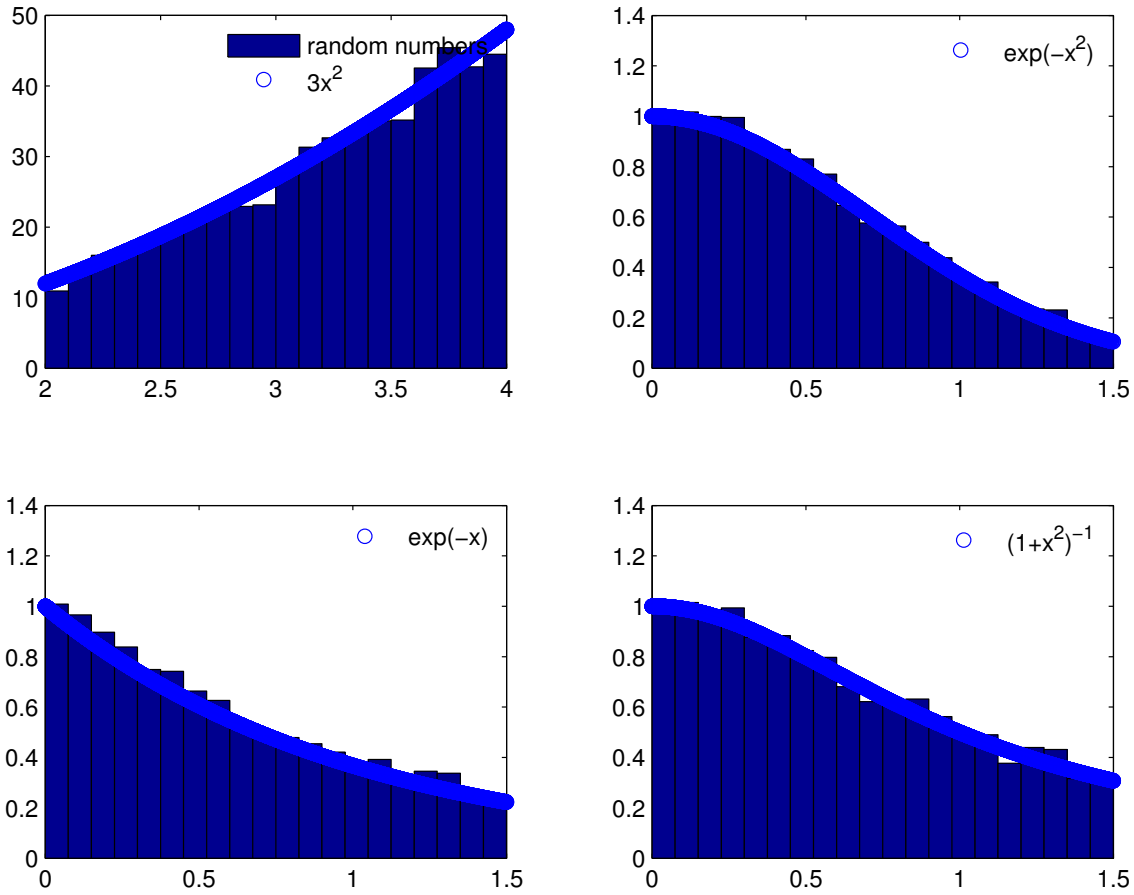

Figure 1.3: Rejection method: All cases yield the expected behavior modulo a constant ($N = 10^4$).

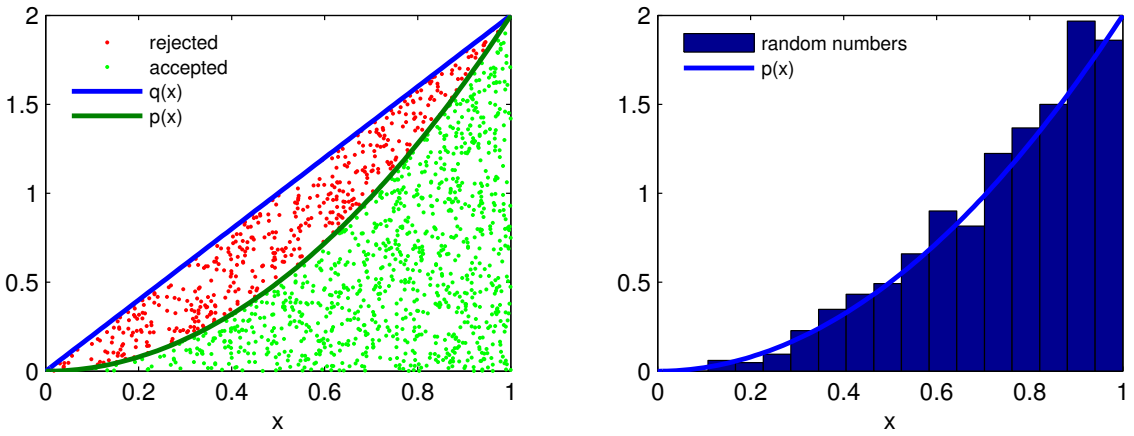

Figure 1.4: Rejection method: A better choice of the density function $q(x)$ yields less rejections and therefore an higher efficiency ($N = 10^3$).

which is bounded by another density function. Imagine the function $f(x) = 2x^2$. A sophisticated choice of a function q would be $q(x) = 2x$. To get random numbers according to that density one can use the inversion method with $F^{-1}(y) = \sqrt{y}$. As one can see in Fig. 1.4, this procedure leads to less rejections than an equally distributed random variable, whereas it produces random variables with a density function proportional to $f(x)$. It is therefore a more efficient procedure than the normal rejection method.

Chapter 2

Numerical Integration

Since analytical solutions to integrals are not always providable, but necessary for research, one needs efficient numerical methods with known convergence behavior. In the following sections I will introduce you to two methods to integrate numerically in arbitrary dimensions.

2.1 Trapezoidal Integration

2.1.1 One-Dimensional Domain

A rather simple approach to integrate a function $f : [a,b] \subset \mathbb{R} \to \mathbb{R}$

$$I = \int_a^b \mathrm{d}x\, f(x) \tag{2.1}$$

in a numerical way is that of trapezoidal integration. It approximates f by a linear interpolation according to

$$p(x) = \frac{(x-b)}{a-b} f(a) + \frac{(x-a)}{b-a} f(b), \tag{2.2}$$

which integrates to

$$I_p = \int_a^b \mathrm{d}x p(x) = \frac{f(b)+f(a)}{2}\,(b-a), \tag{2.3}$$

basically the geometrical formula for the area of a trapezoid. The remainder R of this method is given by

$$R = I - I_p = \int_a^b \mathrm{d}x\,(f(x)-p(x)) \tag{2.4}$$

$$= \int_a^b \mathrm{d}x (x-a)(x-b)\frac{f(x)-p(x)}{(x-a)(x-b)}. \tag{2.5}$$

One now defines a function

$$g(w) := f(w) - p(w) - (w-a)(w-b)\frac{f(x)-p(x)}{(x-a)(x-b)}, \qquad w \in [a,b], \tag{2.6}$$

which has three zeros (at a, b and x). With f being continuous in $[a,b]$ and twice differentiable in $[a,b]$, one can apply Rolle's theorem, with the result of $g'(w)$ having two zeros. Applying Rolle's theorem again yields exactly one zero for $g''(w)$, and therefore

$$g''(\xi(x)) = 0 \tag{2.7}$$

$$0 = f''(\xi(x)) - 2\frac{f(x)-p(x)}{(x-a)(x-b)} \tag{2.8}$$

$$\Rightarrow \qquad \frac{f(x)-p(x)}{(x-a)(x-b)} = \frac{1}{2}\, f''(\xi(x)). \tag{2.9}$$

With $f''(\xi(x))$ being continuous in $[a,b]$ one can use the intermediate value theorem to obtain

$$R = \frac{f''(\eta)}{2} \int_a^b \mathrm{d}x (x-a)(x-b) \tag{2.10}$$

$$= \frac{f''(\eta)}{12}\,(a-b)^3 \tag{2.11}$$

for an $\eta \in [a,b]$.

Advanced Trapezoidal Method

The trapezoidal rule becomes more exact for cutting the interval $[a,b]$ in N intervals of length $h = (b-a)/N$ and applying the trapezoidal rule for every interval $[x_i, x_i+1]$, with $x_0 = a$ and $x_i = x_0 + h \times i$, yielding the approximated integral

$$\tilde{I} = \frac{h}{2}\left[f(x_0) + f(x_N)\right] + h\,\left[\sum_{i=1}^{N-1} f(x_i)\right]. \tag{2.12}$$

The remainder for every interval is

$$R(h) = -\frac{1}{12}h^3 \sum_{i=0}^{N-1} f''(\eta_i), \qquad \eta_i \in [x_i, x_{i+1}]. \tag{2.13}$$

Using the triangle inequality yields

$$|R(h)| \leq \frac{h^3}{12} \sum_{i=0}^{N-1} \left|f''(\eta_i)\right| \tag{2.14}$$

For $h \to 0$, one can rewrite the sum as an integral and use the intermediate value theorem to obtain

$$\frac{h^3}{12} \sum_{i=0}^{N-1} \left|f''(\eta_i)\right| = \frac{h^2}{12} \int_a^b \mathrm{d}x \left|f''(\xi(x))\right| \tag{2.15}$$

$$= \frac{h^2}{12}\,\left|f''(\eta)\right| \int_a^b \mathrm{d}x \tag{2.16}$$

$$= \frac{(b-a)}{12}\, h^2\, \left|f''(\eta)\right|, \qquad \eta \in [a,b] \tag{2.17}$$

$$\Rightarrow \qquad |R(h)| \leq \frac{(b-a)}{12}\, h^2\, \max_{[a,b]} \left|f''(\eta)\right|. \tag{2.18}$$

The absolute error of this method is therefore $O(h^2)$, respectively $O(1/N^2)$. In general, for a domain of dimension d, one needs $N \propto h^{-d}$ discretization points, therefore an error of $O(h^2)$ yields an error of $O\left(1/N^{2/d}\right)$.

2.1.2 Arbitrary-Dimensional Domains

From Trapezoids to Rectangles to a Weighted Mean

From geometrical reasons it should be obvious that the trapezoidal integration is equal to a sum where the trapezoids are replaced by rectangles. These rectangles have the height of the mean of the adjacent function values, therefore the area of the i-th rectangle is

$$A_i = \frac{h}{2}\Big(f(x_i) + f(x_{i+1})\Big),$$

where again $h = (b-a)/N$. This is equal to the Area of the i-th trapezoid. Therefore, a sum over all N rectangles yields the same result as a sum over all trapezoids.

$$I\big(f,\{x\}\big) = \frac{h}{2}\sum_{i=1}^{N}\Big(f(x_i) + f(x_{i+1})\Big)$$

Furthermore, one can reinterpret this sum as a weighted mean over all values of f for the set $\{x_1, x_2, ..., x_N\}$. It is obvious that every value $f(x_i)$ which has two adjacent rectangles is associated with a weight of 2 (since it appears twice in the overall rectangle sum). The two "end values" $f(x_1)$ and $f(x_N)$ have only one adjacent rectangle and therefore are associated with the weight 1. For convience, let us define $\mathcal{A}_i$ to be number of adjacent rectangles for $f(\mathbf{x}_i)$. How to obtain this number will be discussed later. Now, the approximated integral reduces to

$$I\big(f,\{x\}\big) = \frac{h}{2}\sum_{i=1}^{N} 2^{\mathcal{A}_i - 1} f(\mathbf{x}_i).$$

One can wonder if this approach of a weighted mean is applicable for arbitrary dimensions. The answer is yes, which I will proof detailed in the following for the case $f : \mathbb{R}^2 \to \mathbb{R}$. Later, I will sketch a proof for even higher dimensions, which is not finished. However, later numerical results show agreement with the assumption.

Trapezoidal Integration in Two Dimensions

The task is to evaluate the volume under a function $f(\mathbf{x})$ in a twodimensional interval $[\mathbf{a},\mathbf{b}] \equiv [a_x, b_x] \times [a_y, b_y]$ (a rectangle). For an easier treatment in the first place, I consider the interval to be of the same length in every direction of space, to make the rectangle a square and to get an h independent of the direction of space (this can easily be transformed back by scaling h for the different dimensions).
At first, one defines a twodimensional lattice on the interval by dividing $(b_l - a_l)/N$ in both directions $l = 1, 2$ to get $(N+1)^2$ twodimensional vectors $\{\mathbf{x}_0, ..., \mathbf{x}_{(N+1)^2-1}\}$. Now the hypothesis is that one is able to evaluate this integral numerically by considering a sum over cuboids, where the height of the cuboids is gained via the mean of all adjacent f-values. That would yield

$$I = \frac{h^2}{4}\sum_{i=1}^{N}\sum_{j=1}^{N}\Big(f(x_i, y_j) + f(x_{i+1}, y_j) + f(x_i, y_{j+1}) + f(x_{i+1}, y_{j+1})\Big),$$

or, since one would prefer a sum over all x-values due to an easier numerical treatment, the weighted mean

$$I = \frac{h^2}{4}\sum_{i=0}^{(N+1)^2-1} 2^{\mathcal{A}_i - 1} f(\mathbf{x}_i),$$

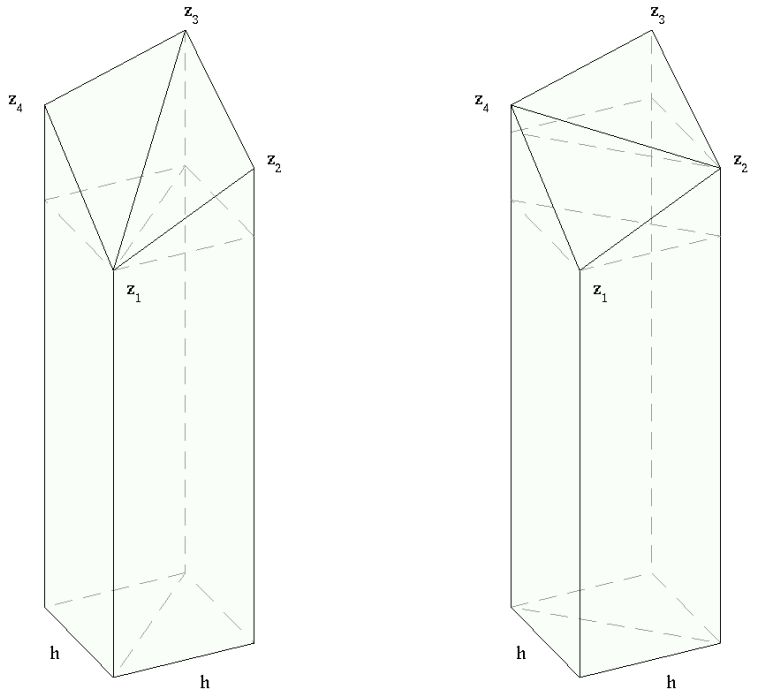

Figure 2.1: For every square of area h^2, one can approximate the surface of f in these two different ways (equal to cutting the square in two triangles in two different ways.) The volume of the three-dimensional trapezoid is obtained by cutting it in a prism and a pyramid.

where $\mathcal{A}_i$ is the number of adjacent cuboids at $\mathbf{x}_i$. But on the first sight, this might not be a valid approach to approximate f.

A valid approximation of the function's surface over $[\mathbf{a},\mathbf{b}]$ is the one with three-dimensional trapezoids (a linear interpolation in three dimensions). Applying a sum over all squares in $[\mathbf{a},\mathbf{b}]$, one recognizes that for every square, it is possible to approximate the surface in two different ways (c.f. Fig. 2.1).

To take the example in this figure, the volume of the three-dimensional trapezoid can be evaluated cutting it in a prism with basal area $h^2/2$ and height z_1 (which is the lowest value of f in the current triangle) and a pyramid. The top of the pyramid is always at the lowest f-value. Now let us calculate the volume of one three-dimensional trapezoid.

