

Numerical Simulations

S. C. Phatak*

Institute of Physics, Bhubaneswar 751 005

March 1, 2007

Abstract

This document is based on a series of lectures given at Institute of Physics. Here we first discuss random number generators which are extensively used in numerical simulations. We then consider some specific problems for numerical simulations on computers.

1 Introduction

Numerical simulations of complex systems are now done in various branches of scientific and engineering subjects. When one is building complex systems one needs to first build prototypes. But this could be an expensive proposition and one would like to have some idea of fixing the parameters of the prototype by some means before a prototype is built and tested. One would also like to investigate the properties of complex systems theoretically to compare with the experimental results in order to have a better understanding of the system. One essentially constructs models of the complex system and makes theoretical investigations based on the physical laws that the system is supposed to follow. This itself is a simulation in a very rudimentary sense. If the model is simple, it may be possible to study the system analytically or with very little computations. However, sometimes more detailed models need to be considered and these may not be amenable to simple analytical investigations. With the continuous increase in the computational power, it is becoming possible to consider complex models and investigate their properties numerically. Some of the examples of numerical simulations in engineering are designing of planes, cars etc for better efficiency, structural designs and so on. In chemistry and biology simulations are used to design complex molecules.

In physics as well, the simulations are playing a bigger role. In fact, some people argue that now we have three branches of physics: theoretical physics, experimental physics and

*email: phatak@iopb.res.in

computational physics and numerical simulations form an important component of computational physics. Although computational physics has become very important, particularly because of phenomenal increase in the computational power, one should not get carried away and demean theoretical and experimental branches. Because, in the final analysis, results from experiments should always clinch the issue. At the most, computations will give guidance about what results are expected from experiments. Also, any amount of computations will not replace theoretical work. One would not be able to develop a new theory from computations. In fact, the computations and simulations only tell us about what results are expected from a given theory. Finally these results are to be confirmed by the experiments. If experiments fail to confirm the results of computations or simulations, it indicates the failure of the theory or inadequacy of the model used in computations.

In computer simulations one constructs a model of physical reality and applies the laws of physics to compute the results. Thus simulations are like the thought experiments which were in vogue during the early days of quantum mechanics. The only difference is the model one constructs is quite complex so one cannot work out the results in mind or on a piece of paper. One has to use a computer for that. And because of the power of computers these days, one can make the model complex to come as close to the actual physical situation as possible. But it must be emphasised that the computer model is still a model and not a physical system. In this sense computational physics is distinct from theoretical physics and experimental physics. It has its own role but it cannot replace either of them. In this series of lectures we are not going to discuss computational physics but concentrate on numerical simulations. Numerical simulations. As the name suggests, numerical simulations attempt to simulate the physical system under study by some computational techniques. Consider, as an example the electric potential $V(\vec{r})$ generated by a complex system of charges. This is a well defined problem which requires the solution of differential equation

$$\nabla^2 V(\vec{r}) = -4\pi\rho(\vec{r}) \tag{1}$$

where $\rho(\vec{r})$ is the given charge distribution. In addition there might be some boundary conditions specified on $V(\vec{r})$. We have spent a lot of time solving this equation for some simple charge distributions. For arbitrary charge distributions, the problem can, in principle, be solved but one has to resort to numerical methods. A standard numerical method of solving the problem is to discretize the space and convert the difference equation to a difference equation. There are a number of numerical methods to solve these difference equations, one of them being matrix inversion. So, we have converted a physics problem into a model problem. The solution of this model problem is obtained by some numerical techniques. If we can make the coordinate grid fine enough, the solution of the model problem is expected to be very close to the solution obtained by solving the differential equation.

If one does an experiment of measuring the electric potential, we expect the results to be close to the results obtained from the calculation.

Let me assure you that the problem discussed above is not just of academic interest. It was needed to be solved in the design and construction of the photon multiplicity detector. One of our students, Anand Dubey, had done numerical simulations and studied the performance of the detector of various shapes. This study was important and essential for the designing of the configuration (shape) of the detector. He did not actually measure the electric potential inside the detector but the knowledge of the potential was required in computing the performance of the detector and the performance was experimentally measured.

Another, much more publicised, example of numerical simulation is the lattice QCD. We know that the quantum chromodynamics (QCD) is the theory strong interactions. One problem with QCD is that the effective coupling constant becomes large as one decreases momentum transfer and standard perturbative calculations cannot be done. Unfortunately the important tests of the theory comes from the low energy behavior of the hadrons at small momentum transfers and one would like to compute those from QCD. For example, one would like to compute the masses of hadrons, their interactions etc from QCD. Wilson showed that if one considers a discrete space and time, and studies the QCD on this space-time lattice, the problem becomes tractable. So, here again we make a model of a real physical situation and apply the laws of physics (in this case QCD) to this model and solve the problem. Wilson had shown that this can be done and in the limit of vanishing lattice spacing, the results of the lattice calculations can be continued to the real theory. A number of people are involved in doing the lattice QCD calculations and obtaining important results of strongly interacting systems. One of the results is that the matter at very high temperatures makes a transition from a hadronic matter to quark gluon plasma.

Simulations are extensively used when one studies complex physical systems consisting of a number of components. An example is the collision of two heavy nuclei. Each nucleus consists of a number of nucleons and when the two nuclei collide, the individual nucleons in the two nuclei go through collision process. This is a very complex process and numerical simulations are used to investigate what happens in such collisions. Other examples of numerical simulations are the study on non-equilibrium systems, mesoscopic systems, polymers and so on. I mentioned an example of detector design above. The simulations are extensively used in the design of experiments, design of instruments etc.

I must mention here that there is no all-inclusive methodology or procedure for simulation applications. For each system one wants to study, one has to develop the model and apply suitable numerical technique for solving the problem. So one doesn't have a cure-all antibiotic. So, we shall consider a few examples and hopefully, by looking at the method-

ology one may be able to investigate new systems. But there is one common tool used in a large number of simulations. That is the Monte Carlo method. This method requires what are called as the random numbers. What we shall do first is to understand what are random numbers and how they are generated on computers. We shall then consider some specific case studies.

2 Random Numbers

Before we discuss how the random numbers are generated on computers, let us first understand what are random numbers. We have some idea of what these are from physical examples. The simplest one is the (binary) numbers obtained by tossing a coin. We know that the result of the tossing of the coin cannot be predicted before hand. In an ideal case, getting a head or a tail (0 or 1 in binary language) is equally probable. We can find a number of such physical processes which can, in principle, be used for generating random numbers. To name a few, we can consider throwing a dice, using roulette wheel, counting decays of a radioactive material in some unit time, noise level in electronic circuit etc. One problem with these is that these are not fast enough. In a realistic simulation calculation we may need billions of random number and one can convince oneself that the time required to produce that many numbers from physical processes would be much larger than the computation time. Storing that many numbers for repeated future use is also prohibitive. So we need an algorithm which is fast enough and can be used on computer. That is the subject matter of this section.

The concept of generating random numbers on a computer appears to be ridiculous. After all, the computer is a predictable, deterministic machine and the idea behind a random numbers is that these are not predictable. So we must be clear on what we mean by random numbers. Based on these ideas we can devise tests for random numbers and we should insist that the numbers generated by the deterministic random number generation algorithm pass these tests. Many people distinguish the numbers generated from such algorithms from the genuine random numbers generated from a physically random process by calling the former as pseudorandom numbers. We shall not make such a distinction. For us, any algorithm which gives numbers which pass the tests gives us random numbers.

First thing which must be made clear is that it is not meaningful to talk of a single random number. Take an example of a dice. When one throws a dice, one gets any number between one and six. So a single number between 1 and 6 is as random as any other number. So, one can only talk of a sequence of numbers which may or may not form a random sequence. So, when we talk of random numbers we are always considering a sequence which is, in principle, an infinite sequence. Below we shall now specify the properties that the random numbers should satisfy. We shall then insist that the sequences generated from a

random number algorithms should satisfy these properties.

2.1 Distribution Test

As a specific example, we shall consider sequences having random numbers distributed uniformly between zero and one¹. Random numbers with any other distribution can always be obtained from these uniformly distributed sequences. This means that the probability of a number lying between x and $x + \Delta x$ is Δx . So, if we have a sequence of N random numbers, we shall have $n_{\Delta x} = N\Delta x$ of these lying between x and $x + \Delta x$, on the average. This is the first test that the sequence must pass.

Note that $n_{\Delta x}$ is the average number and on each trial one would get different values of n . IN fact, we can show that the probability of finding n numbers in an interval x and $x + \Delta x$ out of a set of N random numbers is

$$n_{\Delta x}(N) = \frac{N!}{n!(N-n)!}(\Delta x)^n(1-\Delta x)^{N-n} \quad (2)$$

In the limit of large N the probability peaks at $N\Delta x$ and the peak has a width of $\sqrt{N\Delta x}$. So, if we repeatedly compute n the numbers in an interval x and $x + \Delta x$ for sets of random numbers N (different sets every time), these should be distributed according to the probability distribution given above in eq(2).

