John von Neumann Institute for Computing

Pseudo Random Numbers: Generation and Quality Checks

Wolfhard Janke

published in

Quantum Simulations of Complex Many-Body Systems: From Theory to Algorithms, Lecture Notes,
J. Grotendorst, D. Marx, A. Muramatsu (Eds.),
John von Neumann Institute for Computing, Jülich,
NIC Series, Vol. **10**, ISBN 3-00-009057-6, pp. 447-458, 2002.

http://www.fz-juelich.de/nic-series/volume10

Pseudo Random Numbers: Generation and Quality Checks

Wolfhard Janke

Institut für Theoretische Physik, Universität Leipzig
Augustusplatz 10/11, 04109 Leipzig, Germany
E-mail: wolfhard.janke@itp.uni-leipzig.de

Monte Carlo simulations rely on the quality of pseudo random numbers. Some of the most common algorithms for the generation of uniformly distributed pseudo random numbers and a few physically motivated quality checks are described. Non-uniform distributions of random numbers are also briefly discussed.

1 Introduction

Molecular dynamics and Monte Carlo simulations are important numerical techniques in classical and quantum statistical physics.[1,2] Being a stochastic method, Monte Carlo simulations rely heavily on the use of random numbers. Other important areas making use of random numbers include stochastic optimization techniques and cryptography. In practice, random numbers are generated by deterministic recursive rules, formulated in terms of simple arithmetic operations. Obviously the emerging numbers can at best be pseudo random, and it is a great challenge to design random number generators that approximate "true randomness" as closely as possible. Besides this obvious requirement, pseudo random number generators should yield reproducible results, should be portable between different computer architectures, and should be as efficient as possible since in most applications many millions of random numbers are needed.

There is by now a huge literature on this topic,[3–9] and a simple search in the World-Wide-Web yields hundreds of useful links. The purpose of these lecture notes is to give a brief introduction into the most commonly used pseudo random number generators and to describe a few of the quality checks performed on them which are particularly relevant for Monte Carlo simulation studies.

2 Pseudo Random Number Generators

2.1 Linear Congruential Generators

Among the simplest algorithms for pseudo random numbers are the linear congruential generators[10] which are based on the integer recursion

$$X_{i+1} = (aX_i + c) \bmod m \ , \tag{1}$$

where the integers a, c and m are constants. These generators can be further classified into mixed ($c > 0$) and multiplicative ($c = 0$) types, usually denoted by LCG(a, c, m) and MLCG(a, m), respectively. A LCG generates a sequence of pseudo random integers $X_1, X_2, \ldots$ between 0 and $m - 1$; for a MLCG the lower bound is 1. Each X_i is then

scaled into the interval [0,1). If the multiplier a is a primitive root modulo m and m is prime, the period of this generator is $m-1$.

A commonly used choice of parameters for the MLCG is the miracle number $a = 16\,807 = 7^5$ and $m = 2^{31} - 1$. This yields the GGL generator[11] (sometimes also denoted by CONG or RAN0[12])

$$X_{i+1} = (16\,807 X_i) \bmod (2^{31} - 1) \ , \tag{2}$$

which has extensively been used on IBM computers. The period of $2^{31} - 2 \approx 2.15 \times 10^9$ is relatively short, however, and can easily be exhausted in present day simulations (100 000 Monte Carlo sweeps of a 100×100 lattice consist of 10^9 spin updates). Another known problem of this generator is that D-dimensional vectors $(x_1, x_2, \ldots, x_D)$, $(x_{D+1}, x_{D+2}, \ldots, x_{2D})$, ... formed by consecutive normalized pseudo random numbers $x_i \in [0, 1)$ lie on a relatively small number of parallel hyperplanes. As will be shown in the next section, this is already clearly visible in the smallest non-trivial case $D = 2$.

Also the generator G05FAF of the NAG software package[13] employs a multiplicative linear congruential algorithm MLCG(13^{13}, 2^{59}) or

$$X_{i+1} = (13^{13} X_i) \bmod 2^{59} \ , \tag{3}$$

which apart from the much longer period of $2^{59} - 1 \approx 5.76 \times 10^{17}$ has on vector computers the technical advantage that a vector of n pseudo random numbers agrees exactly with n successive calls of the G05FAF subroutine.

