Chapter 8

Outline of the Monte-Carlo strategy

> 'Iacta Alea est', the die is cast, is what Julius Caesar is reported by Suetonius to have said on January 10, 49 BC as he led his army across the River Rubicon in Northern Italy. (Twelve Ceasars)*Gaius Suetonius*

8.1 Introduction

Monte Carlo methods are nowadays widely used, from the integration of multi-dimensional integrals to solving ab initio problems in chemistry, physics, medicine, biology, or even Dow-Jones forecasting. Computational finance is one of the novel fields where Monte Carlo methods have found a new field of applications, with financial engineering as an emerging field, see for example Refs. [38, 39]. Emerging fields like econophysics [40, 41, 42] are new examples of wide applications of Monte Carlo methods.

Numerical methods that are known as Monte Carlo methods can be loosely described as statistical simulation methods, where statistical simulation is defined in quite general terms to be any method that utilizes sequences of random numbers to perform the simulation. As mentioned in the introduction to this text, a central algorithm in Monte Carlo methods is the Metropolis algorithm, ranked as one of the top ten algorithms in the last century. We discuss this algorithm in the next chapter.

Statistical simulation methods may be contrasted to conventional numerical discretization methods, which typically are applied to ordinary or partial differential equations that describe some underlying physical or mathematical system. In many applications of Monte Carlo, the physical process is simulated directly, and there is no need to even write down the differential equations that describe the behavior of the system. The only requirement is that the physical (or mathematical) system be described by probability distribution functions (PDF's). Once the PDF's are known, the Monte Carlo simulation can proceed by random sampling from the PDF's. Many simulations are then performed (multiple "trials" or "histories") and the desired result is taken as an average over the number of observations (which may be a single observation or perhaps millions of observations). In many practical applications, one can predict the statistical error (the "variance") in this average result, and hence an estimate of the number of Monte Carlo trials that are needed to achieve a given error. If we assume that the physical system can be described by a given probability density function, then the Monte Carlo simulation can proceed by sampling from these PDF's, which necessitates a fast and effective way to generate random numbers uniformly distributed on the interval [0,1]. The outcomes of these random samplings, or trials, must be accumulated or tallied in an appropriate manner to produce the desired result, but the essential characteristic of Monte Carlo is the use of random sampling techniques (and perhaps other algebra to manipulate the outcomes) to arrive

at a solution of the physical problem. In contrast, a conventional numerical solution approach would start with the mathematical model of the physical system, discretizing the differential equations and then solving a set of algebraic equations for the unknown state of the system. It should be kept in mind though, that this general description of Monte Carlo methods may not directly apply to some applications. It is natural to think that Monte Carlo methods are used to simulate random, or stochastic, processes, since these can be described by PDF's. However, this coupling is actually too restrictive because many Monte Carlo applications have no apparent stochastic content, such as the evaluation of a definite integral or the inversion of a system of linear equations. However, in these cases and others, one can pose the desired solution in terms of PDF's, and while this transformation may seem artificial, this step allows the system to be treated as a stochastic process for the purpose of simulation and hence Monte Carlo methods can be applied to simulate the system.

There are, at least four ingredients which are crucial in order to understand the basic Monte-Carlo strategy. These are

1. Random variables,

2. probability distribution functions (PDF),

3. moments of a PDF

4. and its pertinent variance σ.

All these topics will be discussed at length below. We feel however that a brief explanation may be appropriate in order to convey the strategy behind a Monte-Carlo calculation. Let us first demistify the somewhat obscure concept of a random variable. The example we choose is the classic one, the tossing of two dice, its outcome and the corresponding probability. In principle, we could imagine being able to determine exactly the motion of the two dice, and with given initial conditions determine the outcome of the tossing. Alas, we are not capable of pursuing this ideal scheme. However, it does not mean that we do not have a certain knowledge of the outcome. This partial knowledge is given by the probablity of obtaining a certain number when tossing the dice. To be more precise, the tossing of the dice yields the following possible values

$$[2,3,4,5,6,7,8,9,10,11,12]. \tag{8.1}$$

These values are called the *domain*. To this domain we have the corresponding *probabilities*

$$[1/36,2/36/3/36,4/36,5/36,6/36,5/36,4/36,3/36,2/36,1/36]. \tag{8.2}$$

The numbers in the domain are the outcomes of the physical process tossing the dice. *We cannot tell beforehand whether the outcome is 3 or 5 or any other number in this domain. This defines the randomness of the outcome, or unexpectedness or any other synonimous word which encompasses the uncertitude of the final outcome.* The only thing we can tell beforehand is that say the outcome 2 has a certain probability. If our favorite hobby is to spend an hour every evening throwing dice and registering the sequence of outcomes, we will note that the numbers in the above domain

$$[2,3,4,5,6,7,8,9,10,11,12], \tag{8.3}$$

appear in a random order. After 11 throws the results may look like

$$[10,8,6,3,6,9,11,8,12,4,5]. \tag{8.4}$$

Eleven new attempts may results in a totally different sequence of numbers and so forth. Repeating this exercise the next evening, will most likely never give you the same sequences. Thus, we say that the outcome of this hobby of ours is truely random.