The results of applying the distribution test to the machine supplied random number generator are shown in Fig(2.1). Here we have taken a sequence of 1000 random numbers and computed the frequency of numbers between 0.4 and 0.5. The computation is repeated 1000 times. The figure shows that the generator passes the distribution test.

For the distribution test above we have chosen a small interval $(x, x + \Delta x)$. In principle one should then perform the test by choosing different values of x . Instead, we can divide the range $[0, 1]$ into say, m equal intervals and compute the frequency of numbers in each of these intervals. Obviously the average number of numbers in each interval is $\frac{N}{m}$. We can then compute the χ^2 statistic for the measured frequency². The χ^2 statistic is distributed according to the χ^2 distribution of $m - 1$ degrees of freedom. The results of χ^2 test are displayed in Fig(2.2).

2.2 Correlation tests

The distribution test described above checks whether the sequence is uniformly distributed or not. This does not necessarily mean that the sequence is random. For example consider

¹It should now become clear that we shall be considering statistical properties and the sequences must satisfy these properties statistically

²see Appendix?? for the details of computing χ^2 and what results are expected for the statistic

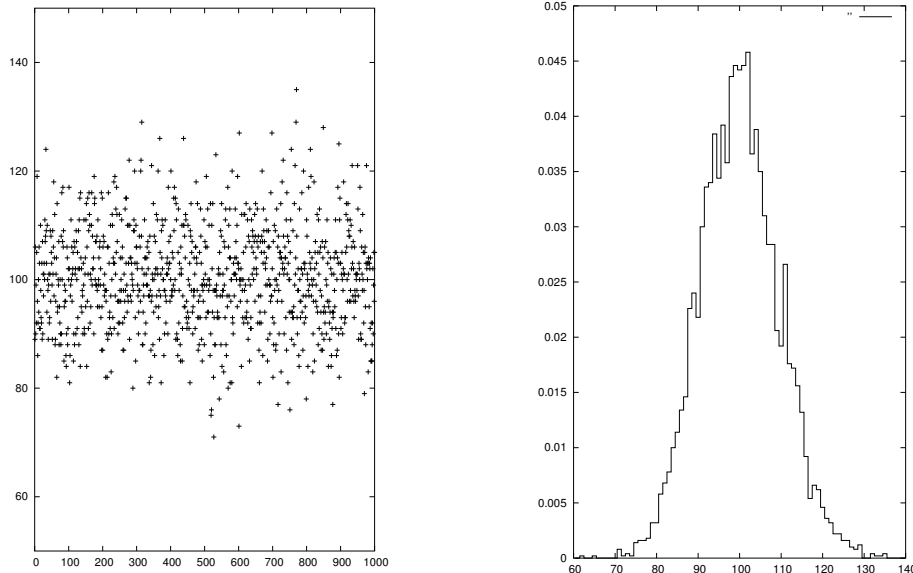


Figure 1: The results of distribution test. The left panel shows the number of random numbers falling between 0.4 and 0.5 for 1000 trials. The right panel shows the frequency distribution. The peak at 100 is well marked and the width is about 10. The computed average and RMS deviations are 100.4 and 9.4 respectively.

two random sequences U_i and V_i having uniform distribution between 0 and 1. Now consider a new sequence W_i which is constructed such that all the odd elements are given by $\frac{1}{2}U_i$ and all even elements are given by $\frac{1}{2}(1 + V_i)$. Clearly, W_i is also uniformly distributed but not random as all its odd elements are always less than 0.5 and all the even ones are greater than 0.5. In other words, there is a well-defined correlation between successive elements. For a random sequence we should not have such a correlation. This is an important requirement for computer generated sequences because, after all, the sequence is generated by a well-defined algorithm and there is likely to be a correlation between successive elements.

In a genuine random sequence, the behavior of successive elements have no correlation. Thus, what ever may be the value of i^{th} element, the $(i + 1)^{th}$ element should be distributed randomly. Thus, if we choose two intervals $(x, x + \Delta x)$ and $(y, y + \Delta y)$, the probability of two successive numbers falling in these two intervals is $\Delta x \Delta y$ and it is independent of x and y .

A test for correlations between two successive numbers can be devised on the lines of the distribution test. A pair of numbers lie in a square of unit area. We divide the square into n^2 equal parts. Considering a sequence of, say, m pairs of random numbers, we compute the number of occurrences in each of the parts. The expected number of occurrences is $\frac{m}{n^2}$. Choosing $n = 10$ and $m = 1000$, we expect 10 occurrences in each part. We then compute the χ^2 statistic for the trial. Since we have $n^2 = 100$ parts, the χ^2 should follow the χ^2

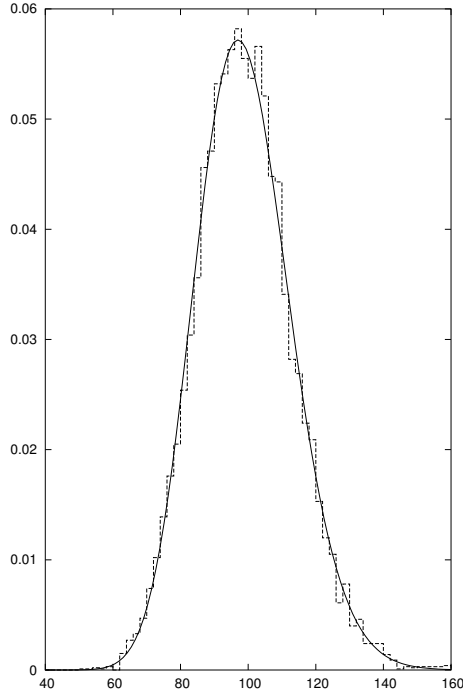


Figure 2: The results of χ^2 test for distribution. The interval $[0,1]$ is divided into 100 equal parts and a set of 1000 random numbers are used. The expected numbers in each interval is 10. The value of χ^2 is computed for each set. Then χ^2 is computed from 10000 trials and compared with the theoretical distribution for χ^2 for 99 degrees of freedom (the continuous curve)

distribution for 99 degrees of freedom. The results for the computer generated sequence are shown in Fig(??).

We have considered the correlations between the successive numbers above. Clearly, we can check for the correlations for triplet, quadruplet and in general n-tuplet of numbers. The test starts becoming either more time-consuming or more crude as the value of n increases. For example, in order to apply the test for quadruplets at the same accuracy as the doublets, we need to divide the four-dimensional space into n^4 parts. This means we have to either have more number of quadruplets in each sequence (so that we have reasonably large occurrence in each part) or we must choose smaller number of parts. Doing the test for triplets and, may be, quadruplets is feasible on the current computers.

There is, however, some reason to believe that absence of correlations in pairs of numbers would imply absence of it for higher n-tuplets. So, the tests for higher n-tuplets may appear to be superficial. Only reason for doing these is to ensure that the generator is not tweaked to ensure no pairwise correlations.

One can do visual tests for checking for possible short range correlations. For example, we can plot the pairs of numbers in a plane. If there is correlations, that will show up as some pattern in the plot. For example, in bad linear congruential random number generators one will see lines in the plot³. The problem is, this test is difficult to quantify.

³Actually, any linear congruential generator will have linear correlation between successive numbers. That is, the plot will always show slanted lines. If the period is sufficiently large, the lines are almost vertical so the correlation is not visible. A method of washing this correlation is to 'shuffle' the sequence.

2.3 Shuffling

We have discussed the correlation test in the preceding section. Generally we expect some correlation between successive numbers obtained from a generator. This is because a new number is generated from the preceding number by means of some algorithm. One can try to reduce or remove this correlation by doing shuffling. It is like shuffling of the deck of cards. We do this to remove the correlations among the neighbouring cards which arise during the game. Below we give an algorithm for shuffling.

The procedure is to first generate an array of random numbers computed by using the generator. The dimension of the array should be large enough for proper shuffling. One possible choice for the dimension is 97. One more number is computed and stored in a location (say y). When a random number routine is called, the number in location y is used to select an element of the array. This element is used as an output of the routine and it is also stored in y , the random number routine is called and its output is stored in the place of the element which was extracted from the array. The code which does these operations is as follows:

```
dimension array(97)
data init\0\
if(init .eq. 0)then
  do i=1,97
    array(i)=rand()
  end do
  y=rand()
  init=1
else
  j=y*97+1
  y=array(j)
  ran=y
  array(j)=rand()
  init=1
end if
```

Here the function `rand()` generates unshuffled sequence of random numbers and the function generating the shuffled random numbers is `ran`.