A frequently used MLCG is RANF,[14,15] originally implemented on (64 bit) CDC computers and later taken as the standard random number generator on CRAY vector and T3D or T3E parallel computers. This generator uses two linear congruential recursions with modulus 2^{48}, i.e., it is a combination of MLCG(M_1, 2^{48}) and MLCG(M_{64}, 2^{48}) with

$$X_{i+1} = (M_1 X_i) \bmod 2^{48} \ , \tag{4}$$

$$X_{i+64} = (M_{64} X_i) \bmod 2^{48} \ , \tag{5}$$

where $M_1 = 44\,485\,709\,377\,909$ and $M_{64} = 247\,908\,122\,798\,849$. The period length of RANF is[16] $2^{46} \approx 7.04 \times 10^{14}$ which is already long enough for most applications.

Another popular choice of parameters yields the generator RAND = LCG(69 069, 1, 2^{32}) with a rather short period of $2^{32} \approx 4.29 \times 10^9$, i.e., the recursion

$$X_{i+1} = (69\,069 X_i + 1) \bmod 2^{32} \ , \tag{6}$$

whose lattice structure is improved for small dimensions but also becomes poor for higher dimensions ($D \geq 6$). The multiplier 69 069 was strongly recommended by Marsaglia[17] and is part of the so-called SUPER-DUPER generator[14] which explains its popularity.

2.2 Lagged Fibonacci Generators

To increase the rather short period of linear congruential generators, it is natural to generalize them to the form

$$X_i = (a_1 X_{i-1} + \cdots + a_r X_{i-r}) \bmod m \ , \tag{7}$$

with $r > 1$ and $a_r \neq 0$. The maximum period is then $m^r - 1$. The special choice $r = 2$, $a_1 = a_2 = 1$ leads to the Fibonacci generator

$$X_i = (X_{i-1} + X_{i-2}) \bmod m \ , \tag{8}$$

whose properties are, however, relatively poor. This has led to the introduction of lagged Fibonacci generators which are initialized with r integers $X_1, X_2, \ldots, X_r$. Similar to (8) one then uses the recursion

$$X_i = (X_{i-r} \otimes X_{i-s}) \bmod m \ , \tag{9}$$

where $s < r$ and $\otimes$ stands short for one of the binary operations $+$, $-$, $\times$, or the exclusive-or operation $\oplus$ (XOR). These generators are denoted by LF($r, s, m, \otimes$). For the usually used addition or subtraction modulo 2^w (w is the word length in bits) the maximal period with suitable choices of r and s is $(2^r - 1)2^{w-1} \approx 2^{r+w-1}$.

An important example is RAN3 = LF($55, 24, m, -$) or

$$X_i = (X_{i-55} - X_{i-24}) \bmod m \ , \tag{10}$$

where for instance in the Numerical Recipes implementation[18] $m = 10^9$ is used. The period length of this specific generator is known[19] to be $2^{55} - 1 \approx 3.60 \times 10^{16}$.

2.3 Shift Register Generators

Another important class of pseudo random number generators are provided by generalized feedback shift register algorithms[20] which are sometimes also called Tausworthe[21] generators. They are based on the theory of primitive trinomials of the form $x^p + x^q + 1$, and are denoted by GFSR($p, q, \oplus$), where $\oplus$ stands again for the exclusive-or operation XOR, or in formulas

$$X_i = X_{i-p} \oplus X_{i-q} \ . \tag{11}$$

The maximal possible period of $2^p - 1$ of this generator is achieved when the primitive trinomial $x^p + x^q + 1$ divides $x^n - 1$ for $n = 2^p - 1$, but for no smaller value of n. This condition can be met by choosing p to be a Mersenne prime, that is a prime number p for which $2^p - 1$ is also a prime.

A standard choice for the parameters is $p = 250$ and $q = 103$. This is the (in)famous (see below) R250 generator GFSR($250, 103, \oplus$) or

$$X_i = X_{i-250} \oplus X_{i-103} \ . \tag{12}$$

The period of R250 is $2^{250} - 1 \approx 1.81 \times 10^{75}$. In one of the earliest implementations[22] the MLCG($16\,807, 2^{31} - 1$), i.e. the GGL recursion (2), was used to initialize the first 250 integers, but many different approaches for the initialization are possible and were indeed used in the literature (a fact which complicates comparisons).

