Appendix

Pseudo-Random Numbers Generated by Computers

In most cases, the random numbers that are produced by computers are not really random but *pseudo-random*, which means that they are generated in a predictable way by using precise deterministic algorithms. Indeed, even though there are algorithms that are purely random (based upon some atmospheric noise in the computer), in many extents it is much better to have codes that determine reproducible calculations. Indeed, debugging is facilitated by the ability to run the same sequence of random numbers. Moreover, random-number generators that are based upon some thermal noise in the computer are usually much slower than deterministic algorithms, thus giving another reason to avoid them. All routines that produce pseudo-random numbers are just complicated algorithms that condense into few hundreds lines a lot of clever tricks of number theory. The output of these algorithms looks random but it is not. Indeed, when we run them twice (under the same initial conditions), they always give the same output. In practice, these algorithms generate a sequence of numbers $r_0, r_1, r_2, \dots$ in which, at each iteration, r_k is given by some complicated function of the preceding ones. The simplest example is given by linear congruential generators, which are of the form:

$$r_k = f(r_{k-1}). \tag{A.1}$$

A simple choice (that should not be used anymore) is given by:

$$i_k = (a i_{k-1} + b) \quad \text{mod M}, \tag{A.2}$$

where a and b are suitable integers (the quality of the generator crucially depends upon them) and $M = 2^{31} - 1$ (for 32-bit generators). A real number in $[0, 1)$ is then obtained by taking $r_k = i_k/M$. Starting from a given *seed* (here, the initial value for i_0), the full chain of pseudo-random numbers is iteratively generated. Here, the main problem is that the sequence repeats identically after (at most) $2^{31} - 1$ iterations. In this sense, the lagged Fibonacci generators give much better results:

https://doi.org/10.1017/9781316417041.014 Published online by Cambridge University Press

$$i_k = i_{k-p} \bigotimes i_{k-q} \quad \mod \mathrm{M}, \tag{A.3}$$

where $\bigotimes$ indicates a generic binary operation, and p and q are fixed integers. Depending on the chosen binary operation and the integers p and q, we can achieve generators with incredibly long periodicities, e.g., as large as 10^{400} or even longer.

When dealing with pseudo-random numbers, we must keep in mind that subsequent numbers are always correlated, since they come from a deterministic procedure. However, the important point is that they must look nearly random when the algorithm is not known, at least on a "coarse grained" level. More precisely, the conditional probability $\omega(i_k|i_{k-1})$ should be as flat as possible, indicating that the probability to have a given i_k should be independent on the fact of having obtained i_{k-1} at the previous step. This is clearly false for an algorithm like Eq. (A.2), where the outcome at the step $k-1$ completely determines the outcome at the next step. However, once we consider a coarse graining of the interval $[0,1)$, a constant $\omega(i_k|i_{k-1})$ may emerge. Moreover, for the lagged Fibonacci generators,

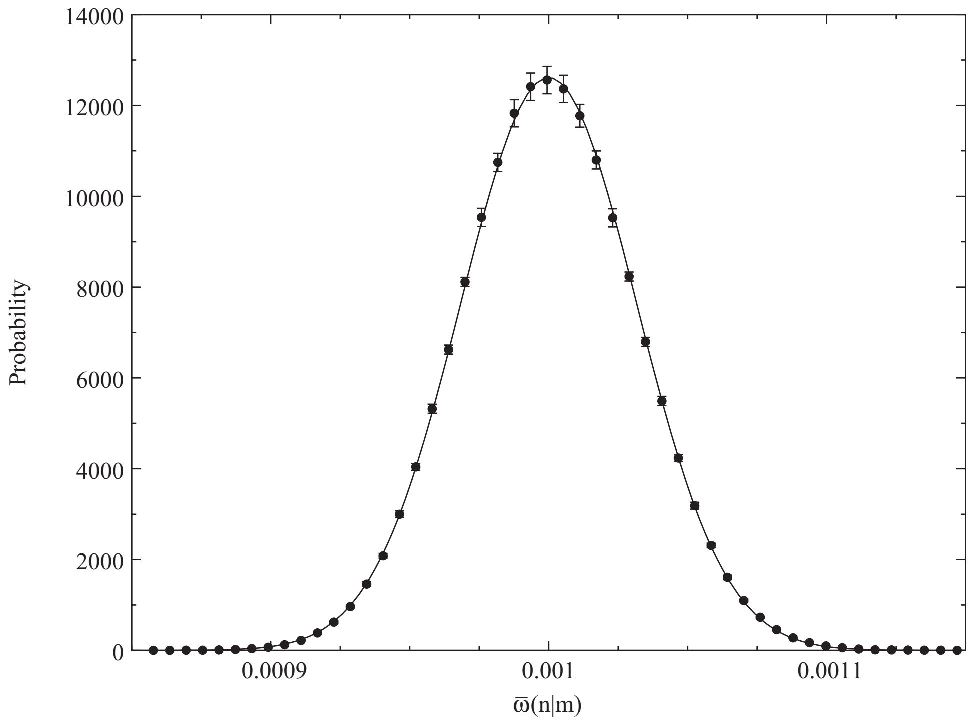

Figure A.1 The interval $[0,1)$ is divided into sub-intervals of width $1/L$, with $L = 1000$. The conditional probability $\overline{\omega}(n|m)$ that two subsequent numbers fall in the sub-intervals $[(n-1)/L, n/L]$ and $[(m-1)/L, m/L]$, respectively, is evaluated and averaged over $M = 10^9$ samples. The distribution function of $\overline{\omega}(n|m)$ is shown.

whenever the period is much longer than $2^{31} - 1 \approx 2 \times 10^9$, the same i_{k-1} will give rise to several different values for i_k along the sequence, leading to a practically uniform probability. In order to exemplify this concept, we divide the interval $[0, 1)$ into L small sub-intervals of width $1/L$, generate M pseudo-random numbers, and calculate the number of times $N(n|m)$ that a number falls in $[(n-1)/L, n/L]$ when the previous one was in $[(m-1)/L, m/L]$. Then, the estimated conditional probability is obtained as $\overline{\omega}(n|m) = N(n|m)/\sum_n N(n|m)$, which is also a pseudo-random number. For a good generator, the probability distribution of $\overline{\omega}(n|m)$ is a Gaussian with mean value equal to $1/L$ and variance $1/M$, as expected from the central limit theorem (the mean of $\overline{\omega}(n|m)$ is $1/L$ and its variance is $1/L + O(1/L^2)$; the number of samplings for each m is $O(M/L)$, on average). In Fig. A.1, we report the case with $L = 1000$ and $M = 10^9$, where the distribution function of $\overline{\omega}(n|m)$ is perfectly fitted by a Gaussian.

In general, random-number generators must satisfy two main requirements (Knuth, 1997):

1. To have a long period. Indeed, every deterministic generator must eventually loop and the goal is to make the period as long as possible.
2. To have a good statistical quality. In this respect, the output of the generator should be practically indistinguishable from the one of a true random-number generator and it should not exhibit any correlation along the sequence of numbers.

A poor quality of the generator can ruin the results of Monte Carlo applications (Ferrenberg et al., 1992); for this reason, it is very important that generators are able to pass the set of empirical tests, as for example the battery of *Diehard Tests*, developed by George Marsaglia, or the *TestU01 Library*, introduced by Pierre L'Ecuyer and Richard Simard.