2.4 Central Limit Test

One can devise a number of tests for investigating random sequences. Some of these are described in Knuth's book^{??}. These have historical importance. These are also interesting

as exercises in statistics and probability theory. We shall not dwell on these more but some of you may want to work out the theory behind these tests and then apply them to random number generators. We shall consider one more test here because it has to do with the central limit theorem. The exact statement of the central limit theorem is, if one considers a number of elements from different statistical distributions and does some mathematical operations with them, then the resulting distribution is Gaussian in the limit of large number of elements under consideration. For applying the theorem, we shall add n successive elements obtained from the uniformly distributed sequence and add them up. In this case, we can show that the resulting distribution should peak at $\frac{n}{2}$ and the width should be \sqrt{n} . For uniformly distributed sequences, one can actually compute the resulting distribution for small n . But one finds that the central limit is reached reasonably well when one adds 10 numbers.

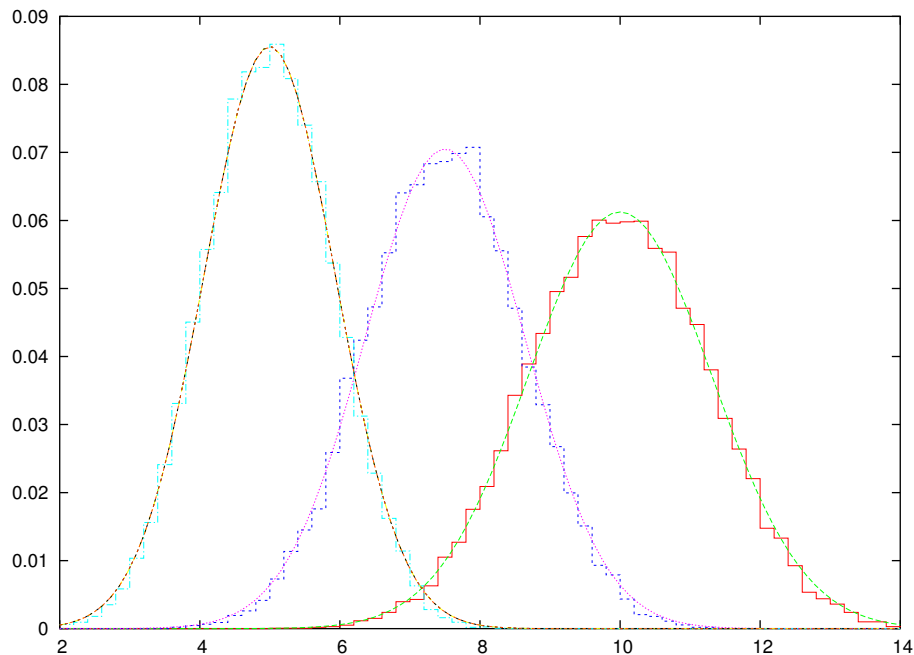


Figure 3: The results for central limit test when 10, 15 and 20 successive numbers are added and the distribution of the resulting numbers is plotted. The figure shows that a very good Gaussian fit is obtained for all cases.

We have applied the test for the machine-given random number generator by adding 10, 15 and 20 numbers at a time. To apply the test, we consider sets of 10, 15 and 20 successive numbers and add them. This is repeated 10,000 times. We expect the average of these trials to be 5, 7.5 and 10 respectively. We could also determine the variance defined by $\sqrt{\langle x^2 \rangle - \langle x \rangle^2}$ where x 's are the sum of 10, 15 and 20 numbers. What we have done in the figure (Fig(2.4)) is plot the measured probability of finding the x 's between

two values. We find that one gets a nice Gaussian fit to the histograms. One could apply χ^2 of Kolmogorov-Smirnov test (see Appendix?? for the details) to these results.. That will give a more quantitative information on the central limit test results.

2.5 Periods of Random Number Generators

The different tests described in the preceding section concern the proper distribution of the numbers in a random sequence and the short range correlations which may exist in the generators. One can device generators in which one reduces or removes the short range correlations. There are other kinds of correlations present in the generators developed for computers. These are the periodicities present in these generators. The physical generators do not have this problem since they depend on genuine random processes. But any algorithm in computer is deterministic. So, by any chance, a number produced in the generator is repeated, the following sequence is exact replica of the preceding sequence. So we end up with a periodic sequence. This is not what we want from a random number generator.

We shall now argue that one cannot avoid periodicity in random number generation algorithms used on computer. First, any algorithm would be of the form

$$x_{i+1} = f(x_i). \tag{3}$$

That is, given a number x_i , we have a single-valued function $f(x)$ which gives the next number, x_{i+1} in the sequence of random numbers. Given x_i , we get a well-defined value of x_{i+1} . Now, a number system on the computer has finite numbers (integers or real numbers). So, any algorithm we have, one will have repetition of one of the number in a sequence after some finite number of iterations. The best we can hope is that the numbers are repeated after visiting all possible numbers in the number system of the computer. That is the largest period we may have. Generally, the length of the period would be much smaller (and in some cases very small). This cannot be avoided and any random number generator would suffer from periodicity. Note that for the algorithms given in eq(3) above, the periods are not very large. Typically these are less than 10^9 or so.

Once we accept that the random number generators have built-in periodicities, how can one live with these? One thing is to use generators which have as long periods as possible. Secondly, one should not use these generators for sequences whose lengths are comparable to the lengths of the periods. In fact the lengths of the sequences should be at least an order of magnitude smaller than the period lengths.

3 Random Number Generator Algorithms

In this section we shall discuss the algorithms used for generating sequences of random numbers. One may think that if one has a complex algorithm, the sequence produced will

be a random sequence. That is not necessary. There must be some basis for an algorithm to produce a random (at least almost random) sequence. In fact, one would like the algorithm to be very simple and fast as that would make the generator computationally efficient. We also want an algorithm which will produce a sequence which is uniformly distributed in a unit interval. In the next section we shall discuss how we can obtain sequences which are distributed according to other distribution functions.

In my opinion, any algorithm which is chaotic would produce random sequences, except for possible short range correlations. The reasoning is as follows. In a chaotic algorithm (which is actually a map transforming a number into another number as per the algorithm), if one starts with two elements arbitrarily close to each other, the difference between the elements after successive iteration grows exponentially. That is the Lyapunov exponent is positive for the chaotic algorithm. This means that if one starts with a set of elements arbitrarily close to each other, they fill up the whole phase space in few iterations. In our case, the phase space is the set of numbers between 0 and 1. That is like mixing of milk in coffee. Initially the drop of milk is confined to a small region of space but on stirring it gets thoroughly mixed up. The process of stirring is a chaotic process and that leads to distribution of milk all over the configuration space. I would like to argue that in a chaotic algorithm, the iterative process leads to filling of the phase space in time. That is, starting with one number (seed), when the algorithm is applied repeatedly, the sequence fills up the phase space. For non-chaotic algorithm, on the other hand, the sequence would not fill the whole phase space, it would have orbits which occupy only a fraction of the phase space. It can therefore be argued that a chaotic algorithm would produce random sequences.

There is however one problem with most of the algorithms, namely there is always short range correlations. That is the successive numbers are correlated. One can correlate this with the Lyapunov exponent of the algorithm. Given a Lyapunov exponent λ , the distance between two elements after each iteration grows as e^λ . Thus, the correlation between two successive iterations decreases as λ increases. Even for a very large λ , one would expect some correlation for successive iterations. But as we consider every other, every third \dots numbers, the correlation would get washed.

The preceding discussion indicates that random number generators should use chaotic algorithms (algorithms which exhibit chaos in the sense of Lyapunov exponent) and these algorithms need not be complex at all. We shall consider some of these algorithms below.

3.1 Linear Congruential Algorithms

Consider an integer map

$$N_{i+1} = (aN_i + c) \bmod m \tag{4}$$

where a , c and m are integers and we seed the map with an integer x_0 and mod operation means dividing the number on left by the number on the right and keeping the remainder. The contention is, this map produces uniformly distributed random integers between 0 and $m-1$. This map is chaotic because if we begin with two integers differing by unity, after each iteration the difference grows by a factor of a and after certain number of iteration this would be of the order of m . Actually, the mod operation is responsible for making the map chaotic. One would need to show that the numbers are uniformly distributed between 0 and $m-1$. Being chaotic, we shall argue that it is a good candidate for generating random numbers.

We are interested in a sequence of random real numbers and not random integers. We can construct a real number between zero and unity by defining $x_i = \frac{N_i}{m}$. We make following points about this algorithm.

1. If the sequence $\{N_i\}$ is uniformly distributed, the sequence $\{x_i\}$ will also be uniformly distributed between 0 and 1.
2. The sequence $\{x_i\}$ cannot have 1 as its element but 0 can be.
3. The linear congruential algorithm is necessarily periodic.
4. The number of elements in the set of numbers $\{x_i\}$ is finite. This number is, in fact, smaller than the total number of numbers between 0 and 1 which can be represented on a computer if m is smaller than the largest possible integer one can have.
5. The largest possible period of the sequence is m .

