Online services A website that delivers true random numbers is `random.org`. Although not very useful for large-scale simulations, the site delivers true random numbers (using atmospheric noise). There is a limit of free random bits. In general, the service costs approximately US$ 1 per 4 million random bits. A large-scale Monte Carlo simulation with 10^{12} random numbers would therefore cost US$ 250,000.

Final recommendation For any scientific applications avoid the use of linear congruential generators, the family of Unix built-in generators `drand48( )`, Numerical Recipe's `ran0( )`, `ran1( )` and `ran2( )`, as well as any home-cooked routines. Instead, use either a multiplicative lagged Fibonacci generator such as `r1279( )`, WELL generators, or the Mersenne Twister. Not only are they good, they are very fast.

4 Testing the quality of random number generators

In the previous chapters we have talked about "good" and "bad" generators mentioning often "statistical tests." There are obvious reasons why a PRNG might be bad: For example, with a period of 10^9 a generator is useless for most scientific applications. As in the case of `r250( )` [10], there can be *very* subtle effects that might bias data in a simulation. These subtle effects can often only be discovered by performing batteries of statistical tests that try to find hidden correlations in the stream of random numbers. In the end, our goal is to obtain pseudo random numbers that are like true random numbers.

Over the years many empirical statistical tests have been developed that attempt to determine if there are any short-time or long-time correlations between the numbers, as well as their distribution. Are these tests enough? No. As in the case of `r250( )`, your simulation could depend in a subtle way on hidden correlations. What is thus the ultimate test? *Run your code with different PRNGs. If the results agree within error bars and the PRNGs used are from different families, the results are likely to be correct.*

4.1 Simple PRNG tests

If your PRNG does not pass the following tests, then you should definitely not use it. The tests are based on the fact that if one assumes that the produced random numbers have no correlations, the error should be purely statistical and scale as $1/\sqrt{N}$ where N is the number of random numbers used for the test.

Simple correlations test For all $n \in \mathbb{N}$ calculate the following function

$$\varepsilon(N,n) = \frac{1}{N}\sum_{i=1}^{N} x_i x_{i+n} - E(x)^2\,, \tag{10}$$

where

$$E(x) = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{11}$$

represents the average over the sampled pseudo random numbers. If the tuplets of numbers are not correlated, $\varepsilon(N,n)$ should converge to zero with a statistical error for $N \to \infty$, i.e.,

$$\varepsilon(N,n) \sim \mathcal{O}(N^{-1/2}) \qquad \forall n \; . \tag{12}$$

Simple moments test Let us assume that the PRNG to be tested produces uniform pseudo random numbers in the interval $[0,1]$. One can analytically show that for a uniform distribution the k-th moment is given by $1/(k+1)$. One can therefore calculate the following function for the kth moment

$$\mu(N,k) = \left| \frac{1}{N}\sum_{i=1}^{N} x_i^k - \frac{1}{k+1} \right| \; . \tag{13}$$

Again, for $N \to \infty$

$$\mu(N,k) \sim \mathcal{O}(N^{-1/2}) \qquad \forall k \; . \tag{14}$$

Graphical tests Another simple test to look for "spatial correlations" is to group a stream of pseudo random numbers into k-tuplets. These tests are also known under the name "spectral tests." For example, a stream of numbers x_1, x_2, ... can be used to produce two-dimensional vectors $\vec{v}_1 = (x_1, x_2)$, $\vec{v}_2 = (x_3, x_4)$, ..., as well as three-dimensional or normalized unit vectors ($\vec{e} = \vec{x}/||\vec{x}||$). Figure 2 shows data for 2-tuplets and normalized 3-tuplets for both good and bad PRNGs. While the good PRNG shows no clear sign of correlations, these are evident in the bad PRNG.

Theoretical details on spectral tests, as well as many other methods to test the quality of PRNGs such as the chi-squared test, can be found in Ref. [14].

4.2 Test suites

There are different test suites that have been developed with the sole purpose of testing PRNGs. In general, it is recommended to use these to test a new generator as they are well established. Probably the oldest and most commonly used test suite is DIEHARD by George Marsaglia.