2.4 Combined Algorithms

If done carefully, the combination of two different generators may improve the performance. An often employed generator based on this construction is the RANMAR generator.[23,24] In the first step it employs a lagged Fibonacci generator,

$$X_i = \begin{cases} X_{i-97} - X_{i-33} \ , & \text{if } X_{i-97} \geq X_{i-33} \ , \\ X_{i-97} - X_{i-33} + 1 \ , & \text{otherwise} \ . \end{cases} \tag{13}$$

Only 24 most significant bits are used for single precision reals. The second part of the generator is a simple arithmetic sequence for the prime modulus $2^{24} - 3 = 16\,777\,213$,

$$Y_i = \begin{cases} Y_i - c \ , & \text{if } Y_i \geq c \ , \\ Y_i - c + d \ , & \text{otherwise} \ , \end{cases} \tag{14}$$

where $c = 7\,654\,321/16\,777\,216$ and $d = 16\,777\,213/16\,777\,216$. The final random number Z_i is then produced by combining the obtained X_i and Y_i as

$$Z_i = \begin{cases} X_i - Y_i \ , & \text{if } X_i \geq Y_i \ , \\ X_i - Y_i + 1 \ , & \text{otherwise} \ . \end{cases} \tag{15}$$

The total period of RANMAR is about[23] $2^{144} \approx 2.23 \times 10^{43}$. This generator has become very popular in high-statistics Monte Carlo simulations.

2.5 Marsaglia-Zaman Generator

The Marsaglia-Zaman generator is based on the so-called "subtract-and-borrow" algorithm.[25,26] It is similar to the lagged Fibonacci generator but supplemented with an extra carry bit. If X_i and b are integers with

$$0 \leq X_i \leq b \ , \qquad i < n \ , \tag{16}$$

then the recursion involving two lags p and q works according to the following prescription. For $n \geq q$ one first computes the difference

$$\Delta_n = X_{n-p} - X_{n-q} - c_{n-1} \ , \tag{17}$$

where c_{n-1} = 0 or 1 is the carry bit. One then determines X_n and c_n through

$$X_n = \begin{cases} \Delta_n \ , & c_n = 0 \ , \quad \text{if } \Delta_n \geq 0 \ , \\ \Delta_n + b \ , & c_n = 1 \ , \quad \text{if } \Delta_n < 0 \ . \end{cases} \tag{18}$$

To start the recursion, the first q values $X_0, X_1, \ldots, X_{q-1}$ together with the carry bit c_{q-1} must be initialized. The generator with the particular choice of parameters $b = 2^{24}$, $p = 24$, and $q = 10$ is known[24] under the name RCARRY.[a] It has the tremendously long period of[24,28] $(2^{24})^{24}/48 \approx 2^{570}$ or about 5.15×10^{171}.

2.6 The "Luxury" RANLUX Generator

Also for the RCARRY generator some deficiencies in empirical tests of randomness were reported in the literature.[5] By analysing Fibonacci generators from the viewpoint of multi-dimensional chaos, Lüscher[28] showed that there was slow and hence poor divergence from nearby initial points. Based on these results, James[29,30] implemented the so-called "luxury" pseudo random number generator RANLUX, in which a certain number of points is discarded following each pass through the recirculation buffer. If the "luxury" level is LUX=0, no points are skipped (and RANLUX runs as RCARRY), if LUX=1, then after 24 values have been returned, 24 more values are discarded. Similarly, for LUX=2, 3,

[a] Notice that the FORTRAN code for this algorithm printed in Ref. 24 contains a typo[27]: In the line UNI=SEEDS(I24)−SEEDS(J24)−CARRY, the indices I24 and J24 should be interchanged.

and 4, the routine discards 73, 199, and 365 values, respectively. In the high "luxury" levels this generator is rather slow, but the quality of the resulting pseudo random numbers is considered to be extremely good. RANLUX has therefore become very popular in the Monte Carlo community, in particular for computationally demanding problems where only a small portion of the total computing time is spent on the generation of pseudo random numbers (such as for instance lattice QCD with dynamical fermions) and for testing purposes. Recently it has been coded in PC assembler language[31] and the FORTRAN90 versions have also been accelerated by conversion to integer arithmetic.[32]

3 Quality Checks

The evaluation of the quality of pseudo random numbers is a difficult problem which has no unique solution. On the one hand there is no single practical test that can verify the realization of randomness in a given pseudo random number sequence. On the other hand, since all pseudo random number generators are based on deterministic rules, there exists always a test in which a given generator will fail. Given this situation one can only try to find some criteria which test at least the most fundamental properties of such "random" sequences. There is a well standardized set of statistical tests[19] such as the uniformity test, the serial test, the gap test, the maximum t test, the collision test, the run test and the park test, bit level tests, the spectral test and visual tests which are well described in the comparative study of many random number generators by Vattulainen *et al.*[5,6]