Case 1) The base of the pyramid has the length h and the height h

In this case, the volume of the pyramid is given by $V = Gh/3$, where G is the basal area of the pyramid (a two-dimensional trapezoid). Therefore the volume of the three-dimensional trapezoid can be calculated as follows (c.f. to Fig. 2.1, left).

$$V_{\text{pyramid}} = \frac{1}{3}Gh \;=\; \frac{1}{3}\left(\frac{1}{2}(z_3 - z_1 + z_4 - z_1)\,h\right) h \;=\; \frac{h^2}{6}\,(z_3 + z_4 - 2z_1)$$

$$V_{\text{prism}} = \frac{h^2}{2}\,z_1$$

$$\begin{aligned} V &= V_{\text{pyramid}} + V_{\text{prisma}} \\ &= \frac{h^2}{6}\,(z_1 + z_3 + z_4) \end{aligned}$$

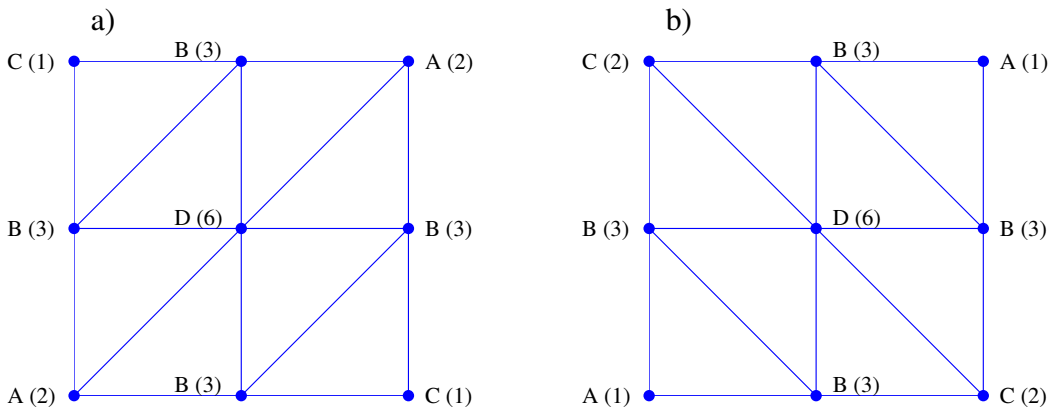

Figure 2.2: Possible choices of triangle orientation for a 2D-lattice. Displayed are the kinds of points marked with a capital letter and with the number of adjacent triangles in paranthesis. A and C do not differ in their number of adjacent squares, but their number of adjacent triangles. **a)** First choice of orientation. **b)** Second choice of orientation. Note that A and C switched their number of adjacent triangles, while the number stayed constant for B and D.

Case 2) The base of the pyramid has the length $h\sqrt{2}$ and the height $h/\sqrt{2}$
This equals the case in Fig. 2.1, right. Here, the volume of the pyramid is

$$V_{\text{Pyramid}} = \frac{1}{3} G \frac{h}{\sqrt{2}} = \frac{1}{3}\left(\frac{1}{2}(z_3 - z_1 + z_2 - z_1)\, h\sqrt{2}\right)\frac{h}{\sqrt{2}} = \frac{h^2}{6}\,(z_3 + z_2 - 2z_1)\,,$$

yielding the volume of the three-dimensional trapezoid

$$\begin{aligned} V &= V_{\text{Pyramid}} + V_{\text{prism}} \\ &= \frac{h^2}{6}\,(z_1 + z_3 + z_2)\,. \end{aligned}$$

As one can see, the volume of one trapezoid is a mean over all adjacent f-values, multiplied with the base area $h^2/2$. The integral of f over the interval $[\mathbf{a},\mathbf{b}]$ can therefore be approximated by a sum over all trapezoids, respectively a weighted sum over all f-values, where the weights are obviously the number of adjacent trapeziods. Unfortunately, the sum depends on the choice of how to cut a square in two triangles, as can be seen in Fig. 2.2.

If one chooses the triangle in direction **a)**, every point with three adjacent triangles is weighted with a 3, every point in the middle of the field is weighted with a 6, the two points $(0,0)$, (N,N) are weighted with a 2 and the two points $(0,N)$, (N,N) are weighted with a 1.

If one chooses the triangle in direction **b)**, every point with three adjacent triangles is weighted with a 3, every point in the middle of the field is weighted with a 6, the two points $(0,0)$, (N,N) are weighted with a 1 and the two points $(0,N)$, (N,N) are weighted with a 2.

A possibility to get rid of the orientation problem, is to build the mean over both orientations, which means summing over every point twice, summing up the weights for every chosen orientation and divide by two afterwards. This yields that every point with three triangles (on twodimensional border of field) is weighted with a 6, every point in the middle of the field is weighted with a 12, the two points $(0,0)$, (N,N) are weighted with a 3 and the two points $(0,N)$, (N,N) are weighted with a 3, as well.

Therefore the weights are $3 \times 2^{\#(\text{adjacent squares})-1}$, and the approximated integral is

$$I = \frac{h^2}{6}\frac{1}{2}3 \sum_{i=1}^{(N+1)^2} 2^{\mathcal{A}_i-1} f(\mathbf{x}_i) \tag{2.19}$$

$$= \frac{h^2}{4} \sum_{i=1}^{(N+1)^2} 2^{\mathcal{A}_i-1} f(\mathbf{x}_i), \tag{2.20}$$

which equals the hypothetic result of evaluating the volume via a mean over cuboids.

Extrapolation to Arbitrary Dimensions

One can now assume that it is always possible to calculate an integral of a function f over an interval $[\mathbf{a},\mathbf{b}]$ of dimension d via adding hypercuboids, where the basal volume of the hypercuboid is a hypercube of dimension d and its height is the mean of all adjacent values of f. This would lead to an integral approximation

$$I_d\big(f,\{\mathbf{x}\}\big) = \frac{h^2}{2^d} \sum_{i=0}^{(N+1)^d-1} 2^{\mathcal{A}_i-1} f(\mathbf{x}_i), \tag{2.21}$$

where $\mathcal{A}_i$ is now the number of adjacent hypercuboids of $\mathbf{x}_i$. Let us sketch a proof of this formula.
First, consider the basal hypercube with side length h to be cut in hypertriangles with the volume

$$V_\Omega = \int_\Omega \mathrm{d}V = \int_0^h \mathrm{d}x_1 \prod_{i=2}^{d} \int_0^{x_{i-1}} \mathrm{d}x_i = \frac{h^d}{d!}. \tag{2.22}$$

For the case $d = 1$, this equals the rectangle side length h, for $d = 2$ it equals one triangle with area $h^2/2$. Note that this also means, that we can cut a hypercube of dimension d in $d!$ hypertriangles of the same form.
We can cut the entire interval in hypertriangles of the form in Eq. (2.22). Note that an integral of f over such a hypertriangle is invariant under translations, rotations and reflections in $\mathbb{R}^{d+1}$. Therefore we can apply a coordinate transformation (with f becoming a transformed function $\hat{f}$), s.t. every integral over an hypertriangle in $[\mathbf{a},\mathbf{b}]$ may be reduced to

$$V = \int_0^h \mathrm{d}x_1 \prod_{i=2}^{d} \int_0^{x_{i-1}} \mathrm{d}x_i \; \hat{f}(\mathbf{x}). \tag{2.23}$$

Now, let us set a linear interpolation

$$p(\mathbf{x}) = \sum_{i=1}^{d} \alpha_i x_i + c, \qquad \alpha_i, c \in \mathbb{R} \tag{2.24}$$

of $\hat{f}$ over the area Ω. Hence, the coefficients α_i, c have to be chosen, s.t. p equals $\hat{f}$ at the d-dimensional points $(0,0,...,0),(h,0,...,0),(h,h,...,0),...,(h,h,...,h)$, $d+1$ points in total.
Now, the approximated volume under the function p over Ω is given by

$$\tilde{V} = \int_0^h \mathrm{d}x_1 \prod_{i=2}^{d} \int_0^{x_{i-1}} \mathrm{d}x_i \; p(\mathbf{x}). \tag{2.25}$$

A formula to help solving this integral is

$$\prod_{i=2}^{d} \int_0^{x_{i-1}} \mathrm{d}x_i \; x_d^p \left(\sum_{j=1}^{d-1} \alpha_j x_j \right) = \frac{x_1^{p+d}}{(p+d)!} \, p! \sum_{j=1}^{d-1} \alpha_j \; (p+1+d-j). \tag{2.26}$$

for $p \in \mathbb{N}$. Its derivation is given in App. C. The volume can now be evaluated to

$$\tilde{V} = \int_0^h \mathrm{d}x_1 \prod_{i=2}^{d} \int_0^{x_{i-1}} \mathrm{d}x_i \; \left(\sum_{j=1}^{d} \alpha_j x_j + c \right) \tag{2.27}$$

$$= c \, \frac{h^d}{d!} \; + \int_0^h \mathrm{d}x_1 \prod_{i=2}^{d} \int_0^{x_{i-1}} \mathrm{d}x_i \; (\alpha_d x_d) \; + \int_0^h \mathrm{d}x_1 \prod_{i=2}^{d} \int_0^{x_{i-1}} \mathrm{d}x_i \; \left(\sum_{j=1}^{d-1} \alpha_j x_j + c \right) \tag{2.28}$$

$$\overset{(2.26),p=0}{=} \; c \, \frac{h^d}{d!} \; + \; \frac{h^{d+1} \, \alpha_d}{(d+1)!} \; + \int_0^h \mathrm{d}x_1 \frac{x_1^d}{d!} \sum_{j=1}^{d-1} \alpha_j \; (d-j+1) \tag{2.29}$$

$$= c \, \frac{h^d}{d!} + \frac{h^{d+1}}{(d+1)!} \sum_{j=1}^{d} \alpha_j \; (d-j+1). \tag{2.30}$$

For comparison, we can calculate the volume of a hyperprism where its basal area is Ω and its height is the mean of all adjacent $\hat{f}$-values.

$$V_{\text{prism}} = \frac{h^d}{d!} \; \frac{1}{d+1} \Big(\hat{f}(0,0,...0) + \hat{f}(h,0,...0) + \hat{f}(h,h,...0) + ... + \hat{f}(h,h,...h) \Big). \tag{2.31}$$

Remember, that the requirement for the linear interpolation p was that it equals $\hat{f}$ at these points. Therefore we can replace $\hat{f}$ with p in this equation and obtain

$$V_{\text{prism}} = \frac{h^d}{(d+1)!} \; \Big((d+1)c + dh\alpha_1 + (d-1)h\alpha_2 + ... + h\alpha_d \Big) \tag{2.32}$$

$$= c \, \frac{h^d}{d!} + \frac{h^{d+1}}{(d+1)!} \sum_{j=1}^{d} \alpha_j \; (d-j+1) \tag{2.33}$$

$$= \tilde{V}. \tag{2.34}$$

Therefore, we can calculate the integral of p over Ω as a hyperprism, where its height is the mean of all adjacent $\hat{f}$-values. Unfortunately, I am not able to proof that a following summation of all hyperprisms and obtaining the mean of all possible orientation yields Eq. (2.21). However, since later numerical results show agreement with Eq. (2.21), I will continue with its numerical evaluation.

Numerical Treatment

For reasons of performance and technical simplicity, I recommend to perform a weighted sum over all possible $\mathbf{x}$-values instead of performing a sum over possible hypercuboids. By performing a sum over hypercuboids, the value of f at $\mathbf{x}_i$ would have to be computed $2^{\mathcal{A}_i)-1}$ times, which reduces performance time. Furthermore, one would intuitively want to perform d sums, one for every dimension, which is not possible to implement with d as a free parameter.

However, one may wonder how to get the number of adjacent hypercubes while performing a single sum over all possible $\mathbf{x}$-values. This number is obtained via the fact that every linear index l for

Algorithm 2 Obtaining a d-dimensional index vector $\mathbf{m}$

Require: l is a linear index in $[0,(N+1)^2-1]$
  $\mathbf{m} = 0$
  for $k = d \to 1$ **do**
    $m_k \leftarrow l \mod (N+1)$
    $l \leftarrow (l-m_k)/(N+1)$
  end for

$(N+1)^2$ can be mapped to a d-dimensional index vector and vice versa. An index vector $\mathbf{m} \in \mathbb{N}^d$ can be obtained via Alg. 2.

Now one can count the occurences of N or 0 in the indice vector. Every occurence of N or 0 decrements the dimensionality of the index vector $\mathbf{m}$ by 1 (since it is located at a border of the hyperfield). The number of adjacent hypercuboids will be the dimensionality($\mathbf{m}$)+1.

2.2 Monte Carlo Integration

The method described in the following uses random numbers to evaluate integrals. It will be shown that for multi-dimensional integrals, this method is more efficent than trapezoidal integration. This chapter is mainly based on the explanations and derivations in [2], where everything is explained in a detailed and sufficient way. Therefore, I will summarize the most important techniques and formulas in the following, while I recommend the studying of [2] for a deeper understanding.

2.2.1 Methods for One-Dimensional Domain

Hit-or-Miss Monte Carlo

Imagine a function $f : [0,1] \subset \mathbb{R} \to [0,1] \subset \mathbb{R}^+$ to be integrated to

$$E(f) = \int_0^1 \mathrm{d}x\, f(x). \tag{2.35}$$

Now, define the sets $S = [0,1] \times [0,1]$ and $A = \{(x,y) : (x,y) \in S \wedge y \leq f(x)\}$, which means that the area of A equals $E(f)$. Suppose two uniformly distributed random variable (X,Y) on S, then the area of A is given by

$$P((X,Y) \in A). \tag{2.36}$$

A numerical evaluation of P can be done by generating N two-dimensional random number vectors $(x_1,y_1),...,(x_N,y_N)$, setting $N_A = |\{i : i \in [1,N] \wedge y_i \leq f(x_i)\}|$ and building the ratio

$$\frac{N_A}{N}, \qquad \Rightarrow \lim_{N\to\infty} \frac{N_A}{N} = P((X,Y) \in A). \tag{2.37}$$

Note that the probability is always positive, hence functions with an image of $\mathbb{R}^-$ will not be calculated in the right way.

Since the algorithm is defined by hitting or missing, N_A is distributed according to the binomial distribution with the variance $NP(1-P)$. Hence the error of N_A is $\propto \sqrt{N}$ and $E(f) = P = N_A/N$ is associated with an error $\propto \frac{1}{\sqrt{N}}$.

To integrate a function $f : [a,b] \subset \mathbb{R} \to [w,z] \subset \mathbb{R}$, one can rescale/transform it to reduce it to the former case.

Simple Sampling

Another possibility to calculate an integral

$$I = \int_a^b \mathrm{d}x\, f(x) \tag{2.38}$$

is the one with density functions. Here, the image of f can be $\mathbb{R}$ Suppose x to be a random variable equally distributed on $[a,b]$ and $p(x)$ the associated density function

$$p(x) = \begin{cases} 1/(b-a), & x \in [a,b] \\ 0, & \text{other} \end{cases}, \tag{2.39}$$

s.t. for the expectation value follows

$$E(x) = \int_{-\infty}^{+\infty} \mathrm{d}x\, x\, p(x) \tag{2.40}$$

$$E(f(x)) = \int_{-\infty}^{+\infty} \mathrm{d}x\, f(x)\, p(x) = \frac{1}{b-a} \int_a^b \mathrm{d}x f(x) \tag{2.41}$$

$$\Rightarrow\ I = (b-a)E(f(x)). \tag{2.42}$$

With large N and because x is uniformly distributed, the expectation value can be estimated by generating N random numbers x_i and evaluating the mean of the function

$$E(f(x)) \approx \frac{1}{N} \sum_{i=1}^{N} f(x_i). \tag{2.43}$$

Following [2], the error of the method is $\propto \frac{1}{\sqrt{N}}$.

Importance Sampling

In case f has rather large values for a certain region in $[a,b]$, a uniformly distributed random variable will lead to a bad result when evaluating the mean. Therefore it might be helpful to introduce a pdf, which is an approximation of f, s.t.

$$I = \int_a^b \mathrm{d}x\, \frac{f(x)}{g(x)}\, g(x). \tag{2.44}$$

Now, $\frac{f(x)}{g(x)}$ is approximately equally distributed, s.t. building the mean with a random variable X with pdf $g(x)$

$$E(f) \approx \frac{1}{N} \sum_{i=1}^{N} X_i \frac{f(X_i)}{g(X_i)} \tag{2.45}$$

yields a numerical more stable result.

