Homework 1; due Tuesday, September 24

PY 502, Computational Physics, Fall 2024

Department of Physics, Boston University

Instructor: Anders Sandvik

TEST OF A RANDOM NUMBER GENERATOR USING RANDOM WALKS

In this assignment, you will test the quality of a random number generator by using it to simulate a random walk, comparing the exact probability distribution with ones obtained in simulations.

Random walk

Consider a stochastic process in which at each time step $t = 1, \ldots, n$ we can move one step "up" or "down" ($x \to x \pm 1$, with probability $1/2$ for $+$ and $-$), starting at $t = 0$ at $x = 0$. Three realizations of such a random walk with $n = 100$ are shown in Fig. 1.

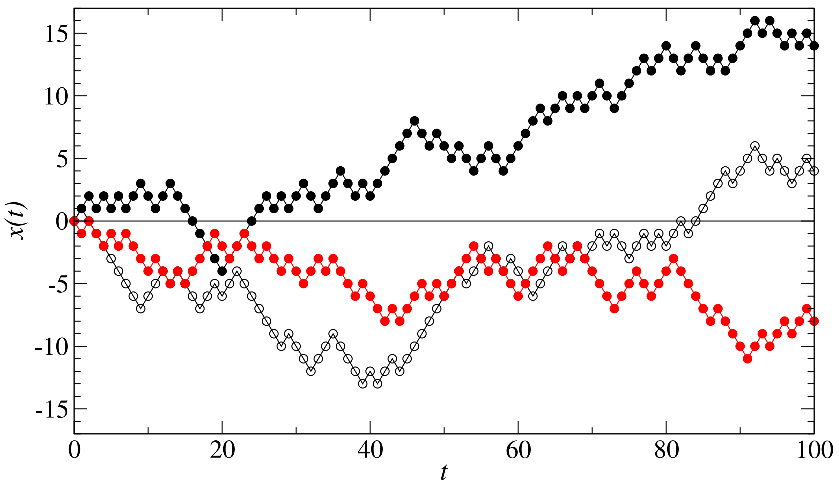

Figure 1: Three different realizations of a random walk with $n = 100$ steps.

It is easy to calculate the probability distribution of x after n steps: The total number of walks having n_+ moves up and n_- down ($n = n_+ + n_-$) is

$$N(n_+, n_-) = \binom{n}{n_+} = \frac{n!}{n_+!n_-!}. \tag{1}$$

The probability for each combination of up and down moves (one random walk) is the same; $1/2^n$. Since $n = n_+ + n_-$ and x after n steps is $n_+ - n_-$, we can write the distribution after n steps as

$$P_n(x) = \frac{1}{2^n} \frac{n!}{[(n+x)/2]![(n-x)/2]!}, \tag{2}$$

which of course is a binomial distribution. Here it should be noted that if n is even, x must also be even (for a non-zero probability), whereas if n is odd x is also odd.

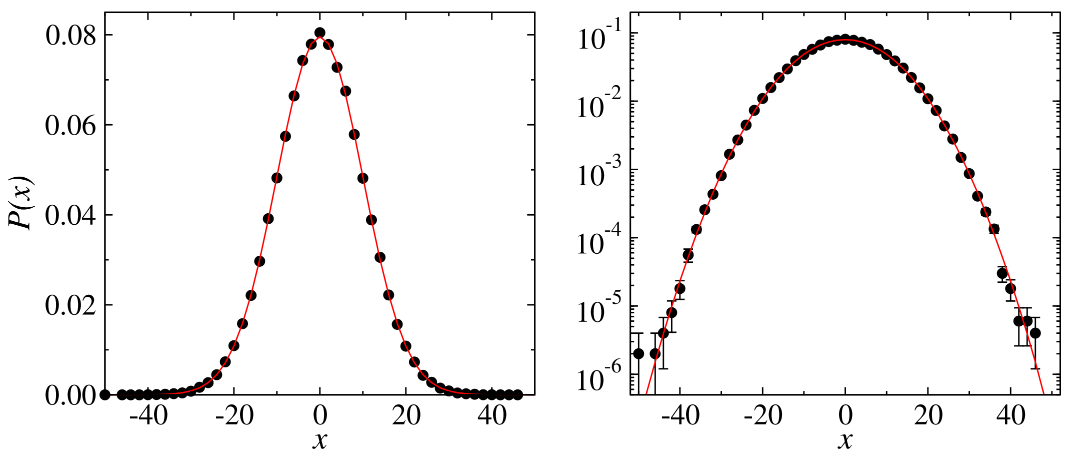

Figure 2: Probability distribution of the displacement after a random walk of $n = 100$ steps, based on 5×10^5 different walks (circles with error bars), compared with the exact distribution (solid curve) plotted on linear (left) and log (right) scales.