Among the most impressive visual tests is the search for lattice structures when consecutive pseudo random numbers are plotted as D-tupels. As can be inspected in Fig. 1, for a multiplicative linear congruential pseudo random number generator a lattice structure is already clearly visible in the smallest non-trivial case $D = 2$ when a small portion of, say, the x-axis is magnified. For this plot a sequence of 10^7 random numbers was generated with the GGL recursion (2), starting with the "pi"-seed $X_0 = 314\,159$. This gave a mean

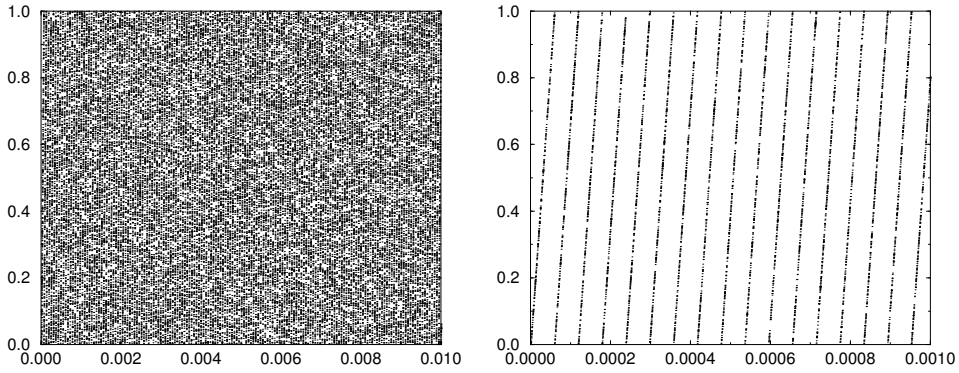

Figure 1. The two-dimensional distribution of 5×10^6 pseudo random number pairs (x_i, x_{i+1}) generated with the GGL recursion (2). While on the scale of the plot on the l.h.s. the distribution "looks" random, on the expanded scale of the plot on the r.h.s. an ordered line structure is clearly revealed.

value of $\overline{x} = 0.499\,795 \approx 1/2$ and a variance of $\sigma^2 = \overline{x^2} - \overline{x}^2 = 0.080\,08 \approx 1/12$. The failure of the MLCG can easily be highlighted by intentionally using a "poor" choice of parameters, for instance[33] $X_{i+1} = (5X_i) \bmod 2^7$, $X_0 = 1$, where the problem becomes immediately obvious.

When subjected to the various mathematical tests mentioned above, the R250 shift register generator turned out to be among the best generators. Consequently it has been used in many Monte Carlo studies. It, therefore, came as a surprise when Ferrenberg *et al.*[34] reported severe problems with this generator in applications to Monte Carlo simulations of the two-dimensional Ising model using the single-cluster Wolff update algorithm.[35] More precisely they performed simulations of a 16×16 square lattice with periodic boundary conditions at the exactly known infinite-volume transition point $\beta_c = \ln(1+\sqrt{2})/2$. They generated 10 runs with 10^6 clusters each (which, on the average, cover at β_c about 55% of the lattice sites). As a result Ferrenberg *et al.* obtained for the energy a systematic deviation from the exact value[36] of about 42σ (at an accuracy level of 0.003%) and for the specific heat an even larger deviation of about -107σ (at an accuracy level of 0.03%). Further simulations with the same statistics but other pseudo random number generators behaved perfectly well.

Subsequently these results have been confirmed by many other authors,[37] and consensus has been reached that triplet correlations in $\langle x_n x_{n-k} x_{n-250} \rangle$ around $k = 147$ are the origin of the problem.[38] Notice that due to "time-reversal symmetry" this value is equivalent[39] to $k = 250 - 147 = 103$ – a value that just coincides with the second parameter $q = 103$ of the R250 generator! While the numerically determined correlator[38] reproduces the theoretically expected value of $(1/2)^3 = 0.125$ for almost all values of k, it drops down to about 0.107 at $k = 147$ or equivalently[39] at $k = 103$. This correlation does not only affect the cluster algorithm but also the Metropolis update which was shown[38] to fail in combination with R250 at the tricritical point of the Blume-Capel model for some "resonant" lattice sizes.