2.2.2 Arbitrary-Dimensional Domain

Hit-or-Miss Monte Carlo

The domain has now the dimensionality $d \in \mathbb{N}$. Imagine a function $f : [\mathbf{a},\mathbf{b}] \to \mathbb{R}^+$, where $[\mathbf{a},\mathbf{b}] \equiv [a_1,b_1] \times [a_2,b_2] \times ... \times [a_d,b_d]$. The task is to compute

$$I = \int_{a_1}^{b_1} \mathrm{d}x_1 \int_{a_2}^{b_2} \mathrm{d}x_2 \; ... \int_{a_d}^{b_d} \mathrm{d}x_d \; f(\mathbf{x}). \tag{2.46}$$

We can again rescale the domain $\mathbf{x} \in [\mathbf{a},\mathbf{b}]$ to $\hat{x}_i = \frac{x_i - a_i}{b_i - a_i}$, and $\hat{f}(\mathbf{x}) = \frac{f(\mathbf{x})}{\max_{[\mathbf{a},\mathbf{b}]}}$, s.t. the problem reduces to $\hat{f} : [0,1]^d \to [0,1]$ and the integral

$$\hat{I} = \int_0^1 \mathrm{d}x_1 \int_0^1 \mathrm{d}x_2 \; ... \int_0^1 \mathrm{d}x_d \; \hat{f}(\mathbf{x}). \tag{2.47}$$

Now consider $S = [0,1]^d$ and the set $A = \{(x_1,x_2,...,x_d,x_{d+1}) \in S : f(x_1,x_2,...,x_d) \leq x_{d+1}\}$, then the volume of A equals $\hat{I}$. For uniformly distributed random variables $X_1,X_2,...,X_{d+1}$, the volume of A equals the probability of $X_1,X_2,...,X_{d+1} \in A$, which can be calculated by generating N random vectors $\mathbf{x}_i \in [0,1]^{d+1}$ and evaluating $P = N_A/N$ with $N_A = |\{i \in [1,N] : \mathbf{x}_i \in A\}|$. Because "hit-or-miss" still forces a binomial distribution of N_A, it is associated with a variance $\propto N$, respectively an error $\propto \sqrt{N}$. Hence, the probability and the integral $\hat{I}$ have an error $\propto 1/\sqrt{N}$.
To obtain the integral of the original problem, one has to multiply with the original volume,

$$I = \hat{I} \max_{[\mathbf{a},\mathbf{b}]} f \prod_{i=1}^{d} (b_i - a_i). \tag{2.48}$$

General Monte Carlo

In multidimensional integrals, the boundarys of higher dimensions may depend on the lower dimension, s.t. the integration is done over a general area Ω: with $f : \Omega \to \mathbb{R}^+$. The general integral is

$$I = \int_{a_1}^{b_1} \mathrm{d}x_1 \int_{a_2(x_1)}^{b_2(x_1)} \mathrm{d}x_2 \int_{a_3(x_1,x_2)}^{b_2(x_1,x_2)} \mathrm{d}x_3 ... \int_{a_d(x_1,x_2,x_3,...,x_{d-1})}^{b_d(x_1,x_2,x_3,...,x_{d-1})} \mathrm{d}x_d \; f(\mathbf{x}) \tag{2.49}$$

The first approach is to use the "Hit-or-Miss"-method. Consider $S = [a_1,b_1] \times [a_2(a_1),b_2(b_1)]$
$\times [a_3(a_1,a_2),b_3(b_1,b_2)] \times ... \times [a_d(a_1,...,a_{d-1}),b_d(b_1,...,b_{d-1})] \times [0,\max f]$ and the set
$A = \{(x_1,...,x_{d+1}) \in S : (x_1,...,x_d) \in \Omega \wedge x_{d+1} \leq f(x_1,...,x_d)\}$. As before, the integral is given through the volume of A. Again, with random variables $X_1,...,X_{d+1}$ and N random vectors $\mathbf{x}_i = (X_1,...X_{d+1})$, one can define the number of hits as $N_A = |\{i \in [1,N] : \mathbf{x}_i \in A\}|$, s.t. the integral is given by

$$I = V_S \, \frac{N_A}{N}, \tag{2.50}$$

where V_S is the volume of S. As before, the error is $\propto 1/\sqrt{N}$, since nothing changed in the argumentation.
Second, one can use the sampling Monte Carlo (MC) approach. One can rewrite Eq. (2.49) to

$$I \approx \frac{1}{N^d} \sum_{i_1,...,i_d=1}^{N} (b_1 - a_1) \; f(Y_1^{i_1},...,Y_d^{i_d}) \prod_{j=2}^{d} \left[b_j(Y_1^{i_1},...,Y_{j-1}^{i_{j-1}}) - a_j(Y_1^{i_1},...,Y_{j-1}^{i_{j-1}}) \right], \tag{2.51}$$

where one has to ensure that the random variables Y_j are uniformly distributed in their intervals. This can be done via scaling according to the boundary functions. Consider $X_1, ..., X_d$ to be uniformly distributed in $[0,1]$. Then scale

$$Y_1 = X_1 \left(b_1 - a_1\right) + a_1 \tag{2.52}$$

$$Y_2 = X_2 \left(b_2(Y_1) - a_2(Y_1)\right) + a_2(Y_1) \tag{2.53}$$

$$\vdots \tag{2.54}$$

$$Y_d = X_d \left(b_d(Y_1, ...Y_d) - a_d(Y_1, ...Y_d)\right) + a_d(Y_1, ...Y_d) \tag{2.55}$$

and evaluate Eq. (2.51). Following the derivation in [2], the error of this method is $\propto N^{-d/2}$, whereas the calculation is done with $N' = N^d$ points in Ω, and the algorithm yields an error $\propto 1/\sqrt{N'}$.

Discrepancy Sampling

Discrepancy sampling is an efficient method to reduce the cost of higher dimensional Monte Carlo integration. I recommend [2] for an explanation. The method reduces the error to $\propto \log^d(N)/N$.

2.3 Numerical Results

The trapezoidal method was implemented as well as several Monte Carlo methods.
First, the trapezoidal method is compared to the Monte Carlo "Hit-or-Miss" method and to the simple sampling approach by calculating the area of a quarter unit circle

$$A = \int_0^1 \mathrm{d}x \, \sqrt{1 - x^2} = \frac{\pi}{4}. \tag{2.56}$$

The method's relative error in comparison to the analytical result is shown in Fig. 2.3, where for every N, the Monte Carlo integrations were done for 1000 samples to yield a stable result.
Both Monte Carlo methods follow the expected convergence behavior of a relative error $\propto 1/\sqrt{N}$. However, the trapezoidal method does not show the expected behavior. This can be derived from the fact, that we demanded the integrand to be C^2, which it is not in this case. Therefore, a different convergence behavior is not unusual.
Second, the simple sampling is compared to the importance sampling. For that approach, I chose the integral

$$I = \int_0^1 \mathrm{d}x e^{-100x} \approx \frac{1}{100}, \tag{2.57}$$

which can be computed with the simple sampling approach as usual. For importance sampling, I generated a random variable Y with pdf $g(x) = e^{-100x}$ by means of the inversion method. Afterwards, the integral has been computed as a mean of Y. The comparison of the relative error is shown in Fig. 2.4. As one can see, the importance sampling method yields better results while the convergence behavior of methods still follows $1/\sqrt{N}$.
The general MC method by means of simple sampling Eq. (2.51) has been tested for the area Ω from Eq. (2.22) with the integral

$$I = \int_0^h \mathrm{d}x_1 \int_0^{x_1} \mathrm{d}x_2 ... \int_0^{x_{d-1}} \mathrm{d}x_d \, \sum_{i=1}^{d} x_i, \tag{2.58}$$

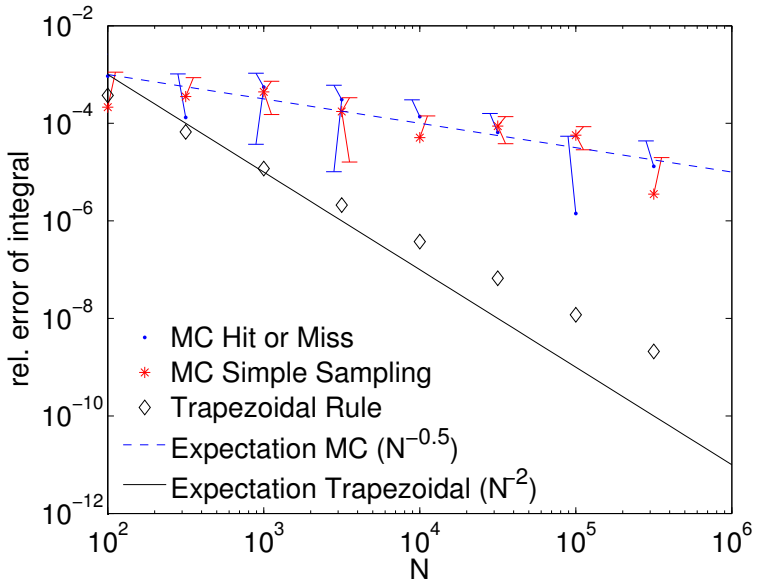

Figure 2.3: Relative error of different approaches for numerical integration of the function $f(x) = \sqrt{1-x^2}$. The MC methods have been used 1000 times each for every N to build a mean of their results.

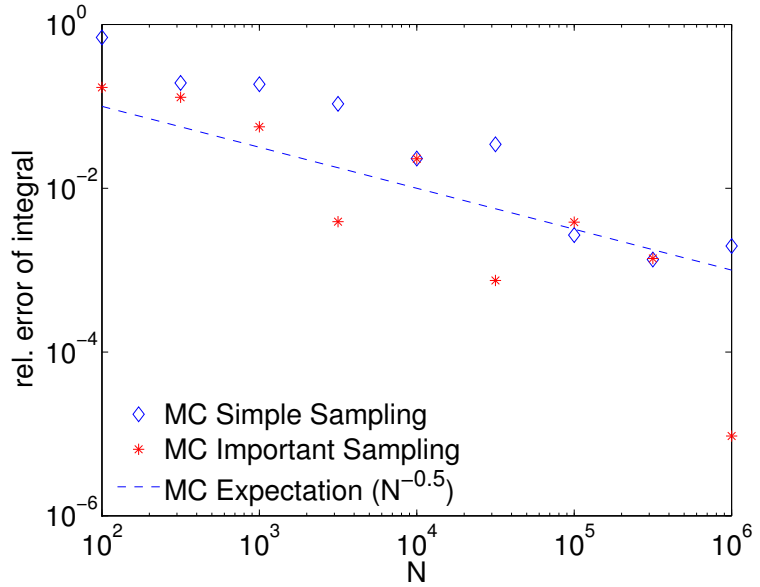

Figure 2.4: Comparison between simple sampling and importance sampling for the integral $\int_0^1 \mathrm{d}x\ \exp(-100x)$.

which can be evaluated analytically by means of Eq. (2.26) to

$$I = \frac{h^{d+1}}{(d-1)!}. \tag{2.59}$$

The numerical computation has been done for $d = 2,...,7$ and for several numbers N of evaluation points, with 500 samples for each. The convergence behavior of the method, which is shown in Fig. 2.5, is still $\propto 1/\sqrt{N}$, whereas the higher dimensionality shifts this behavior by a factor.

For the comparison between trapezoidal integration, normal "Hit or Miss" and the discrepancy sampling "Hit or Miss", I use two different integrals, first

$$I_1 = \int_0^1 \mathrm{d}x_1 \int_0^1 \mathrm{d}x_2 ... \int_0^1 \mathrm{d}x_{d-1} \sqrt{1 - \sum_{i=1}^{d-1} x_i^2} = \frac{\pi^{d/2}}{2^d \Gamma(d/2+1)}, \tag{2.60}$$

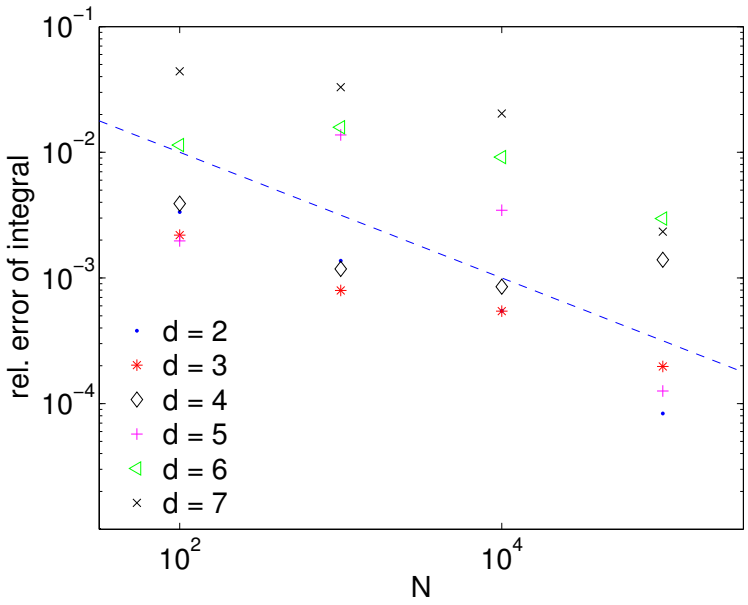

Figure 2.5: General MC method for the integral Eq. (2.58) and different domain dimensions. The convergence shows the expected behavior (dotted line).

which equals the volume of a 2^d-th of the unity ball in $\mathbb{R}^b$. Since this integral is not C^2 in $[0,1]^{d-1}$, which results in an unexpected convergence behavior of the trapezoidal method, I also evaluate the integral

$$I_2 = \int_0^1 \mathrm{d}x_1 \ldots \int_0^1 \mathrm{d}x_{d-1} \prod_{i=1}^{d-1} \sin(x_i) = \sum_{i=0}^{d} \binom{d}{i} (-\cos(1))^i. \tag{2.61}$$

The results are shown in figures Fig. 2.6 and Fig. 2.7. They show the expected behavior, except the trapezoidal method for I_1.

2.4 Conclusion

The Monte Carlo methods provide efficient approaches for numerical calculations. Their main effort is the convergence behavior, which does not depend on the problem's dimension, because of their statistical formulation.

However, for non-discrepancy sampling, the trapezoidal method yields a stronger convergence behavior for a domain dimensionality of $d < 4$. Furthermore, the method's convergence behavior seems to be stronger for $d < 3$. For other applications, I recommend the discrepancy sampling MC method.

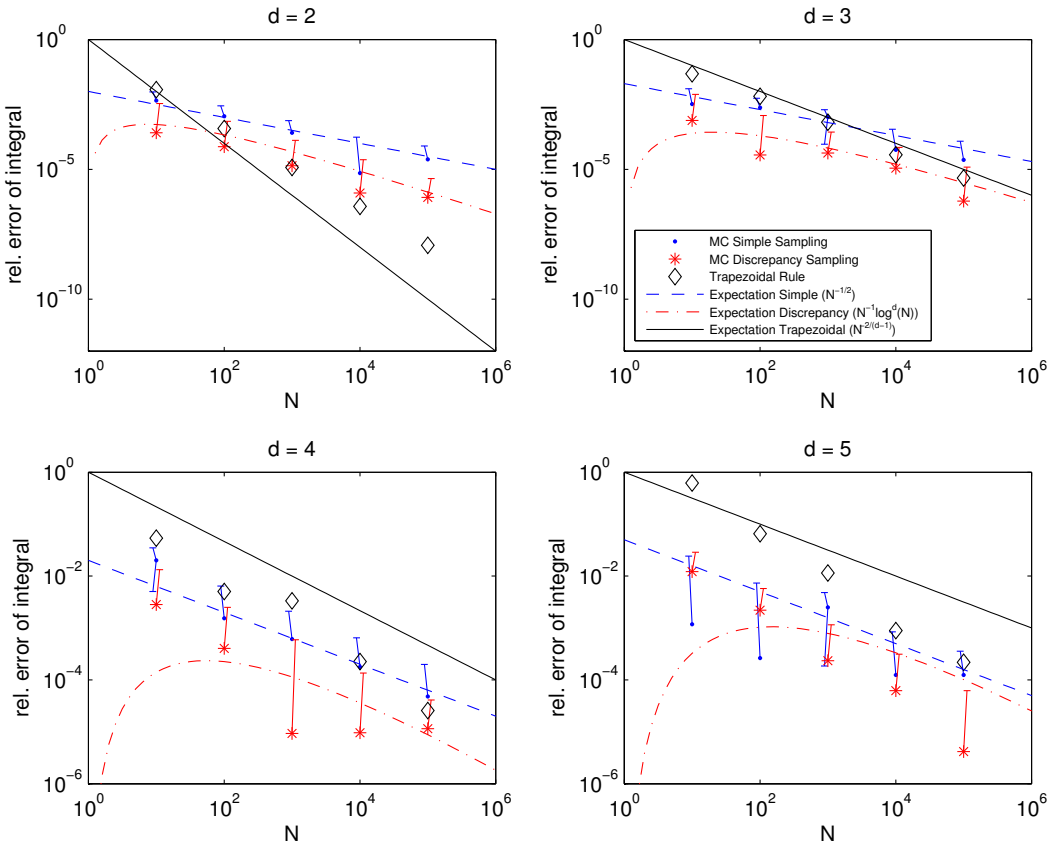

Figure 2.6: Comparison between normal "Hit or Miss", discrepancy sampling and the trapezoidal method for I_1. Because the integrand is not C^2 in the interval, the error of the trapezoidal method differs from the expectation.