We want to choose a , c and m such that the period of the squence generated by linear congruential algorithm is as large as possible. The best choice of m will be the largest integer that can be represented on a given computer. People have investigated the best choice for a and c . One can obtain conditions for the best choice to get largest possible period. One obvious, but not useful, choice is $a = 1$ and $c = 1$. With this the period will be m but N_i 's will not be random. It has been found that algorithm has period m if

1. c is a relative prime to m
2. $a - 1$ is a multiple of p for every prime p dividing m
3. $a - 1$ is multiple om 4 if m is multiple of 4

Different possible choices of a , c and m have been investigated. We shall not discuss this further. For more detailed studies of the choices of a , c and m , we refer to Knuth's book[?].

Below we list a random number generator which uses linear congruential algorithm. This particular function has been well studied and is known to pass a number of tests of random

number sequences, particularly when it is used with shuffling algorithm given in previous section.

```
function ran(idum)
parameter ( ia=16807, im=2147483647, am=1./im,
           1 iq=127773, ir=2836, mask=123459876)
data init/0/
if(data .eq. 0)then
  data=1
  idum1=idum
end if
idum1=ieor(idum1, mask)
k=idum1/iq
idum1=ia*(idum1-k*iq)-ir*k
if(idum1 .lt. 0)idum1=idum1+im
ran=idum1*am
idum1=ieor(idum1,mask)
return
end
```

3.2 Other Generators

One nice thing about linear congruential generators is that they are simple (linear operations) and fast. They also work reasonably well. Their limitations are not very large periods and and possible short range correlations. Short range correlations can be taken care of by shuffling. For increasing the period, one can use combination of two generators as discussed below. Instead of linear congruential generators one may devise generators using more complex algorithms. We shall list some of the well studied algorithms below. In the following section we consider a generator based on logistic map.

3.2.1 Quadratic Generators

In principle, we can replace the linear map by a map with some function of N_i . One may feel that such a function may produce better random sequences than the linear map. However, there doesn't seem to be any proof of that. People have, however investigated quadratic maps of the form

$$N_{i+1} = (dN_i^2 + aN_i + c) \bmod m \quad (5)$$

to determine best values of a , c , d and m to get maximum period. It is not obvious if this algorithm produces more random number than the linear congruential algorithm. One

special case of the quadratic generator is related to the 'middle square' algorithm proposed by von Neumann⁴.

3.2.2 Combination Generators

One can make linear congruential generator more complex by computing the next element in the series by using two preceding elements. Thus the algorithm is

$$N_{i+1} = (a(N_i + N_{i-1}) + c) \bmod m \quad (6)$$

The algorithm is essentially same as the linear algorithm so its 'randomness' is expected to be similar to that of the simple linear congruential generators. But its period will definitely be longer because we have period k when $N_{i+k} = N_i$ as well as $N_{i+k-1} = N_{i-1}$. For such algorithms, the period can be made as high as m^2 . A special case of this type of generator is the Fibonacci sequence,

$$N_{i+1} = (N_i + N_{i-1}) \bmod m. \quad (7)$$

This sequence does have a longer period but is not a good random number generator. People have tried similar algorithm but with i and $i - 1$ replaced by some preceding numbers in the sequence. For example, Mitchell and Moore have considered the sequence

$$N_{i+1} = (N_{i-24} + N_{i-55}) \bmod m. \quad (8)$$

The numbers 24 and 55 appear to be randomly selected but it seems there is some theory behind and this sequence gives a period of $2^{55} - 1$, a long period indeed!

Another method of increasing the period of a sequence is to combine two linear congruential sequences, having different periods (which are not multiples of each other, ideally relative primes) to generate the sequence by using mod operation of one of the sequences. In this case, one can show that the period of the combined generator is the least common multiple of the periods of the two sequences.

We have already discussed shuffling earlier. Now consider the following procedure. Take two linear congruential sequences $\{X_i\}$ and $\{Y_i\}$ with two different different periods. First fill up an array of certain dimension with $\{X_i\}$. Then use next element of $\{Y_i\}$ to pick the output random number from the array $\{X_i\}$. The element of the array is picked by the value $nY_i/m_y + 1$ where n is the dimension of the array and m_y is the modulus for array $\{Y_i\}$. This picked number is the element of the sequence of the random number and the next element of $\{X_i\}$ is put in the place of the picked element in the array. This procedure is identical

⁴von Neumann had proposed that the middle digits of a square of a number can be used as random number. Thus, the sequence is generated by taking a number of N digits and keeping the middle digits of the square as a seed for next iteration as well as an element of the sequence.

to shuffling but the shuffling is done by using another sequence. The consequence is that the period of the resulting sequence is longer. In fact it is the smallest common multiple of the periods of the two sequences. So this procedure gives more random sequence with much longer period.

4 Sequences With Nonuniform Distributions

So far we have discussed sequences having uniform distribution in interval $(0, 1)$. One often needs sequences which have other distributions and in this section we shall describe the methods of generating sequences with such distributions from the uniform sequences. The nonuniform sequences often required are

1. sequences having normal or Gaussian distributions with mean x_0 and variance σ .
2. sequences having Poisson or exponential distributions.
3. sequences having χ^2 distribution of n degrees of freedom.
4. sequences having uniform distribution in arbitrary interval (a, b) .
5. sequences of integers restricted between integers I_1 and I_2 .

Converting the floating point sequence to an integer sequence is straight forward. Given the uniformly distributed sequence $\{x_i\}$, the integer sequence $\{I_i\}$ is obtained by using the integer arithmetic. If we want a sequence of random integers between 0 and $N - 1$, we compute $I_i = N \times x_i$ where x_i is a sequence of uniformly distributed random numbers between 0 and 1. By virtue of integer arithmetic, the fractional part of the product $N \times x_i$ is truncated and we are left with a sequence of random integers between 0 and $N - 1$. For an integer sequence between N_1 and N_2 , we choose $N = N_2 - N_1$ and define the integer sequence to be $I_i = N \times x_i + N_1$. The sequence I_i is indeed random and is restricted between N_1 and N_2 .

Obtaining a uniformly distributed sequence $\{y_i\}$ which is distributed between (a, b) is obtained by defining $y_i = a + (b - a) \times x_i$. Since x_i 's are between $(0, 1)$, y_i 's are between (a, b) . The sequence $\{y_i\}$ is random by virtue of randomness of the sequence $\{x_i\}$. It is also uniformly distributed because an interval dx at x is mapped into an interval $dy = (b - a) \times dx$ at $y = a + (b - a) \times x$, which is a linear transformation and the sequence $\{x_i\}$ is uniformly distributed.

Let us now consider a situation when we want to have a random number sequence distributed according to a function $y(x)$ between limits x_1 and x_2 . We choose the function $y(x)$ to be a normalised is a distribution function. So $Y(x) = \int_{x_1}^x dx' y(x')$ is a monotonically increasing function going from 0 to 1 as x goes from x_1 to x_2 . This indicates an algorithm,

as illustrated in Fig(4), for getting the random sequence distributed according to $y(x)$ from a sequence $\{z_i\}$ which is uniformly distributed between 0 and one. Given an element z_i of the latter, the corresponding element x_i of the former satisfies an equation $z_i = Y(x_i)$ or $x_i = Y^{-1}(z_i)$. As a concrete example, consider the case of random number sequence between 0 and ∞ having distribution function $y(x) = \alpha \exp(-\alpha x)$. Thus, $Y(x) = 1 - \exp(-\alpha x)$ and therefore $x_i = \frac{1}{\alpha} \ln(1 - z_i)$.

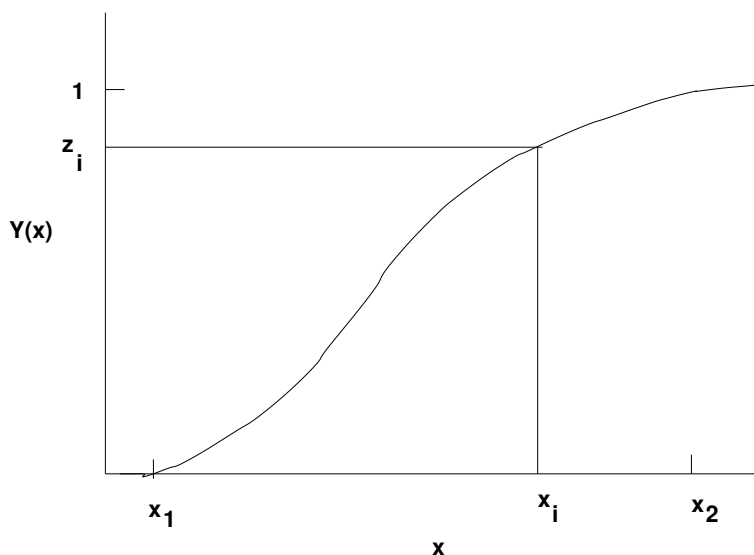


Figure 4: A schematic picture showing how to generate random number sequences having arbitrary distributions from a uniformly distributed sequence. The lower and upper limits of the distribution function are x_1 and x_2 respectively. The curve depicts the function $Y(x)$ referred to in the text. For every element z_i of the uniformly distributed sequence, corresponding element of the former is x_i . The procedure is to draw a horizontal line intersecting z_i on y axis and to draw a vertical line from the point where the first line intersects the curve $Y(x)$. The point where the second line intersects x axis gives the corresponding element of the random sequence of given distribution.