To explicitly relate the probability distribution to the general binomial distribution (which will be referred to later), if the probabilities of + and − are p and $1 - p$, respectively, the distribution of the number of + events z is

$$P_n(z) = p^z (1-p)^z \frac{n!}{z!(n-z)!}. \tag{3}$$

From this distribution we obtain Eq. (2) by setting $p = 1/2$ and $z = (x + n)/2$ (assuming n is even now). We can think of Eq. (3) as the distribution of a random walk where the steps are +1 or 0 instead of +1 and −1, and the relationship between the two types of random walks is trivial.

Simulated random walk

In a simulation, we use a random number generator to make the decisions of + or - movement at each time step (e.g., we generate a floating point number r between 0 and 1, and move up or down if $r < 1/2$ and $r \geq 1/2$, respectively). If the random numbers are not perfect, there will be deviations in the probability distribution from the exact distribution (2).

To compute the simulated distribution, we have to repeat simulations many times, using different sequences of random numbers. Fig. 2 shows the distribution for $n = 100$ steps, obtained using 5×10^5 independent simulated random walks with a good random number generator (as the examples shown in Fig. 1). Here "error bars" for the simulation results have been computed by grouping the results of the simulations into 50 "bins" and computing the standard deviation of the average result based on these bin results (we will discuss data binning more later; for now it is not important how the errors were obtained exactly). The error bars, where visible, show semi-quantitatively that the deviations are consistent with the expected statistical fluctuations; the exact distribution falls mostly within the error bars (statistically, one would expect about 2/3 of the points to be within one standard deviation of the true result). For a bad random number generator, the exact distribution should fall significantly outside the error bars.

Expected statistical fluctuations

We can only perform a finite number of these simulated walks, and therefore we will not obtain the exact distribution (2). We then have to analyze the statistics of the expected *deviations* from the exact distribution, to determine whether the observed deviations from (2) are just due to natural statistical fluctuations based on a finite sample, or whether they are larger than would be expected (in which case they must be due to flaws in the random number generator used). Instead of just computing error bars and examining results visually as in Fig. 2, let us analyze the statistical fluctuations theoretically.

Let us estimate what deviations from the exact distribution (2) should be expected based solely on a finite number N_w of walks used to estimate the distribution. Consider the distribution of the number of walks $C_n(x)$ leading to the location x at the final step. In addition to the expectation value of the number of counts, $\langle C_n(x)\rangle = N_w P_n(x)$, we need the variance. The distribution of $C_n(x)$ is simply another binomial distribution, like Eq. (3) with the number of steps n replaced by N_w, the probability $p = P_n(x)$, and $z = C_n(x)$—the number of walks out of N_w that end up at x.

The variance of the binomial distribution based on N_w repetitions is $\mathrm{var}(z) = N_w p(1-p) = \langle z\rangle(1-p)$. Thus, in our case the variance is

$$\mathrm{var}(C_n(x)) = N_w P_n(x)[1 - P_n(x)]. \tag{4}$$

The factor $1 - P_n(x)$ can be approximated as 1 if n is reasonably large. For simplicity we will now make this approximation and set $\mathrm{var}[C_n(x)] \approx N_w P_n(x)$, but please remember the small expected correction in case your results later on show some small deviations from the derived forms (and it would be easy to correct those forms as well by using the exact form of the variance).

Give the variance, we have the standard deviation $\sqrt{\mathrm{var}[C_n(x)]}$, and we can write the expected number of "hits" at x with its statistical error as

$$C_n(x) = N_w P_n(x) \pm \sqrt{N_w P_n(x)}. \tag{5}$$

Let us now divide this by N_w, to get a properly normalized probability, and compute the expected deviation from the actual distribution;

$$\Delta_n(x) = C_n(x)/N_w - P_n(x). \tag{6}$$

The expected value of this quantity (for a perfect random number generator) is 0, and from (5) we see that the expected (standard) deviation is;

$$\sigma_n(x) = \sqrt{\frac{P_n(x)}{N_w}}. \tag{7}$$

We can now compute the total *squared* deviation (in order to have a sum over positive numbers characterizing the average deviation) over all final outcomes x, and call it Δ^2.

$$\Delta^2 = \sum_{x=-n}^{n} \Delta_n^2(x). \tag{8}$$

According to Eq. (7) its expected value should be

$$\langle \Delta^2 \rangle = \sum_x \sigma_n^2(x) = \frac{1}{N_w} \sum_x P_n(x) = \frac{1}{N_w}. \tag{9}$$

It is appropriate to take the square root, to get an RMS (root-mean-square) deviation;

$$\Delta = \sqrt{\sum_x \Delta_n^2(x)} = \frac{1}{\sqrt{N_w}}. \tag{10}$$

We can test the random number generator by computing Δ according to (8) with (6), where $C_n(x)$ is the actual count based on N_w simulations. According to Eq. (10), for increasing number N_w of simulated random walks, this number should decay as $1/\sqrt{N_w}$ if the random number generator is good. If, on the other hand, the generator is bad, we expect Δ to approach some non-zero value as $N_w \to \infty$, reflecting a distribution for the "random" walk different from (2). We can also look at $\sqrt{N_w}\Delta$, which should approach 1 for a good generator and diverge as $\propto \sqrt{N_w}$ for a bad generator.