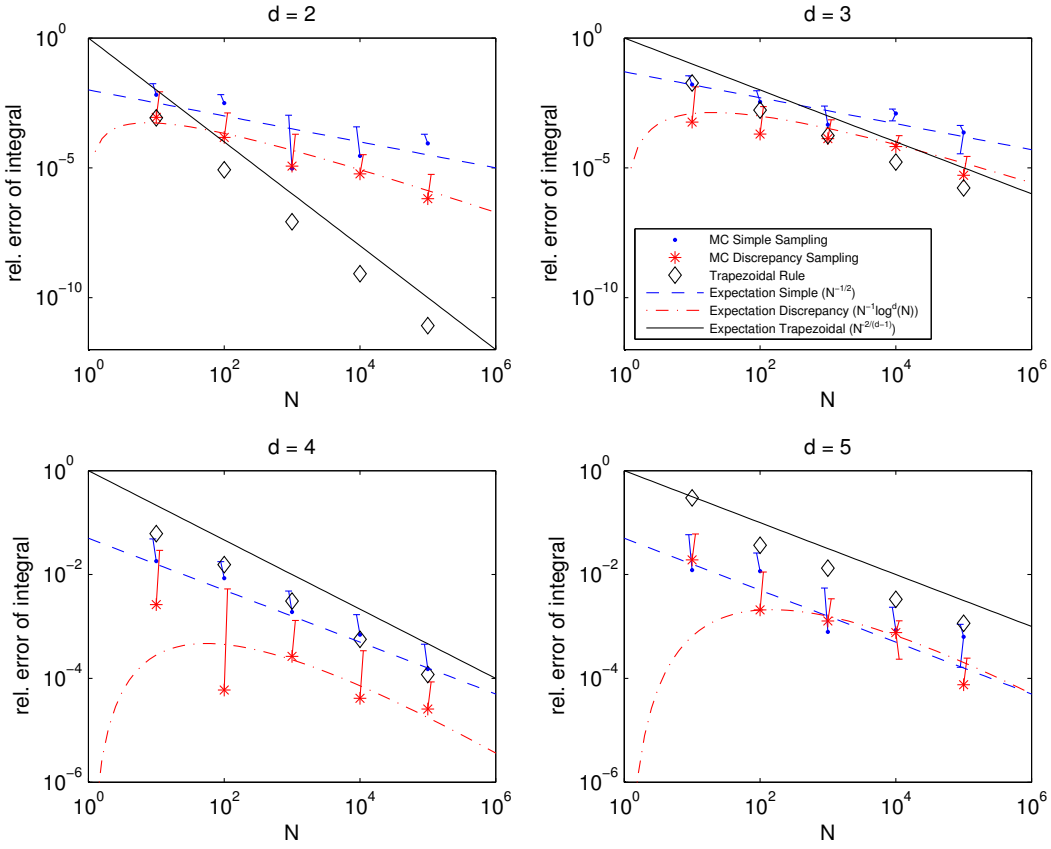

Figure 2.7: Comparison between normal "Hit or Miss", discrepancy sampling and the trapezoidal method for I_2

Appendix A

χ^2-Tests in Detail

The confidence interval is defined by confidence probabilites q_{low} and q_{high}, where the tables of the corresponding χ^2-values are taken from [3].

RANDU

q_high	q_low	max. exceeds	exceeds	test
0.100	0.100	20.000 perc	18.600 perc	pass
0.100	0.050	15.000 perc	15.200 perc	fail
0.100	0.025	12.500 perc	12.000 perc	pass
0.100	0.010	11.000 perc	10.400 perc	pass
0.050	0.100	15.000 perc	12.000 perc	pass
0.050	0.050	10.000 perc	12.800 perc	fail
0.050	0.025	7.500 perc	7.000 perc	pass
0.050	0.010	6.000 perc	5.600 perc	pass
0.025	0.100	12.500 perc	14.400 perc	fail
0.025	0.050	7.500 perc	6.000 perc	pass
0.025	0.025	5.000 perc	5.800 perc	fail
0.025	0.010	3.500 perc	4.400 perc	fail
0.010	0.100	11.000 perc	9.400 perc	pass
0.010	0.050	6.000 perc	5.600 perc	pass
0.010	0.025	3.500 perc	2.400 perc	pass
0.010	0.010	2.000 perc	1.600 perc	pass

Number of fails: 5 out of 16

Quick

q_high	q_low	max. exceeds	exceeds	test
0.100	0.100	20.000 perc	19.400 perc	pass
0.100	0.050	15.000 perc	17.800 perc	fail
0.100	0.025	12.500 perc	12.800 perc	fail
0.100	0.010	11.000 perc	10.600 perc	pass

0.050	0.100	15.000 perc	15.600 perc	fail
0.050	0.050	10.000 perc	10.200 perc	fail
0.050	0.025	7.500 perc	6.800 perc	pass
0.050	0.010	6.000 perc	4.000 perc	pass
0.025	0.100	12.500 perc	12.800 perc	fail
0.025	0.050	7.500 perc	7.000 perc	pass
0.025	0.025	5.000 perc	5.400 perc	fail
0.025	0.010	3.500 perc	4.600 perc	fail
0.010	0.100	11.000 perc	12.000 perc	fail
0.010	0.050	6.000 perc	6.600 perc	fail
0.010	0.025	3.500 perc	5.400 perc	fail
0.010	0.010	2.000 perc	2.600 perc	fail

Number of fails: 11 out of 16

Noname

q_high	q_low	max. exceeds	exceeds	test
0.100	0.100	20.000 perc	49.200 perc	fail
0.100	0.050	15.000 perc	46.200 perc	fail
0.100	0.025	12.500 perc	45.800 perc	fail
0.100	0.010	11.000 perc	46.000 perc	fail
0.050	0.100	15.000 perc	42.200 perc	fail
0.050	0.050	10.000 perc	44.000 perc	fail
0.050	0.025	7.500 perc	44.800 perc	fail
0.050	0.010	6.000 perc	43.600 perc	fail
0.025	0.100	12.500 perc	41.600 perc	fail
0.025	0.050	7.500 perc	44.200 perc	fail
0.025	0.025	5.000 perc	42.200 perc	fail
0.025	0.010	3.500 perc	42.400 perc	fail
0.010	0.100	11.000 perc	40.000 perc	fail

Park-Miller

q_high	q_low	max. exceeds	exceeds	test
0.100	0.100	20.000 perc	21.400 perc	fail
0.100	0.050	15.000 perc	14.400 perc	pass
0.100	0.025	12.500 perc	15.000 perc	fail
0.100	0.010	11.000 perc	8.800 perc	pass
0.050	0.100	15.000 perc	17.400 perc	fail
0.050	0.050	10.000 perc	12.200 perc	fail
0.050	0.025	7.500 perc	7.600 perc	fail
0.050	0.010	6.000 perc	6.400 perc	fail
0.025	0.100	12.500 perc	12.600 perc	fail
0.025	0.050	7.500 perc	8.600 perc	fail

0.025	0.025	5.000 perc	5.800 perc	fail
0.025	0.010	3.500 perc	3.800 perc	fail
0.010	0.100	11.000 perc	12.200 perc	fail

Bad

q_high	q_low	max. exceeds	exceeds	test
0.100	0.100	20.000 perc	100.000 perc	fail
0.100	0.050	15.000 perc	100.000 perc	fail
0.100	0.025	12.500 perc	100.000 perc	fail
0.100	0.010	11.000 perc	100.000 perc	fail
0.050	0.100	15.000 perc	100.000 perc	fail
0.050	0.050	10.000 perc	100.000 perc	fail
0.050	0.025	7.500 perc	100.000 perc	fail
0.050	0.010	6.000 perc	100.000 perc	fail
0.025	0.100	12.500 perc	100.000 perc	fail
0.025	0.050	7.500 perc	100.000 perc	fail
0.025	0.025	5.000 perc	100.000 perc	fail
0.025	0.010	3.500 perc	100.000 perc	fail
0.010	0.100	11.000 perc	100.000 perc	fail

Unix

q_high	q_low	max. exceeds	exceeds	test
0.100	0.100	20.000 perc	20.400 perc	fail
0.100	0.050	15.000 perc	15.600 perc	fail
0.100	0.025	12.500 perc	13.200 perc	fail
0.100	0.010	11.000 perc	9.800 perc	pass
0.050	0.100	15.000 perc	15.000 perc	pass
0.050	0.050	10.000 perc	11.400 perc	fail
0.050	0.025	7.500 perc	8.000 perc	fail
0.050	0.010	6.000 perc	5.000 perc	pass
0.025	0.100	12.500 perc	11.600 perc	pass
0.025	0.050	7.500 perc	9.200 perc	fail
0.025	0.025	5.000 perc	5.600 perc	fail
0.025	0.010	3.500 perc	4.000 perc	fail
0.010	0.100	11.000 perc	11.400 perc	fail

HRC (with GSL and Park-Miller)

q_high	q_low	max. exceeds	exceeds	test
0.100	0.100	20.000 perc	19.400 perc	pass
0.100	0.050	15.000 perc	17.800 perc	fail

0.100	0.025	12.500 perc	15.200 perc	fail
0.100	0.010	11.000 perc	11.400 perc	fail
0.050	0.100	15.000 perc	19.200 perc	fail
0.050	0.050	10.000 perc	10.400 perc	fail
0.050	0.025	7.500 perc	7.200 perc	pass
0.050	0.010	6.000 perc	6.200 perc	fail
0.025	0.100	12.500 perc	11.200 perc	pass
0.025	0.050	7.500 perc	7.600 perc	fail
0.025	0.025	5.000 perc	6.000 perc	fail
0.025	0.010	3.500 perc	4.000 perc	fail
0.010	0.100	11.000 perc	11.800 perc	fail

STD

q_high	q_low	max. exceeds	exceeds	test
0.100	0.100	20.000 perc	21.200 perc	fail
0.100	0.050	15.000 perc	14.800 perc	pass
0.100	0.025	12.500 perc	13.400 perc	fail
0.100	0.010	11.000 perc	12.200 perc	fail
0.050	0.100	15.000 perc	14.400 perc	pass
0.050	0.050	10.000 perc	8.600 perc	pass
0.050	0.025	7.500 perc	6.600 perc	pass
0.050	0.010	6.000 perc	5.400 perc	pass
0.025	0.100	12.500 perc	14.000 perc	fail
0.025	0.050	7.500 perc	9.800 perc	fail
0.025	0.025	5.000 perc	5.200 perc	fail
0.025	0.010	3.500 perc	4.200 perc	fail
0.010	0.100	11.000 perc	10.600 perc	pass

GSL

q_high	q_low	max. exceeds	exceeds	test
0.100	0.100	20.000 perc	20.400 perc	fail
0.100	0.050	15.000 perc	15.600 perc	fail
0.100	0.025	12.500 perc	13.200 perc	fail
0.100	0.010	11.000 perc	9.800 perc	pass
0.050	0.100	15.000 perc	15.000 perc	fail
0.050	0.050	10.000 perc	11.400 perc	fail
0.050	0.025	7.500 perc	8.000 perc	fail
0.050	0.010	6.000 perc	5.000 perc	pass
0.025	0.100	12.500 perc	11.600 perc	pass
0.025	0.050	7.500 perc	9.200 perc	fail
0.025	0.025	5.000 perc	5.600 perc	fail
0.025	0.010	3.500 perc	4.000 perc	fail
0.010	0.100	11.000 perc	11.400 perc	fail

```
0.010   0.050     6.000 perc      6.000 perc    fail
0.010   0.025     3.500 perc      3.200 perc    pass
0.010   0.010     2.000 perc      1.800 perc    pass

Number of fails: 11 out of 16
```

SUN

```
q_high  q_low    max. exceeds       exceeds     test
=====================================================================
0.100   0.100    20.000 perc     21.400 perc    fail
0.100   0.050    15.000 perc     16.200 perc    fail
0.100   0.025    12.500 perc     13.400 perc    fail
0.100   0.010    11.000 perc     10.400 perc    pass
0.050   0.100    15.000 perc     15.800 perc    fail
0.050   0.050    10.000 perc     11.800 perc    fail
0.050   0.025     7.500 perc      8.400 perc    fail
0.050   0.010     6.000 perc      5.400 perc    pass
0.025   0.100    12.500 perc     11.400 perc    pass
0.025   0.050     7.500 perc      9.400 perc    fail
0.025   0.025     5.000 perc      5.800 perc    fail
0.025   0.010     3.500 perc      3.400 perc    pass
0.010   0.100    11.000 perc     11.600 perc    fail
0.010   0.050     6.000 perc      5.800 perc    pass
0.010   0.025     3.500 perc      3.000 perc    pass
0.010   0.010     2.000 perc      1.800 perc    pass

Number of fails: 9 out of 16
```

Appendix B

Figures of RNG Tests

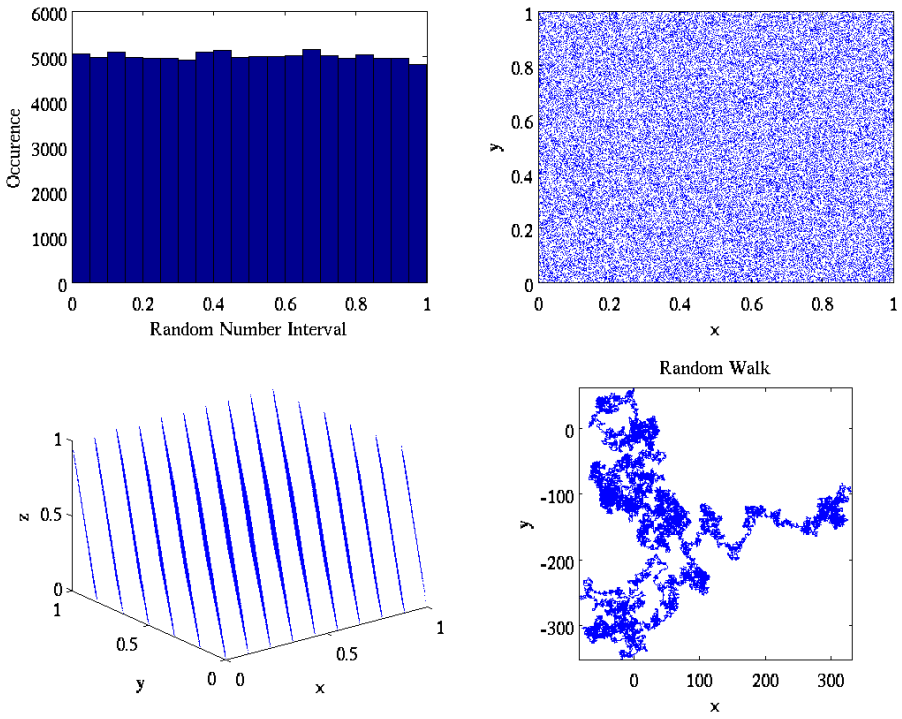

Figure B.1: Test of the RNG **RANDU** with 10^5 random numbers. Top Left: Histogram. Top Right: Correlation test in 2D. Bottom left: Correlation test in 3D. Bottom right: 2D random walk.

