Test of a 64-bit random number generator

Na Xu

In this problem, we generated random numbers by using a 64-bit random number generator. Also, we used these random numbers to simulate the process of random walks. Through comparing the results from the simulation with the theoretical ones, we can then analyze the quality of this random number generator.

1.Introduction and Theory

A Random Walk is a mathematical formalisation of a path that consists of a succession of random steps. At each step, we can move one step up ($x- > x+1$)or down ($x- > x-1$). It is easy to calculate the probability distribution of x after n steps: the total number of walks having n_+ moves up and n_- moves down is

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

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

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

Here it should be noted that if n is even, x must also be even, if n is odd x is also odd. otherwise, P_n(x)=0. In a simulation, we use a random number generator to make the decisions of up or down 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$\geqslant$ 1/2, respectively) If the random numbers are not perfect, there will be deviations in the probability distribution from the exact distribution(2).

Because we can only perform a finite umber of these simulated walks, and therefore we will not obtain the exact distribution (2). Here are the analysis for the deviations: if we assume the statistics of the final point of the random walk (after n steps) as "counting statistics", then if we end up with N_x walks at final position x, the expected standard deviation would be $\sqrt{\mathrm{N}_x}$. Thus, we can write the "counts" $C_n(x)$ as:

$$C_n(x) = N_w P_n(x) \pm \sqrt{N_w P_n(x)}. \tag{3}$$

The expected deviation from the actual distribution is

$$\Delta_n x = C_n(x)/N_w - P_n(x). \tag{4}$$

Then the total sqared deviation is

$$\Delta^2 = \sum_{x=-n}^{n} \Delta_n^2 \tag{5}$$

its expected value should be

$$\langle \Delta^2 \rangle = \frac{1}{N_w} \tag{6}$$

then the root-mean-square deviation is

$$\Delta = \sqrt{\sum_x \Delta_(n)^2(x)} = \frac{1}{\sqrt{N_w}} \tag{7}$$

from (8), we can see that if $\sqrt{N_w}\Delta$ approach 1 as $N_w- > \infty$, then the generator used is a good one, otherwise, if $\sqrt{N_w}\Delta$ diverges with increasing N_w, the generator is bad.

2.Programming

Here we are going to test the 64-bit random number generation, which works according to the algorithm

$$r_{n+1} = a * r_n + c, \tag{8}$$

with a=2862933555777941757 and c=1013904243. For each random number we got, we can do 64 different random walks simutaneously by testing the value of every bit (b from 0-63) is 0 or 1 for each step. After walking for n steps, we record the position x for each bit in an array a(x,b), then after doing Nw different random walks, we can get a distribution of positon x for each b and wrote them into different files. Thus, by comparing the real x distribution with the "real" one P(x), we expect to test the quality of the random number generator.

3.Results and Analysis

Fig.1 shows the behavior of sqrt$N_w\Delta$ for each b (bit), also we plotted different curves with different $N_(w)$. From Fig.1 we can see that when b is small (from 0-5 for N_w=100 and 1000, from 0 to 10 for N_w=10E4 and 10E5, from 0 to 15 for N_w=10E6) the fluctuation is quite noticeable, however, after that, for each b, the value of sqrt$N_w\Delta$ is confined near 1 (which indicates good random numbers). In summary, we can see from Fig.1 that for small b, it can not generate good random numbers,therefore,it can not reproduce the correct statistics of the random walk. However,for

higher order of b, the quality of the random numbers is very good and it can simulate the process of random walk very well.

To further see the how different N_w affect the distribution at smaller b and larger b, we can plot the real position distribution for every bit and compare it with the ideal one. Here I choose b=8 and b=30, which stand for the situations when b is small and when b is big.

From Fig.2, we can see that for small b, the simulated distribution is very different from the ideal one, moreover, as we increase Nw, Δ is not supressed at all. That shows that at small b, it is not a good random number generator. However, for larger b, the simulated distribution is very much the same as the ideal one, also as we increase Nw, Δ is apparently supressed, which shows a clear sign of a good random number generator.

In summary, from what we get from the simulation results, we can conclude that the 64-bit random number generator can generate good random numbers at high-ordered bit. Since it can produce good random number from b=15,then if we use this generator to get float point within the range [0-1),it can give us good and evenly distributed random numbers if the precision is higher than $(0.2)^{48}$, which is 10^{-34}. Thus, for normal use,when we don't need the random number with that high precision, this 64-bit random number generator is a good one for generating random numbers.

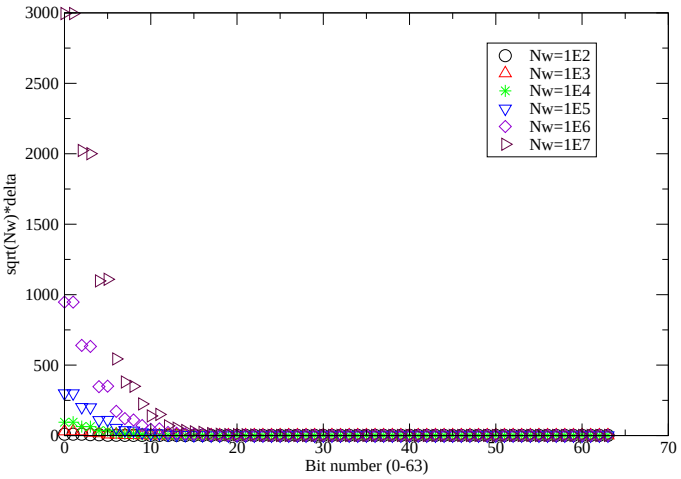

FIG. 1: sqrt$N_w\Delta$ for N_w from $10^2 - 10^7$

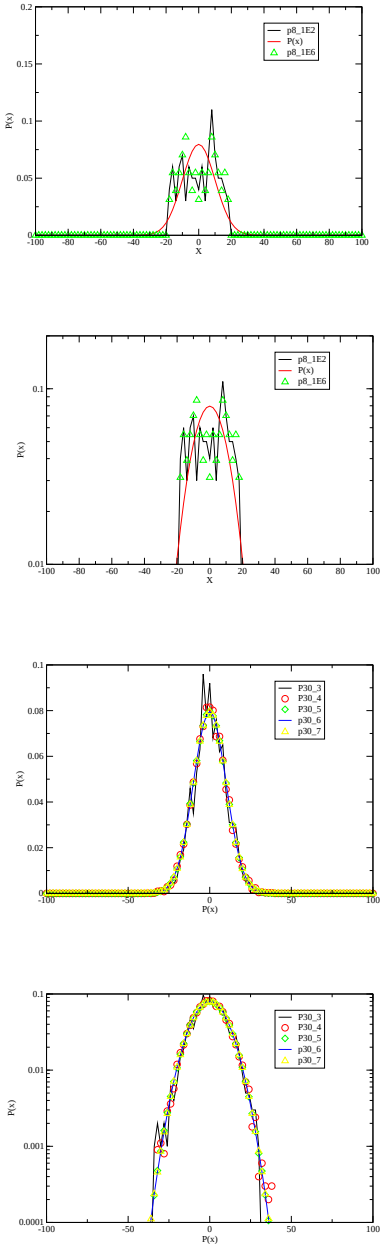

FIG. 2: Real distribution Vs ideal distribution. (a) b=8, for Nw=1E2 and 1E6; (b) same as (a), but use log scale; (c) b=30, for Nw=1E3 to 1E7 (d) same as (c),but use log scale.