By analyzing the recursion of R250 analytically, Heuer *et al.*[39] succeeded to predict an anomalous triplet correlation of $3/28 \approx 0.107\,142\,857\,1\ldots$ at the special value $k = 103$, in perfect agreement with the numerical observation (which they also reconfirmed). Equipped with this finding they were able to propose a modified version of R250 denoted as R250/521, which avoids these triplet correlations and indeed shows a much improved performance. For the two-dimensional Ising model on a 16×16 lattice at β_c they obtained with the R250/521 generator for the energy and specific heat systematic deviations of absolutely tolerable (and in fact, expected) 0.1σ and 1.5σ, respectively, for a set-up of the single-cluster Monte Carlo simulations which was otherwise equivalent to that of Ref. 34. This remarkable improvement indicates that the triplet correlations are very probably responsible for the systematic errors observed by Ferrenberg *et al.*[34]

Around the same time, Shchur and Blöte[40] performed a systematic study of this problem by varying the size L of the square lattice as well as the "magic numbers" (p, q), investigating the four pairs $(p, q) = (36, 11)$, $(89, 38)$, $(127, 64)$, and $(250, 103)$. For all pairs they found at β_c significant deviations from the exact result with a maximum for $L = 7$, 12, 15, and 22, respectively. Since the average cluster size $\langle C \rangle$ of the single-cluster Wolff update algorithm is an (improved) estimator[41] for the susceptibility of Ising models, it scales with lattice size according to $\langle C \rangle = aL^{\gamma/\nu}$, which in two dimensions specializes to $\langle C \rangle = aL^{7/4}$. The coefficient a is non-universal, and for the square geometry it takes the

generator	periodic b.c.		anti-periodic b.c.	
	e	c	e	c
R250	1.414 087(63)	1.356 9(16)	1.214 173(47)	0.704 52(85)
	-2.0σ	$+19.4\sigma$	-1.6σ	$+7.5\sigma$
R250/521	1.414 206(59)	1.325 5(16)	1.214 277(47)	0.698 17(86)
	-0.14σ	-0.27σ	$+0.6\sigma$	$+0.06\sigma$
RANLUX	1.414 253(60)	1.326 9(13)	1.214 215(48)	0.698 08(82)
	$+0.65\sigma$	$+0.75\sigma$	-0.7σ	-0.05σ
exact/average	1.414 213 6	1.325 927 9	1.214 247(34)	0.698 08(82)

Table 1. Comparison of the energy e and specific heat c for a 10×192 Ising system as obtained in single-cluster Monte Carlo simulations using three different pseudo random number generators. The deviations from the exact (periodic b.c.) respectively average (anti-periodic b.c.) values are given in units of the standard deviation σ, indicated in parentheses behind the mean values.

numerically determined value of $a \approx 1.1$. Using this value in the above finite-size scaling formula one easily derives $\langle C \rangle \approx 33$, 85, 126, $141 = 0.55 \times 256$, and 246 for $L = 7$, 12, 15, 16, and 22, respectively. By comparing with the parameter p one thus concludes that the largest deviations from the exact results happen when the average cluster size coincides with p.

Also for asymmetric, strip-like lattices of size 10×192 the failure of the R250 generator in combination with the single-cluster update algorithm was observed.[42] Here periodic boundary conditions (b.c.) were applied in the long direction and both, periodic as well as anti-periodic b.c., in the short direction. The results shown in Table 1 are based on 2×10^6 measurements at β_c. Here the deviations of the energy and specific heat from the exact results are less pronounced than in Refs. 34 and 40. This is, however, consistent with the observation in Ref. 40, since the average cluster sizes, $\langle C \rangle \approx 159$ and $\langle C \rangle \approx 83$ for periodic and anti-periodic b.c., respectively, are relatively far away from the lag $p = 250$. The modified generator R250/521 as well as RANLUX (in "luxury" level LUX = 4), on the

generator	periodic b.c.		anti-periodic b.c.	
	ξ_e	ξ_σ	ξ_e	ξ_σ
R250	1.5796(46)	12.7092(92)	0.7803(84)	4.2878(43)
	-0.3σ	$+3.3\sigma$	-3.8σ	-0.7σ
R250/521	1.5851(48)	12.6787(74)	0.8110(82)	4.2925(39)
	$+0.8\sigma$	-0.02σ	-0.1σ	$+0.5\sigma$
RANLUX	1.5770(50)	12.6789(85)	0.8126(83)	4.2887(40)
	-0.8σ	$+0.006\sigma$	$+0.1\sigma$	-0.5σ
exact/average	1.5812(35)	12.678 845	0.8118(58)	4.2906(28)

Table 2. Same comparison as in Table 1 for the correlation lengths ξ_e and ξ_σ of energy and magnetization densities, respectively, in the long direction of a 10×192 Ising lattice.

other hand, performed very well. When using the R250 generator, small but still significant deviations were also observed[42] for the correlation lengths of the energy and spin densities (measured using the zero-momentum technique[43]), cf. Table 2.