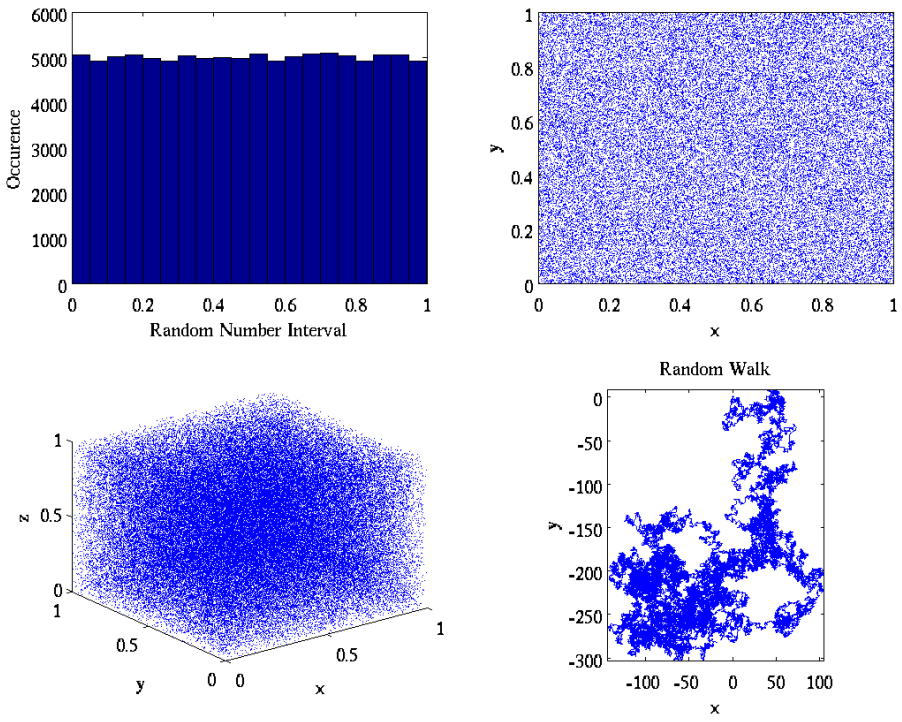

Figure B.2: Test of the RNG **Quick** with 10^5 random numbers. Top Left: Histogram. Top Right: Correlation test in 2D. Bottom left: Correlation test in 3D. Bottom right: 2D random walk.

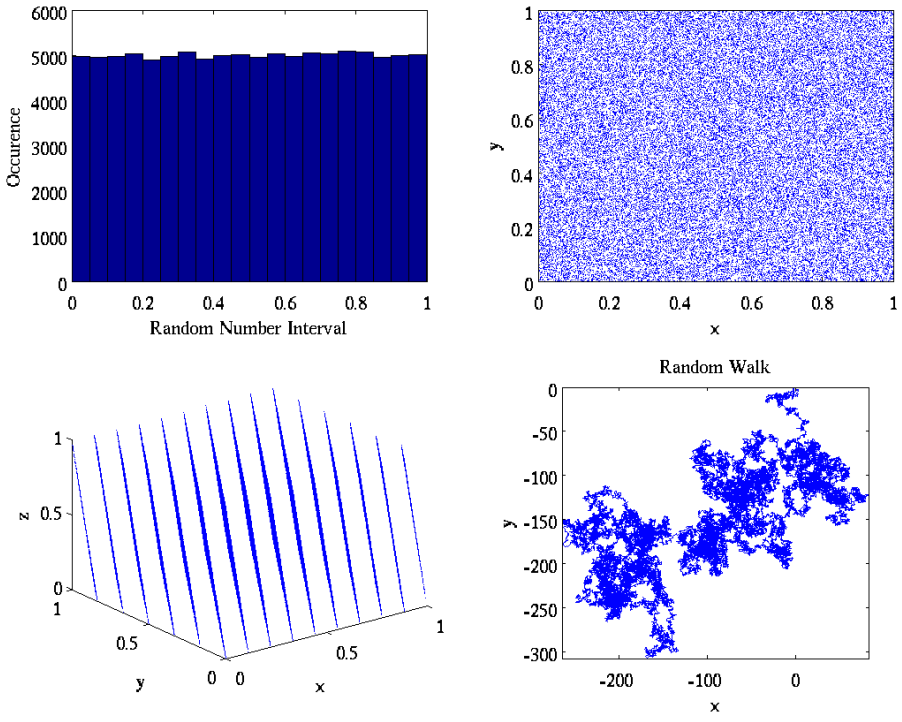

Figure B.3: Test of the RNG **Noname** with 10^5 random numbers. Top Left: Histogram. Top Right: Correlation test in 2D. Bottom left: Correlation test in 3D. Bottom right: 2D random walk.

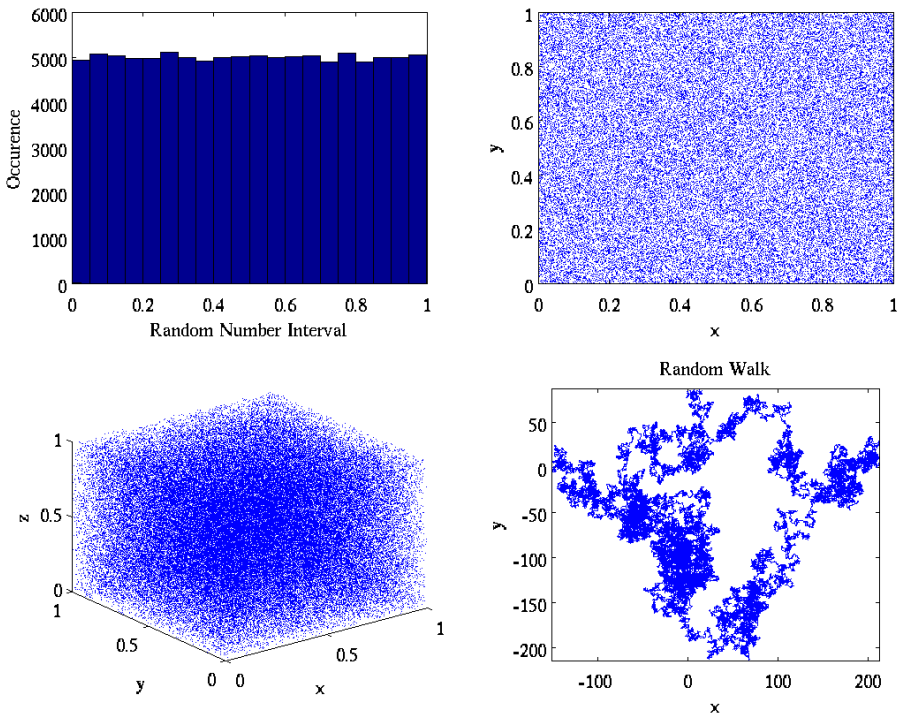

Figure B.4: Test of the RNG **Park-Miller** with 10^5 random numbers. Top Left: Histogram. Top Right: Correlation test in 2D. Bottom left: Correlation test in 3D. Bottom right: 2D random walk.

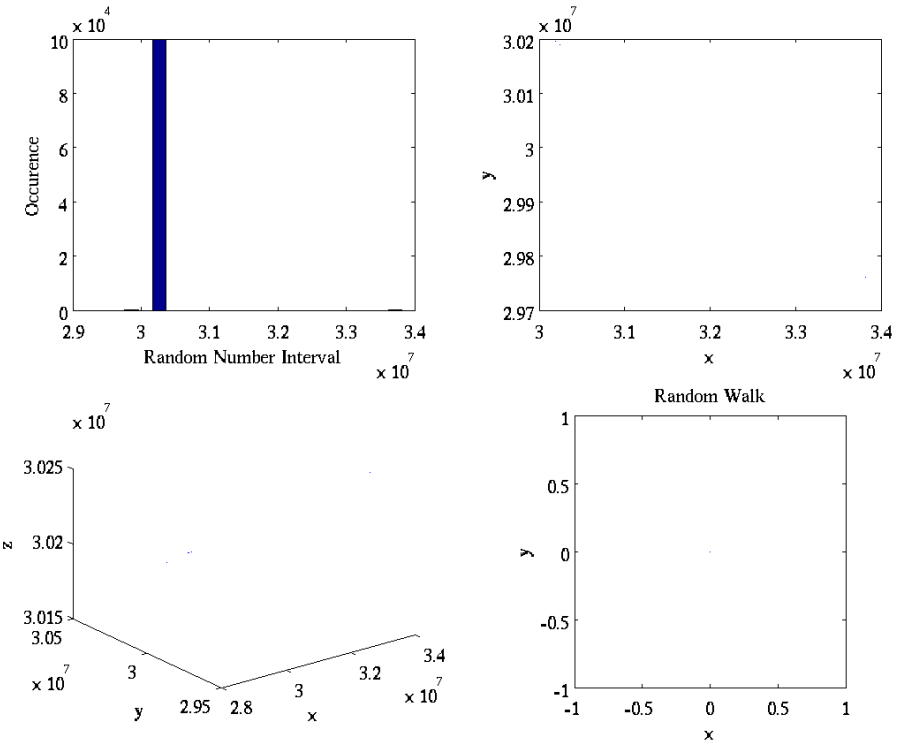

Figure B.5: Test of the RNG **Bad** with 10^5 random numbers. Top Left: Histogram. Top Right: Correlation test in 2D. Bottom left: Correlation test in 3D. Bottom right: 2D random walk.

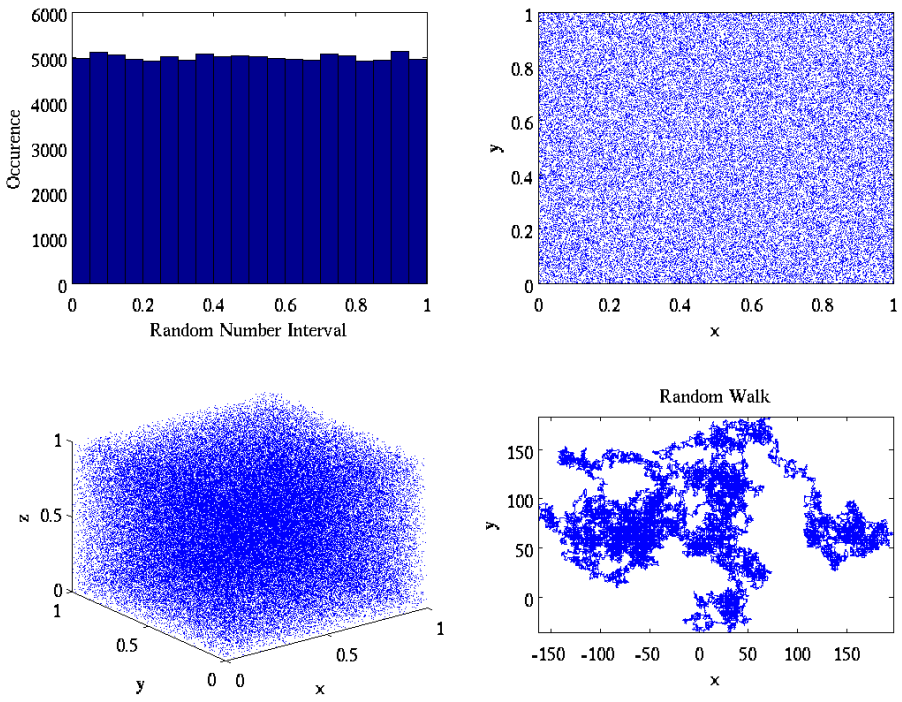

Figure B.6: Test of the RNG **Unix** with 10^5 random numbers. Top Left: Histogram. Top Right: Correlation test in 2D. Bottom left: Correlation test in 3D. Bottom right: 2D random walk.

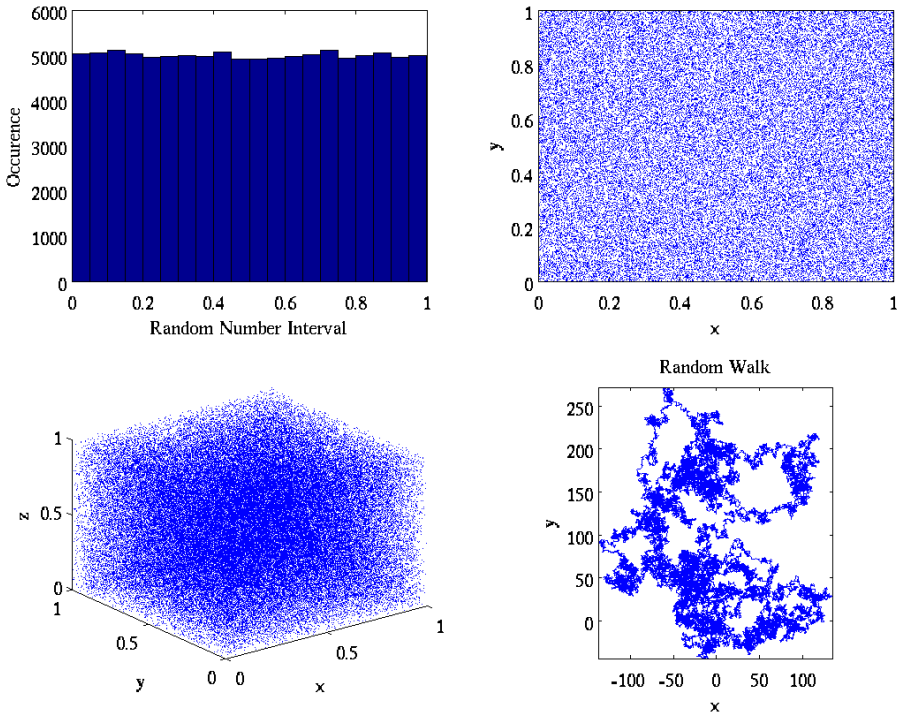

Figure B.7: Test of the RNG **HRC** with 10^5 random numbers. Top Left: Histogram. Top Right: Correlation test in 2D. Bottom left: Correlation test in 3D. Bottom right: 2D random walk.

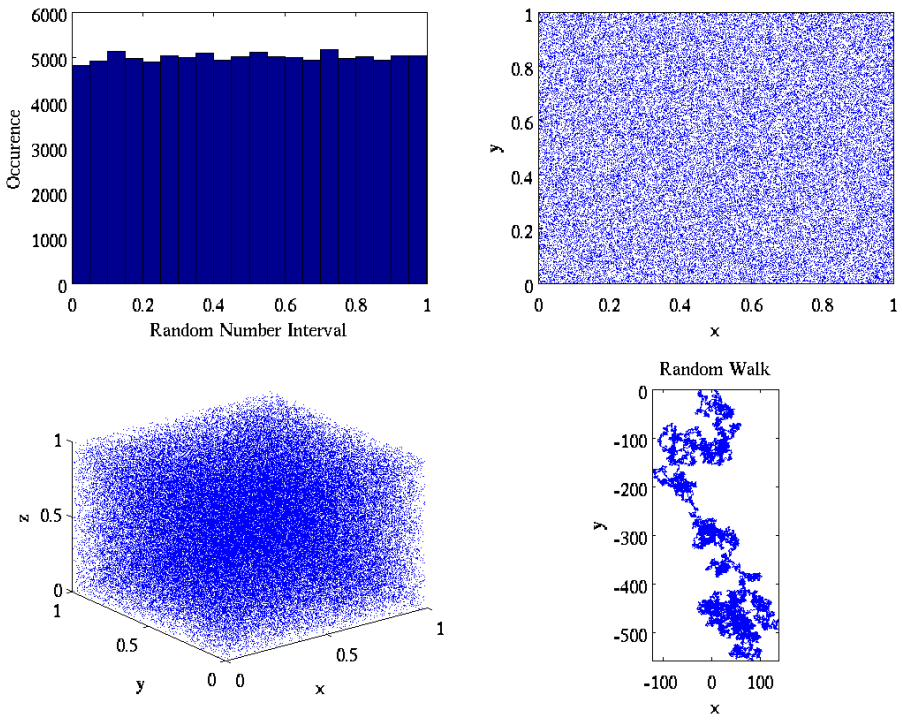

Figure B.8: Test of the RNG **STD** with 10^5 random numbers. Top Left: Histogram. Top Right: Correlation test in 2D. Bottom left: Correlation test in 3D. Bottom right: 2D random walk.

