from the initial boundary of the reactants, then $x(t)$ and $w(t)$ can be written as

$$\langle x\rangle(t) = \frac{\sum_x x\,n(x,t)}{\sum_x n(x,t)} \tag{7.57}$$

$$w(t)^2 = \frac{\sum_x [x - \langle x\rangle(t)]^2\, n(x,t)}{\sum_x n(x,t)}. \tag{7.58}$$

Choose lattice sizes of order 100×100 and average over at least 10 trials. The fluctuations in $x(t)$ and $w(t)$ can be reduced by averaging $n(x,t)$ over the order of 100 time units centered about t. More details can be found in Jiang and Ebner. ■

7.9 ■ RANDOM NUMBER SEQUENCES

So far we have used the random number generator supplied with Java to generate the desired random numbers. In principle, we could have generated these numbers from a random physical process, such as the decay of radioactive nuclei or the thermal noise from a semiconductor device. In practice, random number sequences are generated from a physical process only for specific purposes such as a lottery. Although we could store the outcome of a random physical process so that the random number sequence would be both truly random and reproducible, such a method would usually be inconvenient and inefficient in part because we often require very long sequences. In practice, we use a digital computer, a deterministic machine, to generate sequences of *pseudorandom* numbers. Although these sequences cannot be truly random, such a distinction is unimportant if the sequence satisfies all our criteria for randomness. It is common to refer to random number generators even though we really mean pseudorandom number generators.

Most random number generators yield a sequence in which each number is used to find the succeeding one according to a well-defined algorithm. The most important features of a desirable random number generator are that its sequence satisfies the known statistical tests for randomness, which we will explore in the following problems. We also want the generator to be efficient and machine independent and the sequence to be reproducible.

The most widely used random number generator is based on the *linear congruential* method. One advantage of the linear congruential method is that it is very fast. For a given seed x_0, each number in the sequence is determined by the one-dimensional map

$$x_n = (a x_{n-1} + c) \bmod m, \tag{7.59}$$

where a, c, and m as well as x_n are integers. The notation $y = z \bmod m$ means that m is subtracted from z until $0 \le y < m$. The map (7.59) is characterized by three parameters, the *multiplier* a, the *increment* c, and the *modulus* m. Because m is the largest integer generated by (7.59), the maximum possible *period* is m.

In general, the period depends on all three parameters. For example, if $a = 3$, $c = 4$, $m = 32$, and $x_0 = 1$, the sequence generated by (7.59) is 1, 7, 25, 15, 17, 23, 9, 31, 1, 7, 25, ..., and the period is 8, rather than the maximum possible value of $m = 32$. If we choose a, c, and m carefully such that the maximum period is obtained, then all possible integers between 0 and $m - 1$ would occur in the sequence. Because we usually wish to have random numbers r in the unit interval $0 \le r < 1$, rather than random integers, random number generators usually return the ratio x_n/m which is always less than unity. Several rules have been developed (see Knuth) to obtain the longest period. Some of the properties of the linear congruential method are explored in Problem 7.35.

Another popular random number generator is the *generalized feedback shift register* method which uses bit manipulation (see Sections 14.1 and 14.6). Every integer is represented as a series of 1s and 0s called bits. These bits can be shuffled by using the bitwise `exclusive or` operator $\oplus$ (xor) defined by $a \oplus b = 1$ if the bits $a \neq b$; $a \oplus b = 0$ if $a = b$. The nth member of the sequence is given by

$$x_n = x_{n-p} \oplus x_{n-q}, \tag{7.60}$$

where $p > q$, and p, q, and x_n are integers. The first p random integers must be supplied by another random number generator. As an example of how the operator $\oplus$ works, suppose that $n = 6$, $p = 5$, $q = 3$, $x_3 = 11$, and $x_1 = 6$. Then $x_6 = x_1 \oplus x_3 = 0110 \oplus 1011 = 1101 = 2^3 + 2^2 + 2^0 = 8 + 4 + 1 = 13$. Not all values of p and q lead to good results. Some common pairs are $(p, q) = (31, 3)$, $(250, 103)$, and $(521, 168)$.

In Java and C the exclusive or operation on the integers `m` and `n` is written as `m^n`. The algorithm for producing the random numbers after p integers have been produced is shown in the following. Initially the index k can be set to 0.

1. If $k < q$, set $j = k + q$, else set $j = k - p + q$.
2. Set $x_k = x_k \oplus x_j$; x_k is the desired random number for this iteration. If a random number between 0 and 1 is desired, divide x_k by the maximum possible integer that the computer can hold.
3. Increment k to $(k + 1) \bmod p$.

Because the exclusive or operator and bit manipulation is very fast, this random number generator is very fast. However, the period may not be long enough for some applications, and the correlations between numbers might not be as good as needed. The shuffling algorithm discussed in Problem 7.36 should be used to improve this generator.

These two examples of random number generators illustrate their general nature. That is, numbers in the sequence are used to find the succeeding ones according to a well-defined algorithm. The sequence is determined by the seed, the first number of the sequence, or the first p members of the sequence for the generalized feedback shift register and related generators. Usually, the maximum possible period is related to the size of the computer word, for example, 32 or 64 bits. The choice of the constants and the proper initialization of the sequence is very important, and thus these algorithms must be implemented with care.

There is no necessary and sufficient test for the randomness of a finite sequence of numbers; the most that can be said about any finite sequence of numbers is that it is *apparently* random. Because no single statistical test is a reliable indicator, we need to consider several tests. Some of the best known tests are discussed in Problem 7.35. Many of these tests can be stated in terms of random walks.

Problem 7.35 Statistical tests of randomness

(a) *Period.* An obvious requirement for a random number generator is that its period be much greater than the number of random numbers needed in a specific calculation.