Another simulational test based on Schwinger-Dyson identities has been proposed by Ballesteros and Martín-Mayor.[44] Applications to the two- and three-dimensional Ising model confirmed the flaws in two dimensions reported earlier and showed that also in three dimensions the combination of R250 with the single-cluster update algorithm produces incorrect results.

4 Non-Uniform Pseudo Random Numbers

All basic pseudo random number generators discussed above are designed for uniformly distributed pseudo random numbers $x_i \in [0,1)$. In many applications it is necessary, however, to be able to draw pseudo random numbers from non-uniform distributions.[45] One strategy is to divide this problem into two parts. First, one of the generators described above is used to generate uniformly distributed random numbers, which in a second step are appropriately transformed to follow the specific distribution at hand.

A standard procedure is the inversion method. For a given normalized probability density $f(x)$ one calculates the associated probability distribution (accumulated density in usual physics terms),

$$F(x) = \int_{x_{\min}}^{x} dx' f(x') \ . \tag{19}$$

Due to $F'(x) = f(x) \geq 0$ and the normalization condition, $F(x)$ grows monotonically from 0 to 1, such that the F values are uniformly distributed. Drawing a uniformly distributed pseudo random number R, equating $R = F(x)$ and, if the function inverse is known analytically, setting $x = F^{-1}(R)$ the problem is solved.

For the example of an exponential decay,

$$f(x) = \exp(-x) \ , \qquad x \geq 0 \ , \tag{20}$$

one derives in this way $R = F(x) = 1 - \exp(-x)$ or

$$x = -\ln(1-R) \ , \qquad R \in [0,1) \ . \tag{21}$$

Notice that since $R \in [0,1)$, the formula should be programed in the "complicated" way as shown here; rewriting it as $x = -\ln(R)$ one could occasionally hit $\ln 0$ which would cause a run-time error (with a reaction depending on the operating system used).

Another simple example is the Lorentzian density

$$f(x) = \frac{1}{\pi} \frac{\Gamma}{\Gamma^2 + x^2} \ , \tag{22}$$

where Γ parameterizes the width of the Lorentzian peak. Here one calculates $R = F(x) = \frac{1}{\pi} \int_{-\infty}^{x} dx' \frac{\Gamma}{\Gamma^2 + x'^2} = \frac{1}{2} + \frac{1}{\pi} \tan^{-1}(\frac{x}{\Gamma})$, which can be inverted to give

$$x = \Gamma \tan\left[\pi(R - 1/2)\right] \ , \qquad R \in [0,1) \ . \tag{23}$$

The final and most important example are Gaussian random numbers which follow the probability density (... no problem to remember if you saved a German 10 DM note ...)

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{x^2}{2\sigma^2}\right) \ , \tag{24}$$

where the parameter σ^2 is the squared width of the distribution. Here

$$R = F(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \int_{-\infty}^{x} dx' \exp\left(-\frac{x'^2}{2\sigma^2}\right) = \frac{1}{2}\left[1 + \mathrm{erf}\left(\frac{x}{\sqrt{2\sigma^2}}\right)\right] \ , \tag{25}$$

with $\mathrm{erf}(\cdot)$ denoting the error function which cannot be inverted analytically. Either one could think of numerical inversion schemes (which indeed is the method of choice for really complicated probability distributions) – or one remembers the polar coordinates trick used to calculate the Gaussian integral and continues analytically: Considering the auxiliary two-dimensional product distribution $f_2(x,y) = f(x)f(y) = \frac{1}{2\pi\sigma^2}\exp(-\frac{x^2+y^2}{2\sigma^2})$ and introducing polar coordinates $x = r\cos(\Theta)$, $y = r\sin(\Theta)$, one obtains

$$f_2(x,y)dxdy = \frac{1}{\sigma^2}\exp(-\frac{r^2}{2\sigma^2})rdr\frac{d\Theta}{2\pi} \ , \tag{26}$$

showing immediately that the angle Θ is uniformly distributed between 0 and 2π. Also for the radial coordinate r the inversion is now straightforward since

$$R_1 = F(r) = \frac{1}{\sigma^2}\int_0^r dr' r' \exp\left(-\frac{r'^2}{2\sigma^2}\right) = 1 - \exp\left(-\frac{r^2}{2\sigma^2}\right) \ . \tag{27}$$