The procedure given above, although elegant, is useful only if the inverse of $Y(x)$ can be computed easily, as it happens in case of exponential distribution. A not so elegant (and possibly dangerous) method would be to maintain a table of $Y(x)$ for a large number of values of x and use interpolation technique to compute x_i 's from known z_i 's. The procedure is fast but may not be accurate enough if $Y(x)$ changes very slowly.

When the inverse function is not easy to compute and interpolation is not good, one uses the rejection method for computing the random sequence having a distribution function $y(x)$. The method assumes that one can find another function $f(x) \geq y(x)$ in the region

of interest and that an integral of $f(x)$ as well as its inverse can be readily computed (see Fig(4) for details). Let $F(x)$ be the integral of $f(x)$. The method works as follows. Let x_1 and x_2 be the lower and upper limits of the distribution and $\int_{x_1}^{x_2} dx f(x) = F \neq 1$. To generate a random number x_i of the new sequence, pick a number z_{2i+1} of the uniform sequence between 0 and F (not 0 and 1) and select x_i by using the procedure described in the caption of Fig(4). Now pick one more number z_{2i+2} which is distributed uniformly between 0 and 1. If $z_{2i+2} > y(x_i)/f(x_i)$, reject the number. Else accept x_i as a member of the desired sequence. The basic idea behind the rejection method is following. The first part gives the sequence which is distributed according to the distribution function $f(x)$. But we want a sequence which is distributed according to the function $y(x)$. The second half of the procedure accepts the number with the probability $y(x)/f(x)$. Thus, the accepted numbers have the desired probability distribution.

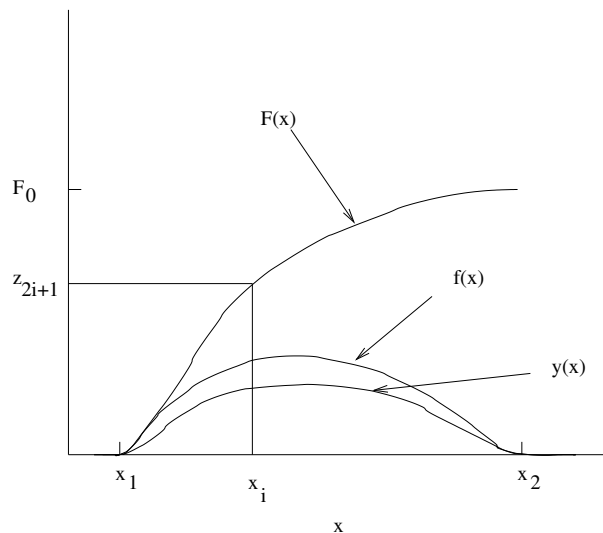


Figure 5: A schematic picture showing how to generate random number sequences using rejection method. $y(x)$ and $f(x)$ are the desired distribution function and enveloping function respectively and $F(x)$ is the integral of $f(x)$.

Finally we end this section by describing an algorithm for computing a sequence of random numbers which are having normal distribution. Let us consider the distribution function to be $y(x) = \exp(-x^2/2)$. Here the mean of the distribution is zero and the variance is unity. The transformation method cannot be used directly here because the integral of Gaussian is the error function and inverse of error function is not simple. For this, we consider a pair of random numbers x_{2i-1} and x_{2i} which are distributed uniformly

between 0 and 1. IN terms of these we define two quantities

$$y_1 = \sqrt{-2 \ln x_1} \cos 2\pi x_2$$

and

$$y_2 = \sqrt{-2 \ln x_1} \sin 2\pi x_2.$$

The product probability

$$\begin{aligned} p(x_1, x_2)dx_1dx_2 &= dx_1dx_2 \\ &= p(y_1, y_2)dy_1dy_2 \\ &= J(x_1, x_2; y_1y_2)dy_1dy_2 \end{aligned} \tag{9}$$

where $J(x_1, x_2; y_1y_2)$ is the Jacobian of the transformation;

$$J = \begin{vmatrix} \frac{\partial x_1}{\partial y_1} & \frac{\partial x_1}{\partial y_2} \\ \frac{\partial x_2}{\partial y_1} & \frac{\partial x_2}{\partial y_2} \end{vmatrix}.$$

The inverse transform from y 's to x 's is

$$x_2 = \frac{1}{2\pi} \tan^{-1}(y_2/y_1)$$

and

$$x_1 = e^{-(y_1^2+y_2^2)/2}.$$

and the Jacobian is

$$J = \frac{1}{2\pi} e^{-y_1^2/2} e^{-y_2^2/2}.$$

This means that both y_1 and y_2 are distributed with Gaussian distribution if x_1 and x_2 are uniformly distributed between 0 and 1. So the procedure for obtaining random numbers distributed according to Gaussian distribution is as follows. For every pair of random numbers x_1 and x_2 uniformly distributed between 0 and 1, compute y_1 and y_2 as prescribed above. The pair of numbers y_1 and y_2 are distributed according to Gaussian (normal) distribution with mean of zero and variace of unity.

5 Logistic Map As Random Number Generator

All the generators discussed in preceeding sections produce sequences of integers and these sequences are converted to real numbers in the interval (0,1). One advantage of working with integers is the operations with integers are faster on computer. Now we shall consider a generator which uses real number arithmetic. This is more of academic interest as, as far as I know, there is no pressing need for using another random number generator. One

reason for considering this generator is that it uses an algorithm in which chaos is built in. So, this, in a way, demonstrates the connection between chaos and randomness. Actually, the mod operation does introduce chaos. The example is the so called cat map in which one stretches and folds a picture (a picture of a cat and that's why the name). After few operations one finds that the picture is thoroughly mixed. You may have noticed that the process is same as mixing the dough. Any way, here we shall consider the logistic map in chaotic regime and we shall show that, after some massaging, it becomes a very good random number generator producing uniformly distributed sequence.

Consider a sequence

$$x_{i+1} = 4x_i(1 - x_i). \quad (10)$$

This is a quadratic map (unlike the linear integer maps we considered before). For $x_0 > 1$ and $x_0 < 0$, x_i 's rapidly run to $-\infty$ but for $0 < x_0 < 1$, x_i 's always remain between 0 and 1. The sequence given in eq(10) is a special case of the map, called logistic map, $x_{i+1} = \lambda x_i(1 - x_i)$ for $\lambda = 4$. This map is extensively studied and we shall discuss a few points relevant for our purpose. For $\lambda = 4$, the map is chaotic, that is the Lyapunov exponent is positive (in fact it is $\sqrt{2}$). The properties of the map become clear when we use a variable θ such that

$$x_i = \sin^2\left(\frac{\theta_i}{2}\right) = \frac{1}{2}(1 - \cos \theta_i) \quad (11)$$

We then have

$$\begin{aligned} x_{i+1} &= 4x_i(1 - x_i). \\ &= 4\sin^2\frac{\theta_i}{2}\cos^2\frac{\theta_i}{2} \\ &= \sin^2\theta_i \end{aligned} \quad (12)$$

Thus $\theta_{i+1} = \theta_i$. But θ_i , being an angle, should be taken modulo π . What we find is that the map ($x_{i+1} = 4x_i(1 - x_i)$) is equivalent to ($\theta_{i+1} = 2\theta_i \pmod{\pi}$) so the properties of the sequence $\{x_i\}$ can be deduced from the properties of the sequence $\{\theta_i\}$. The map ($\theta_{i+1} = 2\theta_i \pmod{\pi}$) is chaotic with Lyapunov exponent $\ln 2$ because $\delta\theta$ increase by factor of 2 on each iteration and $\delta\theta_i = 2^i\delta\theta_0 = \exp\{\ln 2\}^i\delta\theta_0$. Also note that the algorithm for the sequence θ_i is similar to the linear congruential algorithm except that the latter is for integers whereas the former is for real numbers. So, this could be a possible candidate for random number generator.