DIEHARD The software is freely available [5] and comprises 16 standard tests. Most of the tests in DIEHARD return a p-value, which should be uniform on the interval $[0,1)$ if the pseudo random numbers are truly independent random bits. When a bit stream fails the test, p-values near 0 or 1 to six or more digits are obtained (for details see Refs. [5] and [14]). DIEHARD includes the following tests (selection):

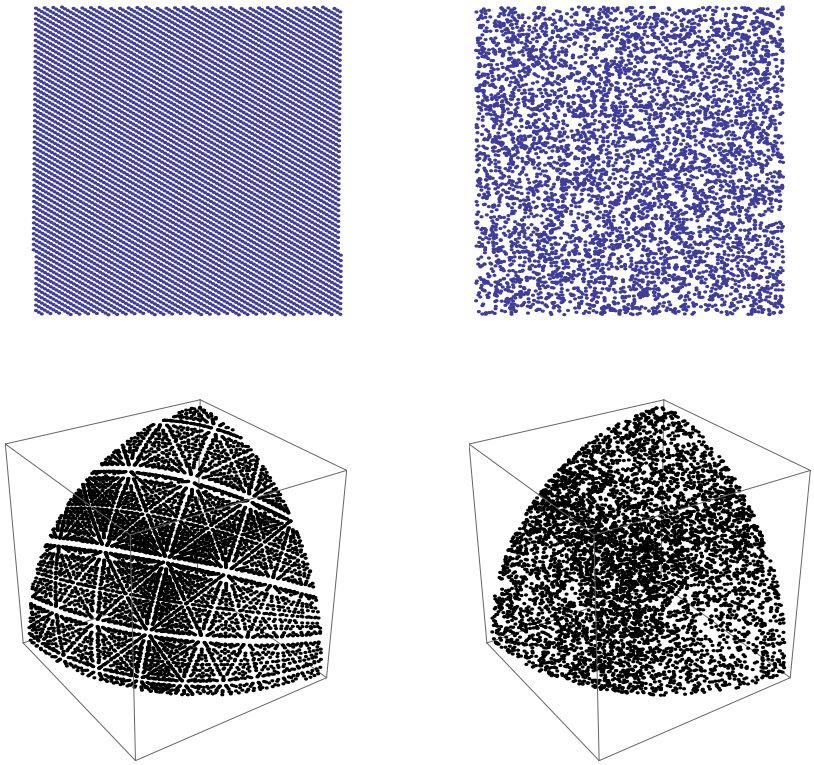

Figure 2: Graphical correlations test using a home-cooked linear congruential generator with $a = 106$, $c = 1283$ and $m = 6075$ (left panels) and the Mersenne Twister (right panels). The top row shows 2-tuplets in the plane. Correlations (left-hand-side) are clearly visible for the home-cooked PRNG. The bottom row shows 3-tuplets on the unit sphere. Again, the home-cooked PRNG (left-hand-side), albeit pretty, shows extreme correlations.

☐ Overlapping permutations test: Analyze sequences of five consecutive random numbers. The $5! = 120$ possible permutations should occur (statistically) with equal probability.

☐ Birthday spacings test: Choose random points on a large interval. The spacings between the points should be asymptotically Poisson-distributed.

☐ Binary rank test for 32×32 matrices: A random 32×32 binary matrix is formed. The rank is determined, it can be between 0 and 32. Ranks less than 29 are rare, and so their counts are pooled with those of 29. Ranks are found for 40 000 random matrices and a chi-square test [14] is performed for ranks 32, 31, 30, and ≤ 29.

- ☐ Parking lot test: Randomly place unit circles in a 100×100 square. If the circle overlaps an existing one, choose a new position until the circle does not overlap. After 12 000 attempts, the number of successfully "parked" circles should follow a certain normal distribution.

It is beyond the scope of this lecture to outline all tests. In general, the DIEHARD tests perform operations on random number streams that in the end should be either distributed according to a given distribution that can be computed analytically, or the problem is reduced to a case where a chi-square or Kolmogorov-Smirnov test [14] can be applied to measure the quality of the random series.

NIST test suite The US National Institute of Standards and Technology (NIST) has also published a PRNG test suite [6]. It can be downloaded freely from their website. The test suite contains 15 tests that are extremely well documented. The software is available for many architectures and operating systems and considerably more up-to-date than DIEHARD. Quite a few of the tests are from the DIEHARD test suite, however, some are novel tests that very nicely test the properties of PRNGs.

L'Ecuyer's test suite Pierre L'Ecuyer has not only developed different PRNGs, he has also designed TestU01, a ANSI C software library of utilities for the empirical statistical testing of PRNGs [7]. In addition, the library implements several PRNGs and is very well documented.

5 Nonuniform random numbers

Standard random number generators typically produce either bit streams, uniform random integers between `0` and `INT_MAX`, or floating-point numbers in the interval $[0, 1)$. However, in many applications it is desirable to have random numbers distributed according to a probability distribution that is not uniform. In this section different approaches to generate nonuniform random numbers are presented.

5.1 Binary to decimal

Some generators merely produce streams of binary bits. Using the relation

$$u = \sum_{i=0}^{B} b_i 2^i \tag{15}$$

integers x between 0 and 2^B can be produced. The bit stream b_i is buffered into blocks of B bits and from there an integer is constructed. If floating-point random numbers in the interval $[0, 1)$ are needed, we need to replace $u \to u/2^B$.