Programming task

The random number generator you should test here is the 64-bit linear congruential generator discussed in class, which uses the algorithm

$$r_{k+1} = a \cdot r_k + c, \tag{11}$$

with $a = 2862933555777941757$ and $c = 1013904243$. You should work with these by declaring variables of type `UInt64` in Julia.

A random walk consists of n steps, but you do not have to store all the values r_k, $k = 1, \ldots, n$, just use a single integer, called r below, which is updated, $r = ar + c$ at each iteration.

Normally, when using this kind of random number generator, the integers r would be converted into double precision floating-point numbers between 0 and 1. Here we will instead let the individual bit values $\{0, 1\}$ of r_n determine 64 different random walks with displacements $x[i]$, $i = 1, \ldots 64$. To extract the bits of an integer r to bit values and store them in an array b of 64 integer elements $b[i]$, $i = 1, \ldots, 64$, we can use the Julia bitwise functions `>>> k` (right shift by k steps) and `&` (and) in this way:

$$b[i] = (r >>> (i-1))\ \&\ 1, \tag{12}$$

where it should be noted that the bits in the computer are numbered $0, \ldots, 63$, while the array should have indices $i = 1, \ldots, 64$ (the convention in Julia). The step to take in the random walk as defined here can then be obtained as $b[i] = 2b[i] - 1$, so that the walk governed by bit i is updated as $x[i] = x[i] + b[i]$ after each step n of the random number generation in Eq. (11 . We want to check for differences in the randomness among the bits, through the quality of random walks based on them.

Write a program that tests all the bits in the same run, i.e., perform 64 different random walks simultaneously, using the 64 different bits. The program should read in the number of random walks to be performed, N_w, and the number of steps to be taken in each walk, n, from a file `read.in` (integers on two different lines). Read the initial integer r_0 to be used as a seed for the random number generator (11) from this same file (an integer on the third line). Note that the seed can be any integer, but if you repeat the calculation multiple times to collect more data, you should make sure to use a different seen every time (otherwise each run will produce identical random walks, and the computed statistical properties will not reflect independent random walks). Note, however, that you should not "reseed" the generator after each random walk, just continue iterating Eq. (11) even when you start a new walk after resetting the locations to $x[i] = 0$.

After each set of walks, accumulate the counts $C_n(x)$ of x values for for each bit in an array, i.e., it should be a two-dimensional array, with indices corresponding to the possible x values for each bit (again because of Julia conventions the indices have to be shifted from the actual values $[-n, n]$ to $[1, 2n + 1]$). After all the N_w walks have been performed, compute the deviations Δ based on Eqs. (8) and (6). Write $\sqrt{N_w}\Delta$ for all bits to a file `d.dat` (containing 64 lines, with b and $\sqrt{N_w}\Delta(b)$ on each line). Also, the program should write the full distributions for each bit to files with names `pnn.dat`, where `nn` stands for the bit numbers; `nn=00,01,...,63`. Write a separate line for each x and the corresponding simulated probability (number of counts divided by N_w). Write the exact probability $P_n(x)$ in the third column.

Plan the program carefully before you start coding. Break down the simulations into functions for well-defined tasks to make the program easy to read and debug. Also pay attention to code efficiency, avoid use of global variable in time consuming tasks, etc.

Hint: To compute $P_n(x)$, write down an expression for its logarithm first and evaluate it, in order to avoid numerical problems with very large and very small numbers.

Simulation tasks and report

Do calculations for $n = 100$ and vary the number of walks. Use $N_w = 10^m$, with $m = 2, 3, \ldots,$ up to m as high as you can reasonably do on the SCC cluster—$m = 6$ or 7 will be sufficient and can be easily done with interactive runs using "SCC on demand", but it will be useful to go further with batch jobs (using up to several hours of CPU time).

Write a report showing the ability (or lack thereof) of the individual bits to reproduce the correct statistics of the random walk. Show some illustrative graphs of the simulated probability distributions compared with the correct $P_n(x)$ (it is useful to plot both on linear and log scales, as in Fig. 2). Show the behavior of $\sqrt{N_w}\Delta$ versus the bit number $b = 0, \ldots, 63$ for the different N_w you used. Explain qualitatively why you think the results look as they do.

Assuming that there are no correlations between the bits b (which may not be true and could also be tested), how do you judge this generator's ability to produce random floating-point numbers when the integers between 0 and $2^{63} - 1$ are linearly converted to the range $[0, 1)$?

Your report should explain clearly what you have done. Each figure should have a caption explaining the figure, and the figures should be discussed and referenced in the text. The figures should have clear axis markings and the plotting symbols should have shapes/colors that are clearly visible. Consider your report as an opportunity to practice scientific writing!