Why not use the sequence θ_i as a random number generator? The problem with this algorithm is that this sequence has very bad periodicities. We can show that any θ_i which is a rational fraction of π is periodic. But any irrational fraction is not!. So, it is not a good idea to use the sequence θ_i as a random number generator. On the other hand, the

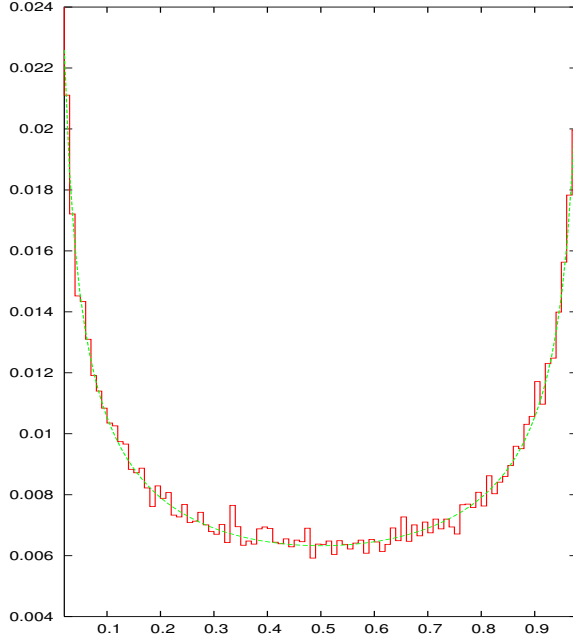


Figure 6: Distribution of the logmap sequence. The curve is fitted by $P(x) = \frac{1}{\pi\sqrt{x(1-x)}}$ as expected from the theory. Clearly, the sequence will fail the distribution test for a uniform sequence.

sequence x_i is not periodic when x_0 is a rational fraction except for a few special cases⁵. Thus, it is convenient to consider the sequence $\{x_i\}$ and not $\{\theta_i\}$ as a candidate generator.

Now the question is, how are the numbers in the sequence $\{x_i\}$ distributed. For a good generator, we would like the numbers to be distributed uniformly. What we can say is that the numbers in a sequence $\{\theta_i\}$ are distributed uniformly. We can deduce the distribution function for $\{x_i\}$ as follows. Assuming that these are distributed according to a function $f(x)$, we have $f(x)dx = d\theta/\pi$. Since $x = \sin^2 \theta/2$, $dx = \sin \theta/2 \cos \theta/2 = \sqrt{x(1-x)}$ and therefore $f(x) = \frac{1}{\pi\sqrt{x(1-x)}}$. In Fig(5) the distribution of the sequence $\{x_i\}$ obtained from the recurrence relation of eq(10) is plotted and a fit with the curve $f(x) = \frac{1}{\pi\sqrt{x(1-x)}}$ is shown. It clearly shows that the distribution is as expected.

We can now obtain the uniformly distributed sequence by an inverse transformation. Eq(11) gives relation between x and θ and since θ is defined between 0 and π , we define a variable $y = \theta/\pi = \frac{1}{\pi} \cos^{-1}(1-2x)$, so that y is distributed between 0 and 1. We also claim that the distribution is uniform. So, we have a sequence $\{y_i\}$ where $y_i = \frac{1}{\pi} \cos^{-1}(1-2x_i)$ and the sequence $\{x_i\}$ is generated by means of eq(10). This sequence has a strong correlation between successive elements but, as we have seen earlier, this correlation can be removed by shuffling. The results of distribution correlation tests for the sequence $\{y_i\}$ are shown in Fig(5).

As we see, both of these tests are satisfied well by the proposed generator. So, in

⁵ $x_0 = 0$ is one example with period 1. Other problem cases are $x_0 = 1$ and 0.5 because for these $x_1 = 0$ and $x_2 = 0$, respectively. All other periodic cases require x_0 to be irrational and any number represented on computer is necessarily a rational number.

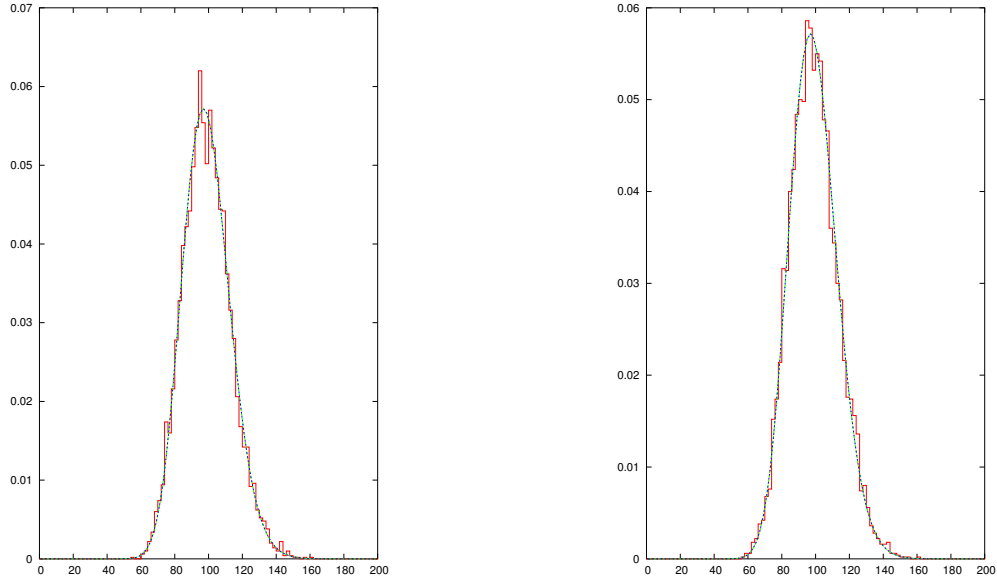


Figure 7: The results of distribution (left panel) and correlation (right panel) tests for a random number generator based on logistic map. Every fifth number from the logistic map sequence is chosen and shuffling of the numbers is done. For shuffling, another logistic map sequence is used. For distribution test, set of 1000 numbers are divided into 100 bins (between 0 and 1) and χ^2 statistic is computed. For correlation test, thousand pairs of numbers are divided into 100 equal sized bins and χ^2 statistic is computed. This is repeated 5000 times and the results are plotted as histograms. For comparison, the theoretical χ^2 distribution for 99 degrees of freedom is also shown as a continuous line.

principle, we can use the random number generator based on logistic map in our calculations. We had argued that the generator would be periodic. That is not really correct because of the finite size used in representing a real number on computers. Unlike linear congruential generators, we do not have a well defined theory behind the periods of the logistic map generator. Numerical investigations reveal that, depending on the initial seed x_0 , one has different periods and the sequence goes into a periodic loop after certain number of (not well-defined) iterations. We did a large number of trials and we found that the number of periods appear to be very small. We could locate less than 10 periods. We also found that the lengths of the periods are $\sim 10^3$ for single precision computations and $\sim 10^9$ for double precision calculations. Our conjecture is that the map goes into periodic loop when x_i become small and close to zero. We claim that this happens because when $x_{i-1} \sim 1$, there is a loss of accuracy when one computes $1 - x$ in eq(10) and that probably leads to degenerate x_i and therefore periodicity. That probably explains why the lengths of the period are about half of the accuracy of the numbers in single and double precision.

The main problem with the logistic map generator is that it is rather slow. It is slow,

primarily because, to get uniformly distributed sequence, we need to compute inverse cosine of the numbers generated in logistic map sequence of eq(10). This operation is an order of magnitude slower than floating point multiplication and addition. Whereas the linear congruential generators are much faster as the integer multiplication and addition as well as modulus operation is fast. Because of this problem we cannot recommend the logistic map generator for heavy duty simulation work.

6 Numerical Simulations

We shall now come to the topic of numerical simulations in physics where random number generators are used. There are some situations in which use of random number sequences is natural. One example is the computation of statistical properties. Another is where one is computing a quantum process. In many of such processes, one needs to compute the probability of occurrence of an event and then its consequences. In some processes, the system is deterministic but because of complexity of the problem we take recourse to random sequences. We shall consider some examples of simulations. These are not exhaustive but hopefully these give some idea of how one can go about designing numerical simulations, develop programs for carrying these out etc.