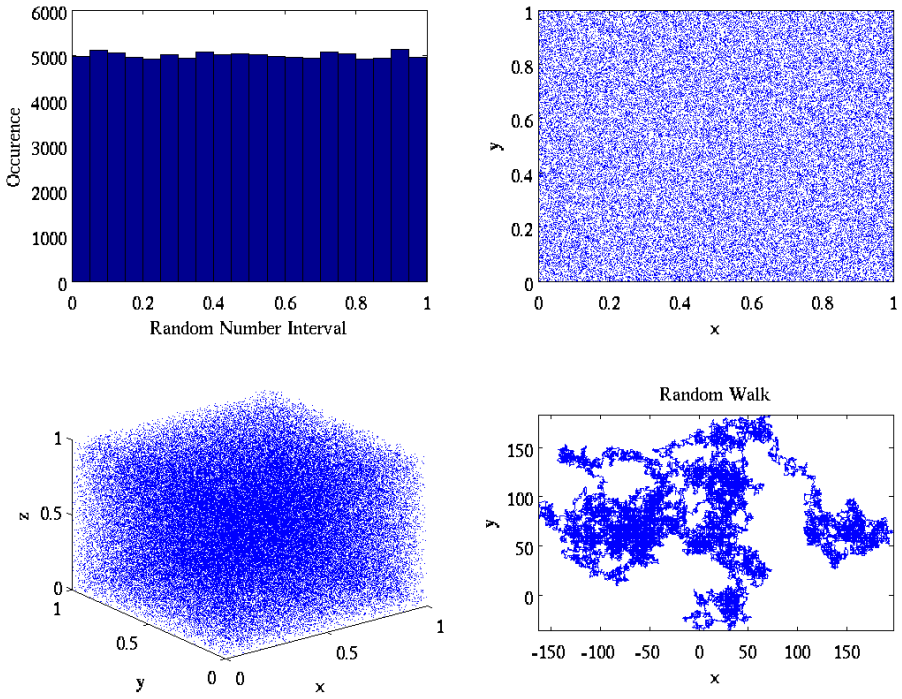

Figure B.9: Test of the RNG **GSL** with 10^5 random numbers. Top Left: Histogram. Top Right: Correlation test in 2D. Bottom left: Correlation test in 3D. Bottom right: 2D random walk.

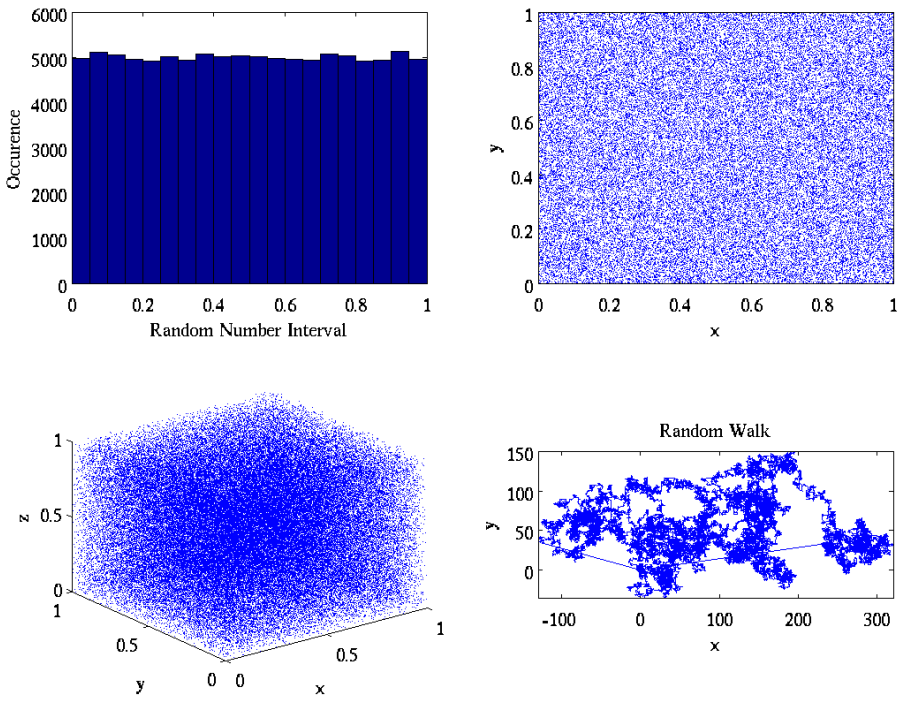

Figure B.10: Test of the RNG **SUN** with 10^5 random numbers. Top Left: Histogram. Top Right: Correlation test in 2D. Bottom left: Correlation test in 3D. Bottom right: 2D random walk.

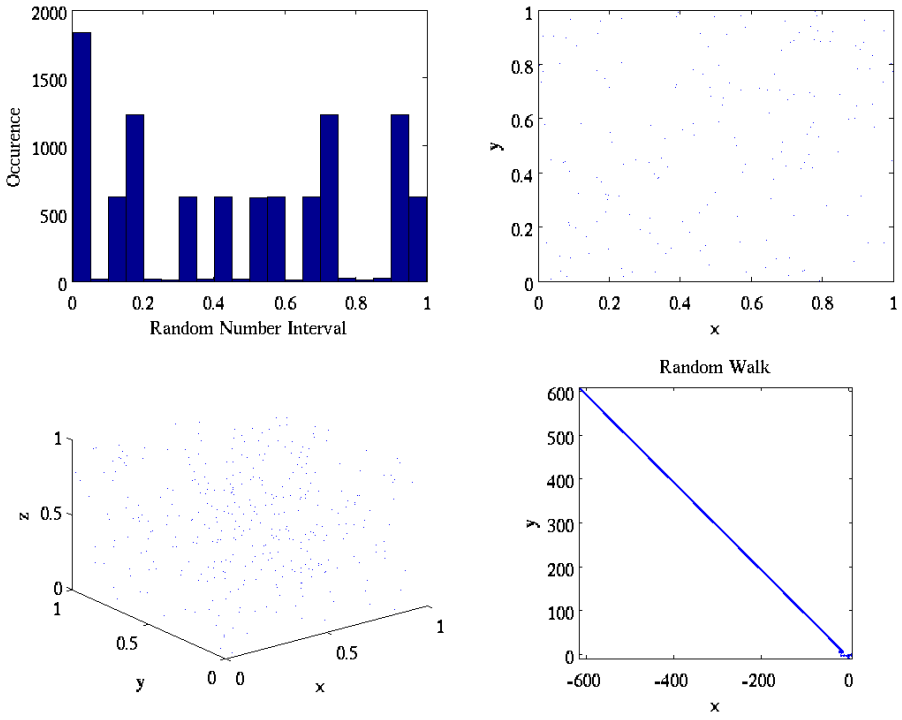

Figure B.11: Test of the RNG **HRC** (here: combination of GSL and SUN) with 10^4 random numbers. Top Left: Histogram. Top Right: Correlation test in 2D. Bottom left: Correlation test in 3D. Bottom right: 2D random walk.

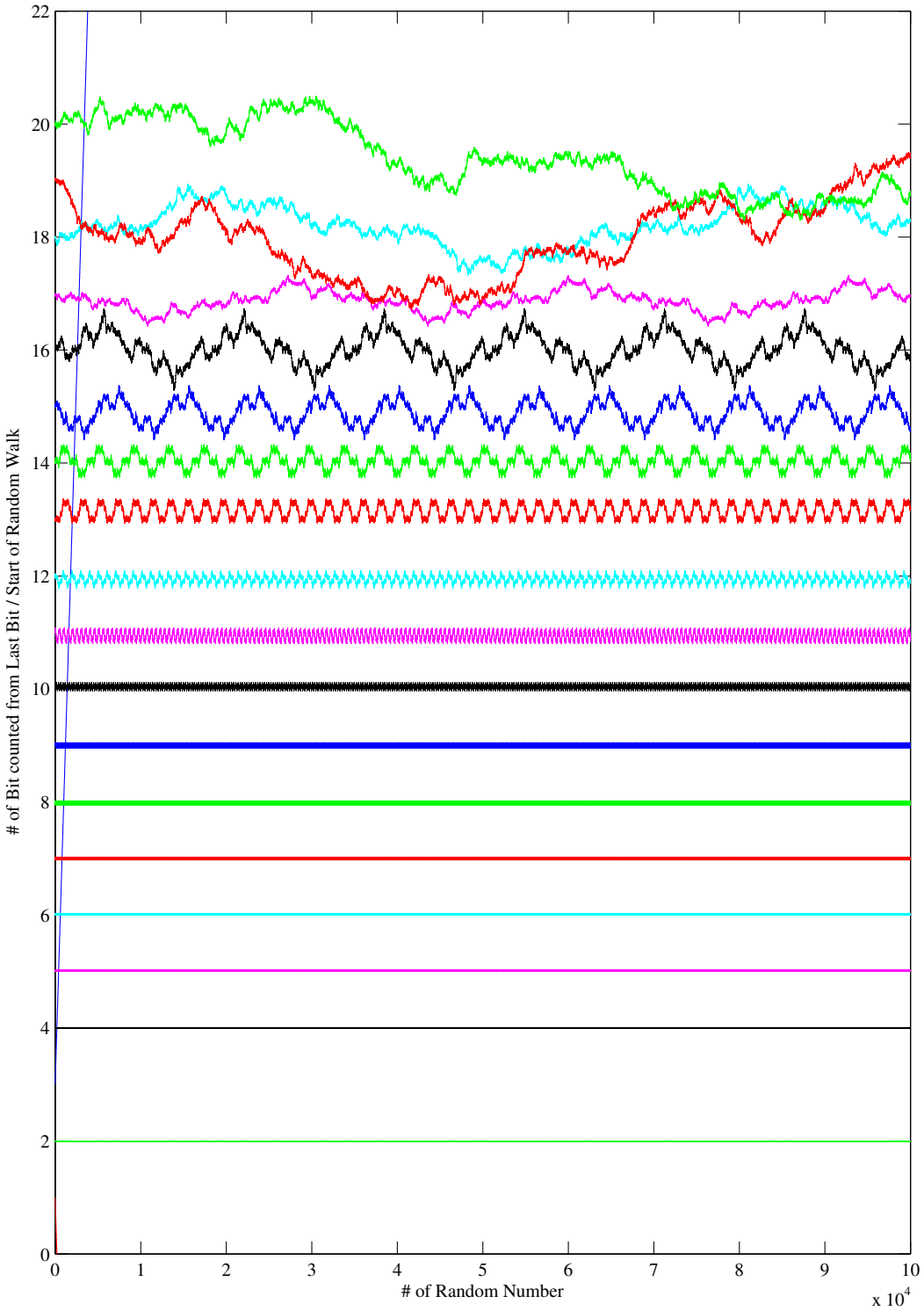

Figure B.12: Binary random walks for the lowest 20 bits of random numbers generated with the **RANDU** RNG. The seed is 45.

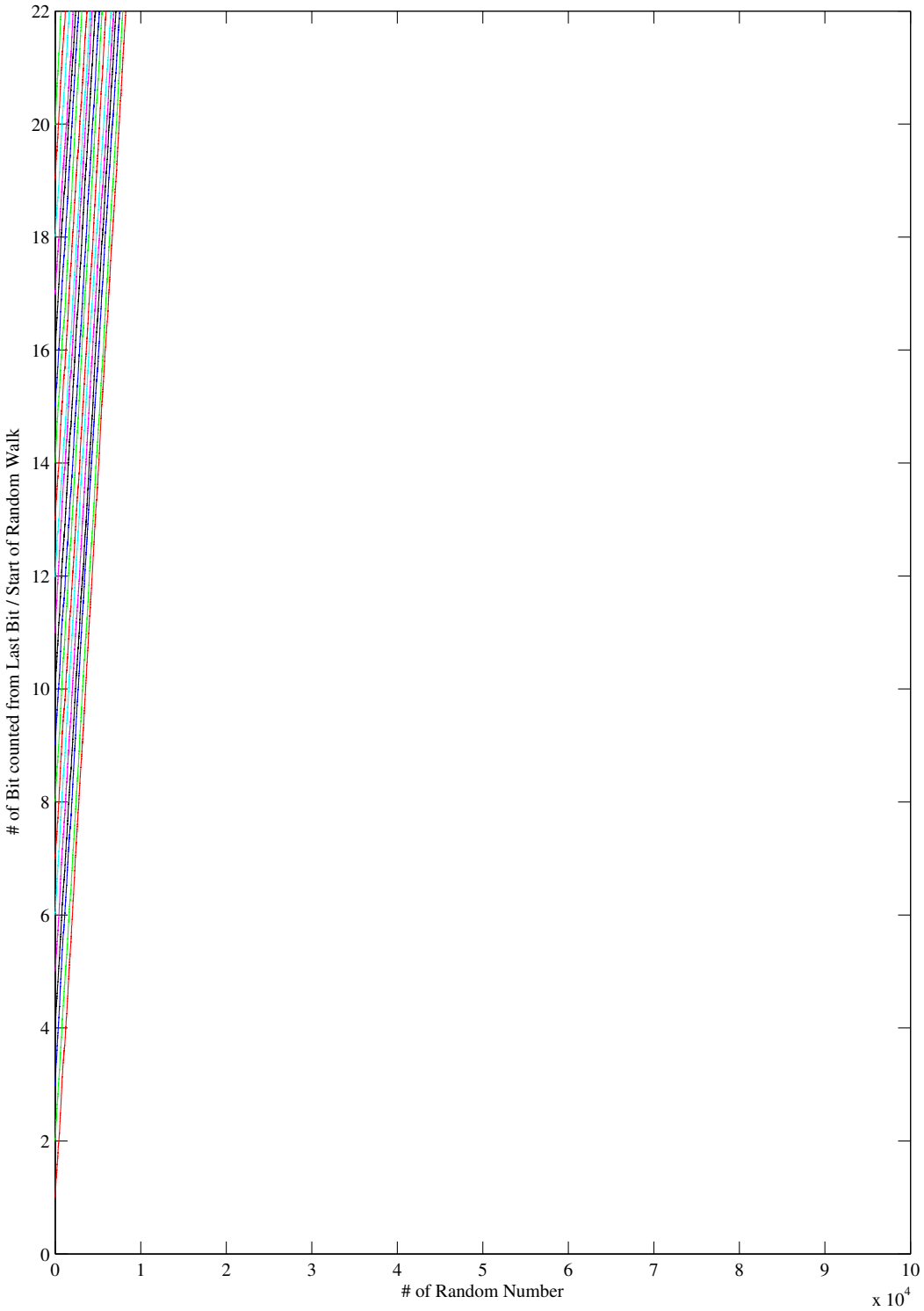

Figure B.13: Binary random walks for the lowest 20 bits of random numbers generated with the **Quick** RNG. The seed is 45.