We thus arrive at the so-called Box-Müller method: Draw two uniformly distributed random numbers R_1 and R_2, and compute

$$r = \sqrt{-2\sigma^2\ln(1-R_1)} \ , \qquad R_1 \in [0,1) \ , \tag{28}$$

$$\Theta = 2\pi R_2 \ , \qquad R_2 \in [0,1) \ . \tag{29}$$

Then

$$x = r\cos(\Theta) \ , \tag{30}$$

$$y = r\sin(\Theta) \ , \tag{31}$$

is a pair of two independent Gaussian distributed pseudo random numbers.

Especially for Gaussian random numbers there is another procedure which directly makes use of the central limit theorem and the fact that averages of arbitrarily distributed random numbers (under certain rather mild conditions) tend asymptotically to a Gaussian distribution. In practice one uses, of course, again uniformly distributed pseudo random numbers x_i generated with one of the algorithms described in the previous section. Recalling the mean value $\overline{x} = 1/2$ and variance $\sigma^2 = \overline{x^2} - \overline{x}^2 = 1/12$ for uniform random numbers, it is straightforward to see that

$$X = \left(\sum_{i=1}^{n} x_i - \frac{n}{2}\right)\sqrt{\frac{12}{n}} \tag{32}$$

is (approximately) a Gaussian distributed random number around $X = 0$ with unit variance. Of course, since $X_{\max} = -X_{\min} = \sqrt{3n}$, this can be strictly true only asymptotically as $n \to \infty$. But even the convenient choice $n = 12$ leads already to a reasonable approximation[46] in the range $|X| < 2 = 2\sigma$ with errors less than 9×10^{-3}.

Another, physically motivated direct method based on simulating N molecules has recently been discussed by Fernández and Criado.[47]

5 Summary

The generation of "good" pseudo random numbers is quite a delicate issue that requires some care and extensive quality tests. It is therefore highly recommended not to invent ones own "secret" recursion rules but to use one of the well-known generators which have been tested and applied by many other workers in the field. If such a well-accepted generator would turn out to be problematic in some specific application, one could at least be sure that the Monte Carlo community as a whole would work hard to track the origin of the problem – as it has happened with the (in)famous R250 generator. Being based on deterministic recursion rules, it is trivial that for any pseudo random number generator one can design a test where it would fail. Thanks to the by now available quite sophisticated mathematical and physically motivated empirical tests one can be very confident, however, that standard generators will yield sufficiently "random" numbers in most applications.

Acknowledgments

I would like to thank Tilman Sauer, Andreas Weber and Martin Weigel for pseudo random, but very useful discussions on randomly selected topics relevant for these lecture notes.

References

1. D. Frenkel and B. Smit, *Understanding Molecular Simulation – From Algorithms to Applications* (Academic Press, San Diego, 1996).
2. D.P. Landau and K. Binder, *A Guide to Monte Carlo Simulations in Statistical Physics* (Cambridge University Press, Cambridge, 2000).
3. F. Gutbrod, in: *Annual Review of Computational Physics VI*, ed. D. Stauffer (World Scientific, Singapore, 1999), p. 203.
4. D. Stauffer, in: *Computational Physics: Selected Methods – Simple Exercises – Serious Applications*, eds. K.H. Hoffmann and M. Schreiber (Springer, Berlin, 1996), p. 1.
5. I. Vattulainen, K. Kankaala, J. Saarinen, and T. Ala-Nissila, Comp. Phys. Comm. **86**, 209 (1995).
6. I Vattulainen, Licentiate in Technology thesis, University of Helsinki (1994) [cond-mat/9411062].
7. K. Kankaala, T. Ala-Nissila, and I. Vattulainen, Phys. Rev. **E48**, R4211 (1993).
8. I. Vattulainen, T. Ala-Nissila, and K. Kankaala, Phys. Rev. Lett. **73**, 2513 (1994).
9. I. Vattulainen, T. Ala-Nissila, and K. Kankaala, Phys. Rev. **E52**, 3205 (1995).
10. D.H. Lehmer, in: *Proc. 2nd Symp. on Large-Scale Digital Calculating Machinery* (Harvard University Press, Cambridge, 1951), p. 141.
11. S.K. Park and K.W. Miller, Comm. ACM **31**, 1192 (1988).
12. W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery, *Numerical Recipes in Fortran 77 – The Art of Scientific Computing*, second corrected edition (Cambridge University Press, Cambridge, 1996), pp. 269-270.