6.1 Numerical Integration — Monte Carlo method

The example of numerical integration is an example in which use of random sequences is one of the methods. It is a sort of simulation because we 'simulate' the actual integral by a monte Carlo Simulation. Consider a one dimensional definite integral

$$I = \int_a^b f(x)dx \tag{13}$$

By definition, the integral is

$$I = \lim_{n \rightarrow \infty} \sum_{i=1, n} \frac{b-a}{n} f\left(a + \frac{i+1/2}{n}(b-a)\right) \tag{14}$$

and equivalently, we can also write it is $I = (b-a)\bar{f}$ where \bar{f} is the mean value of the function in the interval (a, b) . Usually, the numerical integrations are performed using the definition of the integration given in eq(14) or its more accurate variant⁶. Instead of this standard method, we may evaluate the integral by estimating the average value of the

⁶One cannot take the limit $n \rightarrow \infty$ on computer. So what one does is to evaluate the sum for finite n . The value chosen is such that the error is tolerable. Basically the error is proportional to the derivative of the function and inversely proportional to square of n . There are methods in which the error is inversely proportional to higher powers on n and then the integration is more accurate

function in the interval. Let us consider a sequence of random numbers $\{x_i\}$ which are distributed uniformly in the interval (a, b) . The average value of the function can then be estimated as $\sum_{i=1, N} f(x_i)/N$ where N is the number of elements in the sequence chosen for the evaluation. Then the estimate of the integral is

$$I = \frac{b-a}{N} \sum_{i=1, N} f(x_i) \quad (15)$$

The formula essentially follows from the mean value theorem of integration. One can also understand the formula as follows. If we take a small interval dx within (a, b) , the contribution from this region to the integral is $dx\bar{f}(x)$ where $\bar{f}(x)$ is average value of the function in the interval dx . The number of points falling in this interval is $n = dx/(b-a) \times N$. Now the average value of the function in dx is $\frac{\sum_i f(x_i)}{n} = \frac{b-a}{Ndx} \sum_i f(x_i)$ where the sum is over all x_i 's falling in the interval dx . Thus, the contribution from the region dx to the integral is $\frac{(b-a)}{N} \sum_i f(x_i)$. Clearly to get the formula in eq(15), we need to sum over all the intervals dx which is equivalent to summation over all the points uniformly distributed in interval (a, b) .

Now we come to the estimation of the error in the value of the integral. Since the Monte Carlo procedure is essentially a statistical averaging process, typically the error is proportional to the inverse of the square root of N . Clearly, for a one dimensional integration, the Monte Carlo method is not a very efficient procedure in comparison with the standard methods of numerical integrations. That is, even though the procedure is correct, the approach to the exact value of the integral is rather slow in comparison with the methods which are based on eq(14) or their variants. For example, typically one needs between 10 and 100 points where the integrand is evaluated, for a reasonably accurate evaluation of a one dimensional integral⁷ and generally the accuracy is better than 1%. To get similar level of accuracy by Monte Carlo integration, we may need to evaluate the integrand at about 10^4 points or more. However, the Monte-Carlo integration becomes competitive when one is doing multi-dimensional integrations and when the integrand does not separable. The reason is, while doing d -dimensional integration, we may need $\sim 10^{2d}$ number of integrand evaluations to get an accuracy of 1% or so. That is because integration along each dimension is like an independent integral. But in Monte Carlo method, the error scales like $1/\sqrt{N}$ where N is the number of function evaluations in the formula of eq(15). Thus, the number of function evaluations required to get 1% accuracy are still $\sim 10^4$ or so! Actually, these may be an order of magnitude more, if the integration boundary is complicated, but still much smaller than what is required for standard integration calculations.

⁷Actually the number of points required to do an integral to desired accuracy depend on the range of the integration and the structure of the integrand.

One can also use rejection method, described earlier, for Monte Carlo integration. Consider an integral $\int_{x_1}^{x_2} dx f(x)$ with $f(x)$ being positive and nonsingular⁸. Let f_0 be the maximum value of $f(x)$ in the interval (x_1, x_2) . We now have a rectangle of height f_0 and width $x_2 - x_1$. We now draw a pair of uniformly distributed random numbers, x_{2i+1} and x_{2i+2} with x_{2i+1} being distributed between x_1 and x_2 and x_{2i+2} being distributed between 0 and f_0 . The integration is then defined as

$$\int_{x_1}^{x_2} dx f(x) = \sum' f(x_{2i+1}) \quad (16)$$

where ' on summation indicates that the sum is restricted to those x_{2i+1} for which $x_{2i+2} < f(x_{2i+1})$.

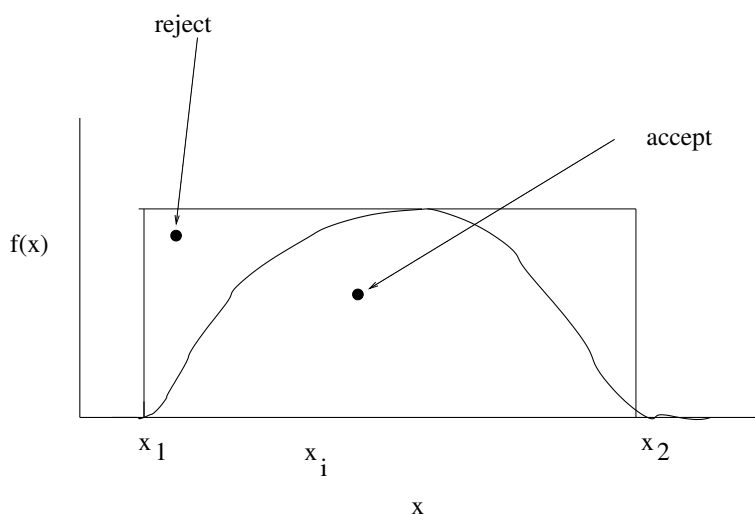


Figure 8: A schematic picture illustrating the rejection method of Monte Carlo integration. The points falling inside the curve are included in the summation in eq(16). The points falling outside are rejected.

The rejection method, which is illustrated in Fig(6.1), actually uses an interpretation of integration, namely an integral $\int_{x_1}^{x_2} dx f(x)$ is the area under the curve $f(x)$ between limits x_1 and x_2 . The pair of numbers x_{2i+1} and x_{2i+2} constitute a point in the rectangle of width $x_2 - x_1$ and height f_0 . In the summation above we include all the points which are below the curve and reject the points which are outside. Thus one is computing the area under the curve and therefore the integral. It should be obvious how one can generalize the method for multi-dimensional integrals as well as for functions having positive and negative values

⁸The restriction of positivity of the function can be relaxed but singular functions, having integrable singularity cannot be integrated by Monte Carlo methods

in the region of integration. The estimated error in rejection method is proportional to $1/\sqrt{n}$ where n is the number of random pairs selected for calculation.

6.2 Radioactive Decay

In this subsection we shall consider a simulation of a well-known physical process, namely radioactive decay of nuclei. The exponential decay law of radioactivity is well known and it follows from quantum mechanical considerations. It has also been verified. Here we shall consider a computer simulation of the process. One really doesn't need to do this because the decay law is derived analytically. None the less, the simulation is very simple and it tells us how one can go about doing the simulations on computers.

From quantum mechanics, we find that the probability of the decay of a nucleus is constant and independent of the number of nuclei one has in a sample. This probability depends on the transition matrix element of one nuclear state to another with emission of an α particle. Actually, the same principle holds for β -decay, emission of photons from excited atomic or nuclear states etc. Thus the number of decays occurring in a unit time ($\frac{dN(t)}{dt}$) is given by $\lambda N(t)$ where λ is the decay probability and $N(t)$ is the number of active nuclei at time t . Thus, the decrease in the number of nuclei in unit time satisfies the differential equation

$$\frac{dN(t)}{dt} = -\lambda N(t) \quad (17)$$

and the solution of this equation is $N(t) = N(t=0) \exp\{-\lambda t\}$. There are cases where a nucleus A decays into nucleus B with decay rate λ_a and the nucleus B, in turn, decays into nucleus C with decay rate λ_b and so on. In that case, one gets a set of coupled first order differential equations to be solved. These can, of course, be solved analytically. In the following we shall see how a numerical simulation of such a process is done. As a concrete example, we shall consider nucleus A decaying with rate λ_a and the daughter nucleus B decaying with rate λ_b .

Let us begin with the assumption that at time $t = 0$ we have some large number (N_0) of nuclei of species A and none of species B. Since the decay rate of nucleus A is λ_a , the probability of decay of any given nucleus in unit time is λ_a . In the simulation, we generate random numbers distributed uniformly between 0 and 1⁹. The procedure of computing which of N_0 nuclei decay is, we draw a random number and if it is smaller than λ_a , we say that the nucleus decays and not otherwise. We repeat this for all N_0 nuclei. So, at the end of the process, we have certain dN_0 number of nuclei decaying. This gives the number of nuclei of type A and B after unit time ($N_0 - dN_0$ and dN_0 respectively). We now repeat this for the next unit time interval to compute the number of nuclei of type A *AS WELL*

⁹It so happens that generally random numbers generate such random sequences so we don't have to do anything more

AS of type B decaying. The process continues till one ends up with very small number of nuclei of type A and B. The result of this computer experiment is a table of numbers giving the number of nuclei of type A and B as a function of time. From this table one can compute the values of the decay rates.