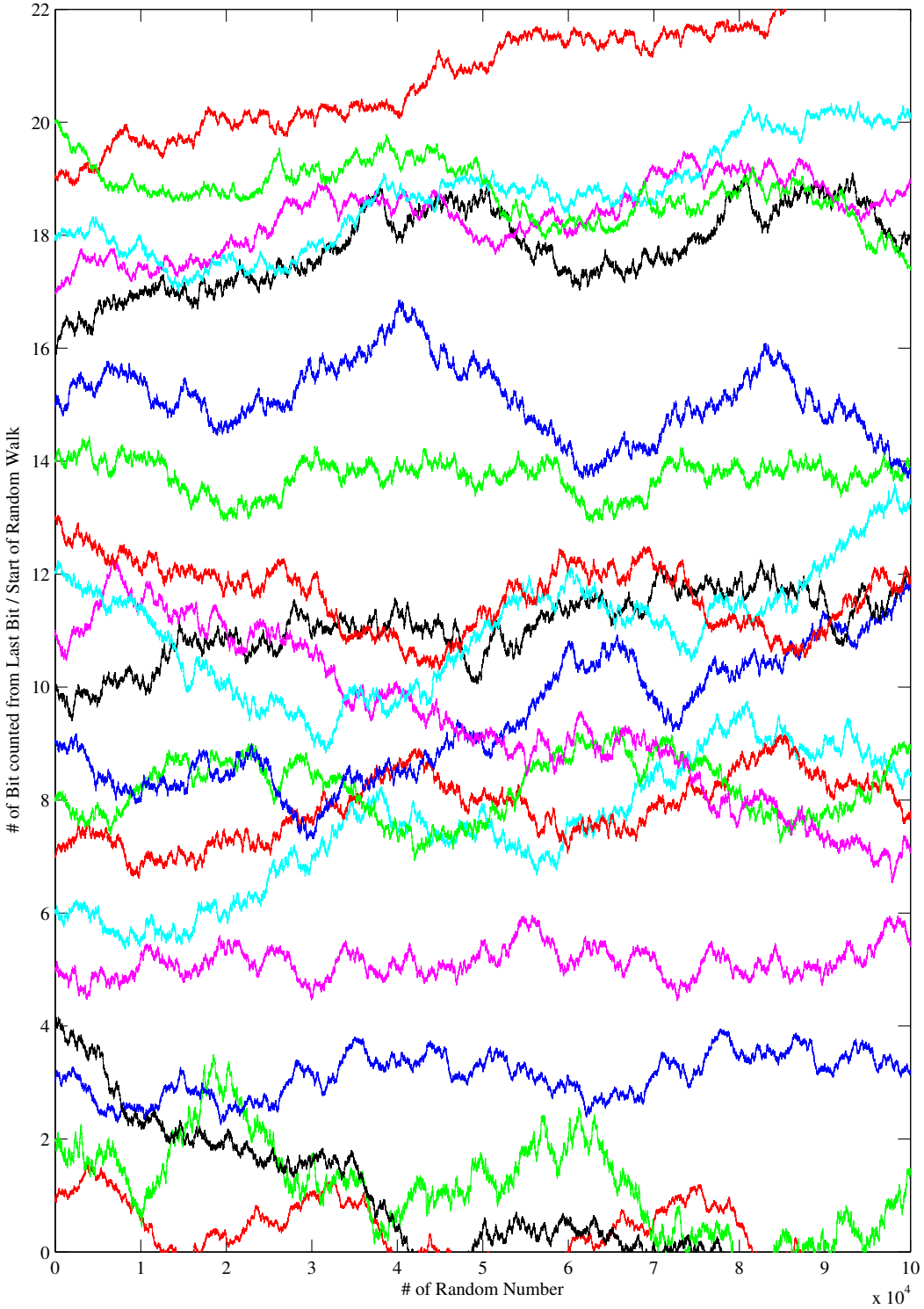

Figure B.14: Binary random walks for the lowest 20 bits of random numbers generated with the **Noname** RNG. The seed is 45.

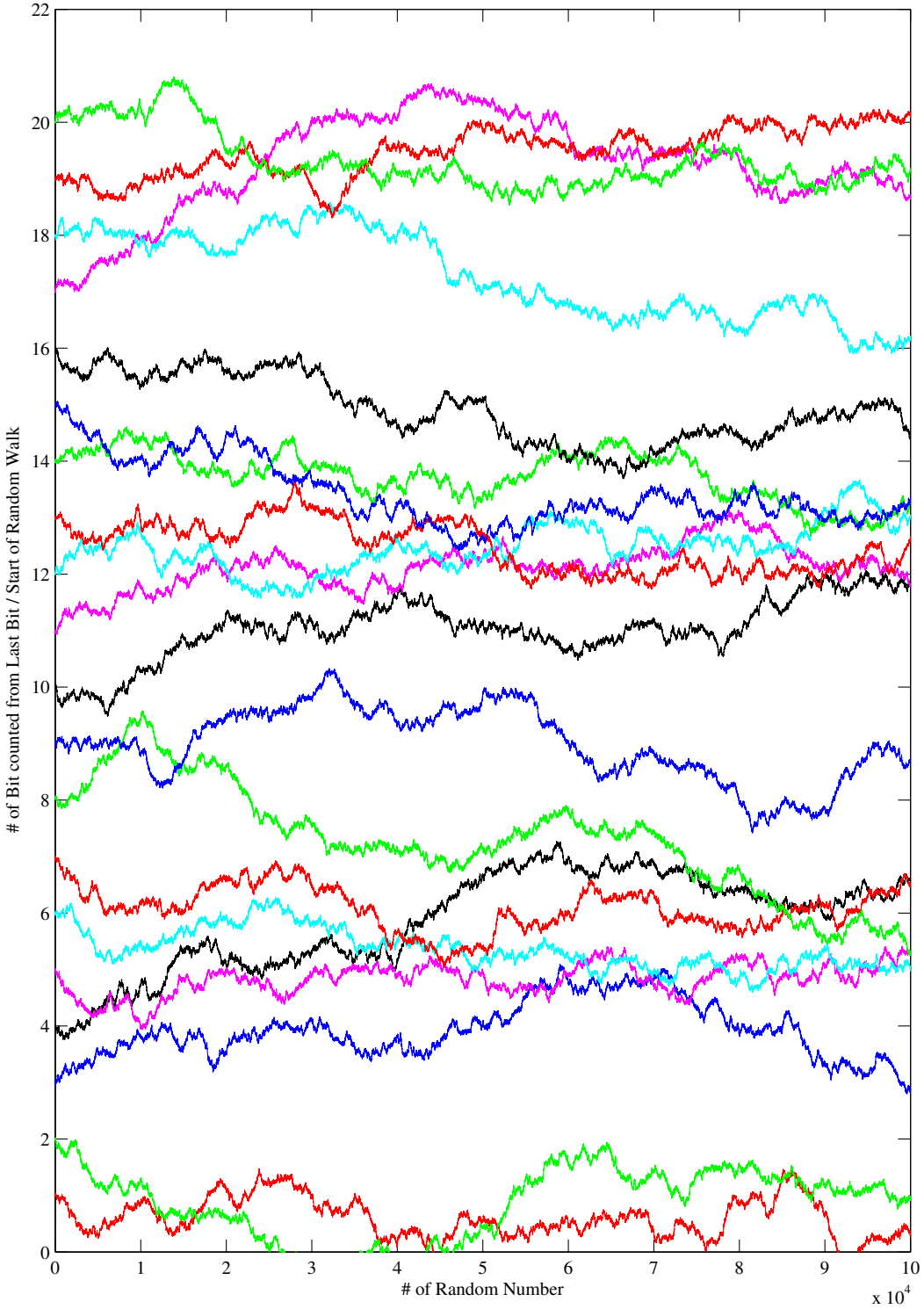

Figure B.15: Binary random walks for the lowest 20 bits of random numbers generated with the **Park-Miller** RNG. The seed is 45.

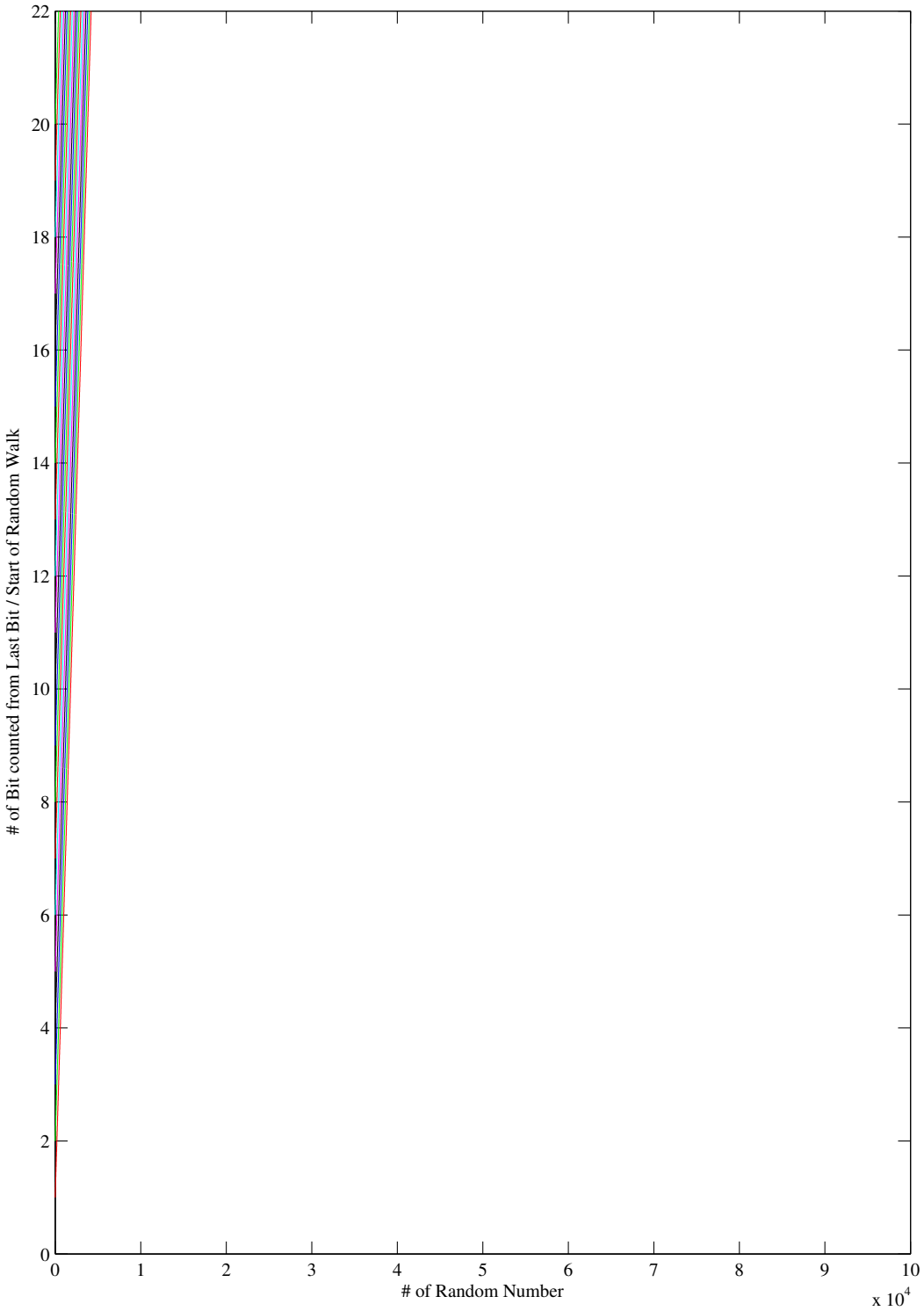

Figure B.16: Binary random walks for the lowest 20 bits of random numbers generated with the **Bad** RNG. The seed is 45.

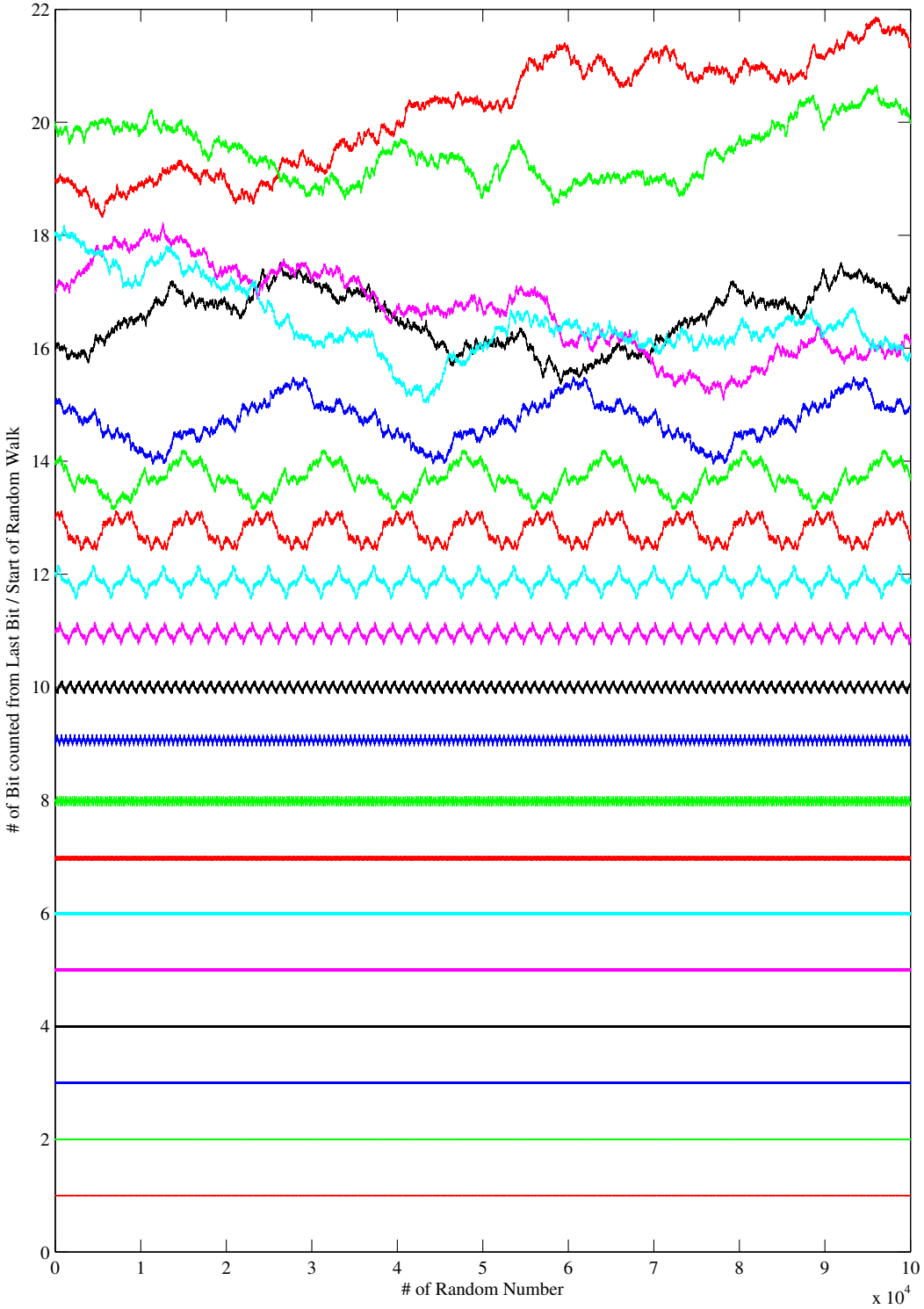

Figure B.17: Binary random walks for the lowest 20 bits of random numbers generated with the **Unix** RNG. The seed is 45.