13. *NAG Fortran Library Manual, Mark 14*, **7** (Numerical Algorithms Group Inc., 1990).
14. S.L. Anderson, SIAM Review **32**, 221 (1990).
15. G.S. Fishman, Math. Comp. **54**, 331 (1990).
16. A. De Matteis and S. Pagnutti, Parallel Computing **13**, 193 (1990).
17. G. Marsaglia, in: *Applications of Number Theory to Numerical Analysis*, ed. S.K. Zaremba (Academic Press, New York, 1972), p. 249.
18. W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery, in Ref. 12, p. 273.
19. D.E. Knuth, *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*, second edition (Addison-Wesley, Reading, Massachusetts, 1981).
20. T.G. Lewis and W.H. Payne, J. Assoc. Comput. Mach. **20**, 456 (1973).
21. R.C. Tausworthe, Math. Comp. **19**, 201 (1965).
22. S. Kirkpatrick and E.P. Stoll, J. Comp. Phys. **40**, 517 (1981).
23. G. Marsaglia and A. Zaman, Stat. & Prob. Lett. **8**, 329 (1990).
24. F. James, *A Review of Pseudorandom Number Generators*, Comp. Phys. Comm. **60**, 329 (1990).
25. G. Marsaglia, B. Narasimham, and A. Zaman, Comp. Phys. Comm. **60**, 345 (1990).
26. G. Marsaglia and A. Zaman, Ann. Appl. Prob. **1**, 462 (1991).
27. Private communication (1993) of F. James to M. Lüscher (Ref. 28).
28. M. Lüscher, Comp. Phys. Comm. **79**, 100 (1994).
29. F. James, Comp. Phys. Comm. **79**, 111 (1994).
30. F. James, Comp. Phys. Comm. **97**, 357 (1996).
31. K.G. Hamilton, Comp. Phys. Comm. **101**, 249 (1997).
32. K.G. Hamilton and F. James, Comp. Phys. Comm. **101**, 241 (1997).
33. P. Blaudeck, in: *Computational Physics: Selected Methods – Simple Exercises – Serious Applications*, eds. K.H. Hoffmann and M. Schreiber (Springer, Berlin, 1996), p. 9.
34. A.M. Ferrenberg, D. P. Landau, and Y.J. Wong, Phys. Rev. Lett. **69**, 3382 (1992).
35. U. Wolff, Phys. Rev. Lett. **62**, 361 (1989); Nucl. Phys. **B322**, 759 (1989).
36. B. Kaufman, Phys. Rev. **76**, 1232 (1949); A.E. Ferdinand and M.E. Fisher, Phys. Rev. **185**, 832 (1969). For a Fortran code, see W. Janke, in: *Computational Physics: Selected Methods – Simple Exercises – Serious Applications*, eds. K.H. Hoffmann and M. Schreiber (Springer, Berlin, 1996), p. 10, and the accompanying diskette.
37. P.D. Coddington, Int. J. Mod. Phys. **C5**, 547 (1994) [cond-mat/9309017].
38. F. Schmid and N.B. Wilding, Int. J. Mod. Phys. **C6**, 781 (1995) [cond-mat/9512135].
39. A. Heuer, B. Dünweg, and A.M. Ferrenberg, Comp. Phys. Comm. **103**, 1 (1997).
40. L.N. Shchur and H.W.J. Blöte, Phys. Rev. **E55**, R4905 (1997) [cond-mat/9703050].
41. U. Wolff, Nucl. Phys. **B334**, 581 (1990).
42. M. Weigel, Diploma thesis, Universität Mainz (1998), unpublished.
43. M. Weigel and W. Janke, Phys. Rev. Lett. **82**, 2318 (1999); Phys. Rev. **B62**, 6343 (2000).
44. H.G. Ballesteros and V. Martín-Mayor, Phys. Rev. **E58**, 6787 (1998).
45. L. Devroye, *Non-Uniform Random Variate Generation* (Springer, Berlin, 1986).
46. M. Abramowitz and I.A. Stegun (eds.), *Handbook of Mathematical Functions*, 9th printing (Dover, New York, 1972), p. 953.
47. J.F. Fernández and C. Criado, preprint cond-mat/9901202.