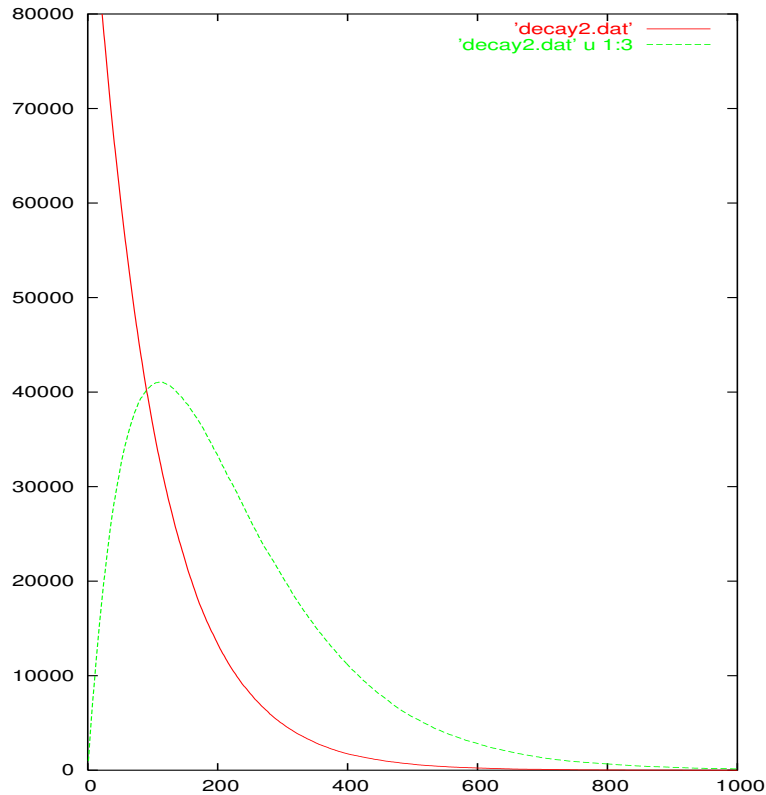


Figure 9: Result of the radioactive reactions $A \rightarrow B$ followed by $B \rightarrow C$. To begin with we start with $5 \cdot 10^7$ nuclei of type A and none of type B. Number of counts of decay of nuclei A and B are plotted.

The results of a simulation calculation based on the procedure outlined above are shown in Fig(6.2). The calculation considers reactions $A \rightarrow B$ followed by $B \rightarrow C$. The plot shows the number of decays of nuclei A and B (red and blue curves respectively). The decay rates for the decay of species A and B are 0.01 and 0.03 respectively. The exponential fall off is clearly indicated in the graph.

6.3 Ising Model Simulation

We shall now consider a well-studied model in condensed matter physics. Consider a system of spin 1/2 particles in d dimensions arranged on a specific lattice. To be specific, let us

consider a cubic lattice. We shall consider two projections of the spin and consider a Hamiltonian in which there is an external magnetic field interacting with the spins and the spins interacting with the nearest neighbours with interaction proportional to the product of individual spin projections. Thus the sign of the interaction is opposite for parallel and antiparallel configurations. Specifically, for a one dimensional chain, the Hamiltonian is

$$H = -J \sum_i s_i s_{i+1} - \mu B \sum_i s_i. \quad (18)$$

Ideally, one has an infinite chain but in simulation one has finite number of spins. One usually assumes that the last spin couples to the first spin, thus imposing cyclic boundary condition. If the strength of spin-spin interaction J is positive, the energy is lowered by aligning the spins (ferromagnetic interaction). For negative J we have anti-ferromagnetic interaction. This model is called the Ising model and exact solutions for one and two dimensions exist. So, as such, there is no interest in doing computer simulations for these systems except to show that the simulation calculations work. However, if one changes the Hamiltonian by, say, including the next-to-nearest neighbour interaction or considering two different types of spins etc, the simulation calculations essentially remain same. Thus, the generalizations of Ising model simulations offer a very useful tool for studies in condensed matter. The algorithm, which we will discuss below, is quite simple and it is not very difficult to write a computer code for it.

What we want to study is the thermodynamic properties of the system described above. We shall be having a finite lattice and we shall be interested in doing the computations for lattices of different sizes to deduce the properties of infinite lattices. We shall be assuming that the system is interacting with an external heat bath so the temperature of the lattice is fixed but not the energy of the system. For such systems, the quantity of interest is the partition function defined as

$$\begin{aligned} Z(\beta) &= \text{tr } e^{-\beta H} \\ &= \sum_{\alpha} e^{-\beta E_{\alpha}} \end{aligned} \quad (19)$$

where $\beta = 1/T$ is the inverse temperature (we take the Boltzmann constant k_B to be unity) and the summation is over all possible configurations α of the spins on the lattice. The observable corresponding to any operator O is

$$\langle O \rangle = \frac{\sum_{\alpha} O_{\alpha,\alpha} e^{-\beta E_{\alpha}}}{Z(\beta)} \quad (20)$$

where $O_{\alpha,\alpha}$ is the expectation value of O in the spin configuration α . Some of the quantities of interest are internal energy of the system, magnetization and their derivatives with respect to temperature. The derivatives give specific heat, etc.

The brute force method would be to consider all possible configurations of the system and evaluate the quantities of interest. That is not economical because, while computing averages, the sum is weighted by exponential factors and the configurations with large energies give very small contribution to the sums. But more important thing is, the number of configurations increase very rapidly with the number of lattice points used in the computations. For one dimensional lattice of n spins, there are 2^n different possible configurations. For 30 spins we have $2^{32} \sim 10^9$ configurations. That is bad enough already. But for two or three dimensional lattices things get out of hand for very few number of spins in each direction. So the brute force method is out of question and one has to use techniques which will somehow select the configurations which are lower in energy (so that the important contributions are selected) in the computations. One of the technique which achieves this is the Metropolis algorithm. We shall discuss that below.

The computation of “best” configurations goes as follows. One selects a configuration of spins to begin with. It could be what is called as a “cold” start where we align all the spins or it could be a “hot” start where one assigns the spin projections randomly or it could be any other choice. One then computes the energy of this configuration. The next series of steps, which are repeated a number of times, is the Metropolis algorithm. The steps are

1. randomly select a spin and flip its spin projection
2. compute the energy of the system
3. if the energy of the new configuration is lower than the energy of the old configuration, keep the new configuration for next iteration
4. if the energy of the new configuration is higher, compute $\exp[\beta(E_{old} - E_{new})]$. This number is between 0 and 1 since $E_{old} < E_{new}$. Draw a random number which is uniformly distributed between 0 and 1 and if this number is smaller than $\exp[\beta(E_{old} - E_{new})]$, keep the new configuration for next iteration. Else keep the old configuration.

The steps outlined above are repeated a large number of times till the energy of the system “stabilizes”. What we mean by that is it fluctuates about a mean value and the fluctuations are of the order of $1/\beta$. At that stage the configuration is selected for further computations. Typically, the number of iterations required for stabilization are about $10n$ where n is the number of spins. One needs more iterations at lower temperatures.

The procedure outlined above gives one configuration of the system. We need to repeat the process for a large number of times to select sufficiently large number of initial configurations so that we have sufficiently large number of configurations available for averaging. The computation of averages is essentially a statistical calculation. So, if we use N trial configurations, the estimated error in the results decreases as inverse of \sqrt{N} .

Why does the Metropolis algorithm work? In particular, why do we permit an increase in the energy of the system during the process of obtaining a configuration? There are two reasons for that. First, the system is in thermal contact with a heat bath. Thus the energy of the system is not fixed and it fluctuates. That is, the system is not at the minimum energy configuration. If we did not permit increase in energy, the system will try to reach an energy minimum. At least it will reach a local minimum. But that is not what we want for a thermodynamic system. Secondly, when the system is in a thermodynamic equilibrium with the heat bath, it continuously changes its state due to exchange of energy with the heat bath and it explores all possible phase space, with the probability related to the Boltzmann factor $\exp(-\beta E_\alpha)$. That is why we need to include the possibility of increase in the energy of the system during evolution. That ensures that all possible states are explored by the system. However, because of the rejection algorithm, the fluctuations in the energy are restricted to the energy differences of the order of the temperature of the heat bath. The rejection algorithm also ensures that the configurations with large energy are automatically rejected and only those configurations for which the Boltzmann factor is reasonably large are retained.

Apart from the use of Metropolis algorithm in statistical mechanics calculations, one can also use it in finding a minimum of a multi-dimensional function. It is particularly useful when the function has a large number of secondary minima and one wants to locate the absolute minimum. Because the algorithm permits going “uphill” (with a statistical factor), one will not get stuck in a local minimum and after sufficient number of steps one is likely to end up in the absolute minimum.