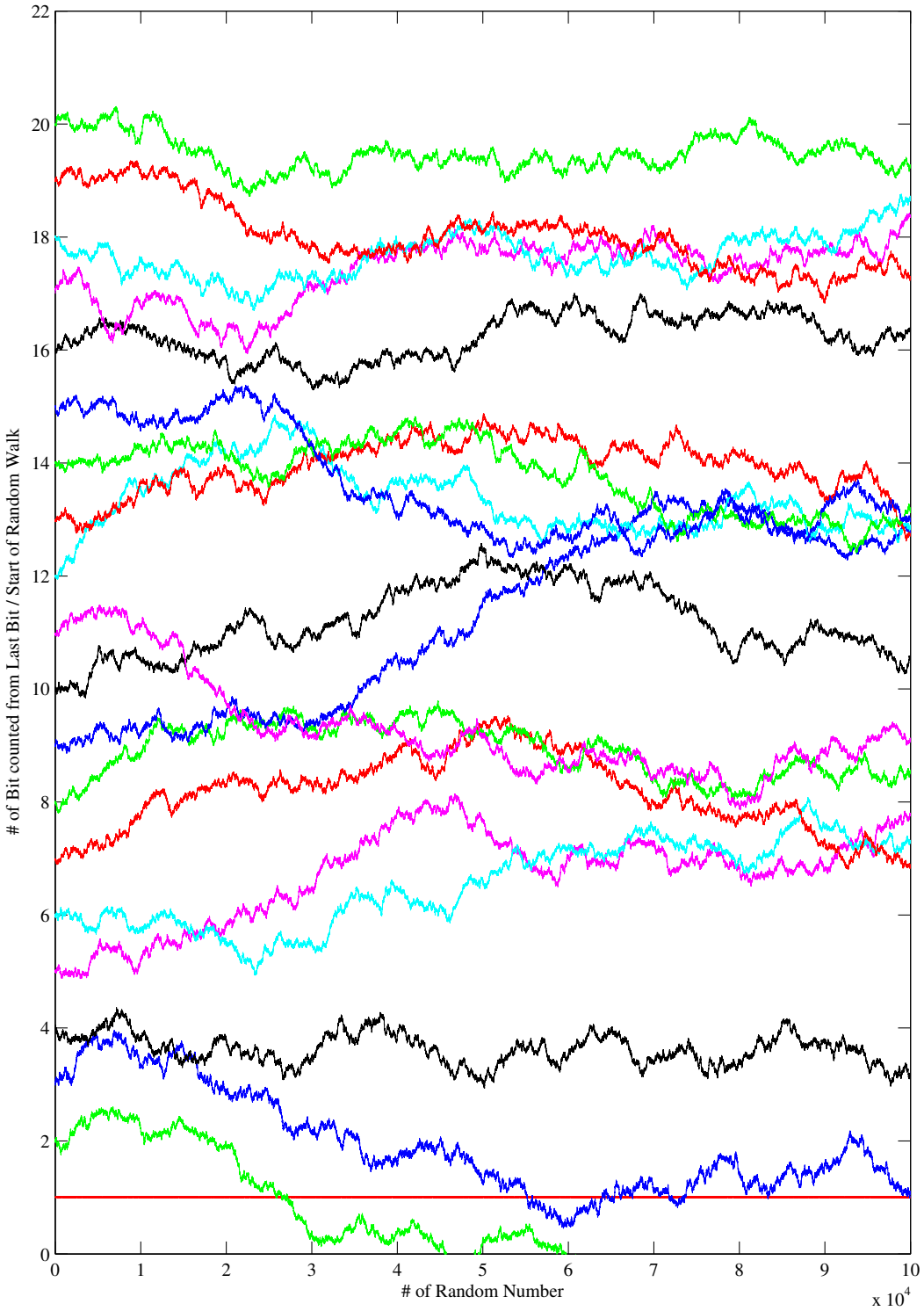

Figure B.18: Binary random walks for the lowest 20 bits of random numbers generated with the **HRC** RNG. The seed is 45.

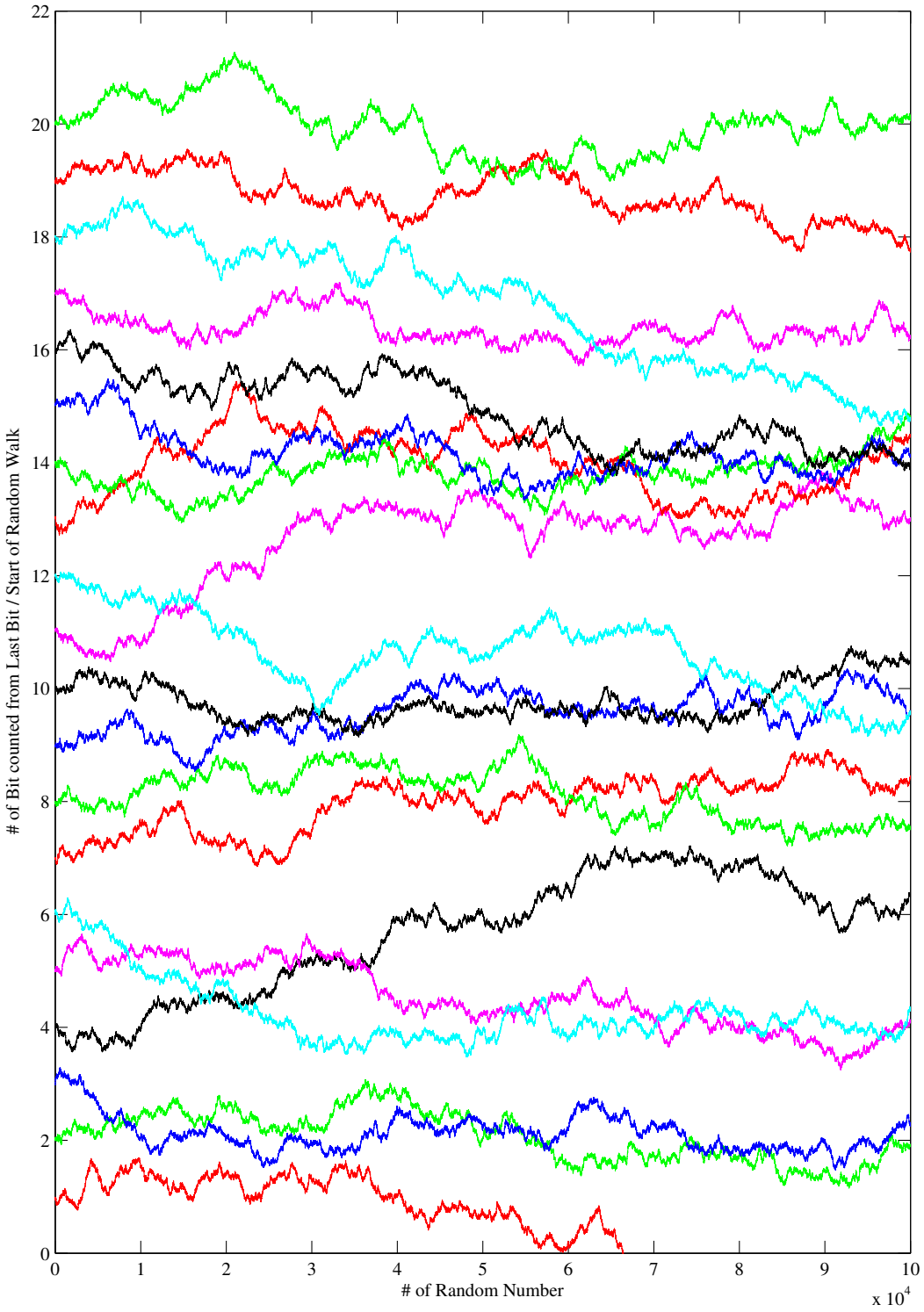

Figure B.19: Binary random walks for the lowest 20 bits of random numbers generated with the **STD** RNG. The seed is 45.

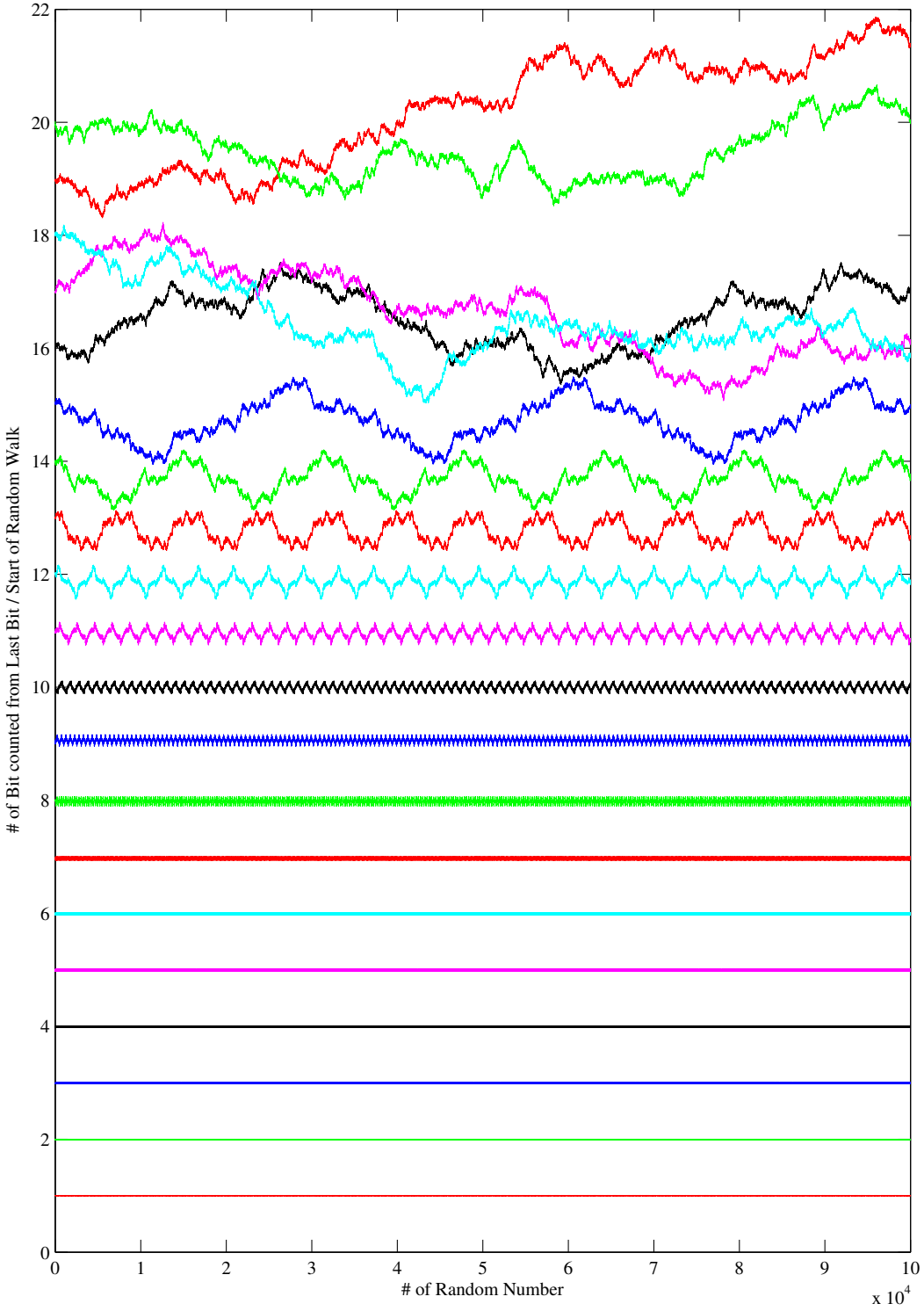

Figure B.20: Binary random walks for the lowest 20 bits of random numbers generated with the **GSL** RNG. The seed is 45.

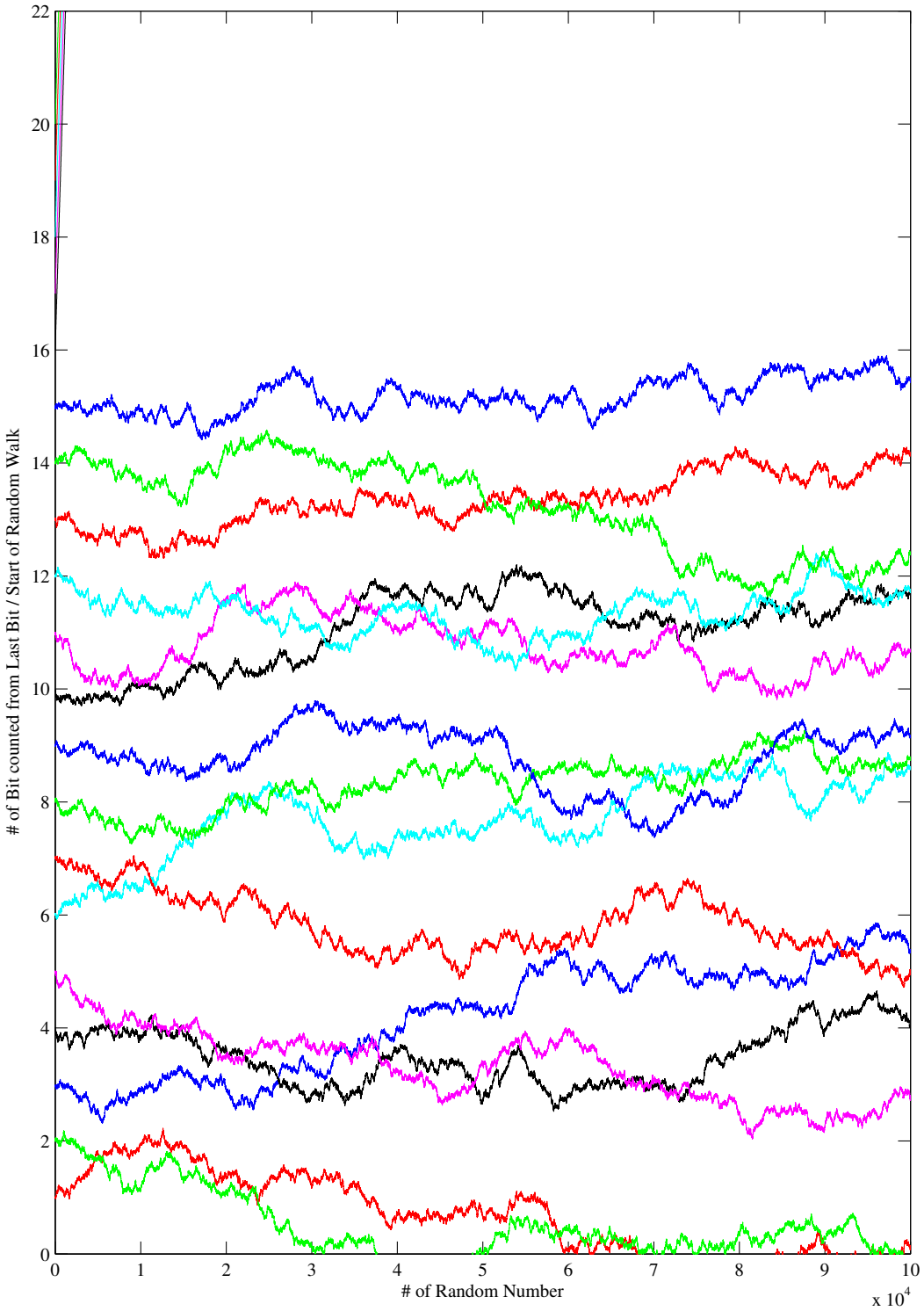

Figure B.21: Binary random walks for the lowest 20 bits of random numbers generated with the **SUN** RNG. The seed is 45.

Appendix C

Derivation of Formula for Trapezoidal Integration

Here, I derive Eq. (2.26). Note that $p \in \mathbb{N} \cup \{0\}$.

$$\prod_{i=2}^{d} \int_0^{x_{i-1}} \mathrm{d}x_i \, x_d^p \left(\sum_{j=1}^{d-1} \alpha_j x_j \right) = \prod_{i=2}^{d-1} \int_0^{x_{i-1}} \mathrm{d}x_i \left(\frac{x_{d-1}^{p+2} \, \alpha_{d-1}}{p+1} + \frac{x_{d-1}^{p+1}}{p+1} \sum_{j=1}^{d-2} \alpha_j x_j \right) \tag{C.1}$$

$$= \frac{x_1^{p+d} \, (p+2) \, p! \, \alpha_{d-1}}{(p+d)!} + \frac{1}{p+1} \prod_{i=2}^{d-1} \int_0^{x_{i-1}} \mathrm{d}x_i \, x_{d-1}^{p+1} \left(\sum_{j=1}^{d-2} \alpha_j x_j \right) \tag{C.2}$$

$$\overset{(*)}{=} \frac{x_1^{p+d} \, p!}{(p+d)!} \sum_{j=1}^{d-1} \alpha_j \, (p+1+d-j) \tag{C.3}$$

Recursion was used at (*).

Bibliography

[1] Assignment 3 of the course "Scientific Computing", B.F. Auer (2011), http://www.staff.science.uu.nl/~faggi101/labsci/site/ga_assign.pdf

[2] Script "Laboratory Class Scientific Computing", B.F. Auer, A.-J. Yzelman, A. van Dam, A. Swart, (2011)

[3] "Critical Values of the Chi-Square Distribution", NIST (2011), http://itl.nist.gov/div898/handbook/eda/section3/eda3674.htm