Random variables are hence characterized by a domain which contains all possible values that the random value may take. This domain has a corresponding PDF.

To give you another example of possible random number spare time activities, consider the radioactive decay of an α-particle from a certain nucleus. Assume that you have at your disposal a Geiger-counter which registers every 10 ms whether an α-particle reaches the counter or not. If we record a hit as 1 and no observation as zero, and repeat this experiment for a long time, the outcome of the experiment is also truely random. We cannot form a specific pattern from the above observations. The only possibility to say something about the outcome is given by the PDF, which in this case is the well-known exponential function

$$\lambda \exp -(\lambda x), \tag{8.5}$$

with λ being proportional to the half-life of the given nucleus which decays.

Good texts on Monte Carlo methods are the monographs of Robert and Casella, Johnson and Fishman, see Refs. [43, 44, 45].

8.1.1 First illustration of the use of Monte-Carlo methods, crude integration

With this definition of a random variable and its associated PDF, we attempt now a clarification of the Monte-Carlo strategy by using the evaluation of an integral as our example.

In chapter 7 we discussed standard methods for evaluating an integral like

$$I = \int_0^1 f(x)dx \approx \sum_{i=1}^{N} \omega_i f(x_i), \tag{8.6}$$

where ω_i are the weights determined by the specific integration method (like Simpson's or Taylor's methods) with x_i the given mesh points. To give you a feeling of how we are to evaluate the above integral using Monte-Carlo, we employ here the crudest possible approach. Later on we will present slightly more refined approaches. This crude approach consists in setting all weights equal 1, $\omega_i = 1$. That corresponds to the rectangle method presented in Eq. (7.5), displayed again here

$$I = \int_a^b f(x)dx \approx h\sum_{i=1}^{N} f(x_{i-1/2}),$$

where $f(x_{i-1/2})$ is the midpoint value of f for a given value $x_{i-1/2}$. Setting $h = (b-a)/N$ where $b = 1$, $a = 0$, we can then rewrite the above integral as

$$I = \int_0^1 f(x)dx \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i), \tag{8.7}$$

where x_i are the midpoint values of x. Introducing the concept of the average of the function f for a given PDF $p(x)$ as

$$\langle f\rangle = \frac{1}{N}\sum_{i=1}^{N} f(x_i)p(x_i), \tag{8.8}$$

and identify $p(x)$ with the uniform distribution, viz $p(x) = 1$ when $x \in [0, 1]$ and zero for all other values of x. The integral is is then the average of f over the interval $x \in [0, 1]$

$$I = \int_0^1 f(x)dx \approx \langle f \rangle. \tag{8.9}$$

In addition to the average value $\langle f \rangle$ the other important quantity in a Monte-Carlo calculation is the variance σ^2 and the standard deviation σ. We define first the variance of the integral with f for a uniform distribution in the interval $x \in [0, 1]$ to be

$$\sigma_f^2 = \frac{1}{N}\sum_{i=1}^{N}(f(x_i) - \langle f \rangle)^2 p(x_i), \tag{8.10}$$

amd inserting the uniform distribution this yields

$$\sigma_f^2 = \frac{1}{N}\sum_{i=1}^{N} f(x_i)^2 - \left(\frac{1}{N}\sum_{i=1}^{N} f(x_i)\right)^2, \tag{8.11}$$

or

$$\sigma_f^2 = \left(\langle f^2 \rangle - \langle f \rangle^2\right). \tag{8.12}$$

which is nothing but a measure of the extent to which f deviates from its average over the region of integration. The standard deviation is defined as the square root of the variance. If we consider the above results for a fixed value of N as a measurement, we could however recalculate the above average and variance for a series of different measurements. If each such measumerent produces a set of averages for the integral I denoted $\langle f \rangle_l$, we have for M measurements that the integral is given by

$$\langle I \rangle_M = \frac{1}{M}\sum_{l=1}^{M}\langle f \rangle_l. \tag{8.13}$$

We show in section 8.3 that if we can consider the probability of correlated events to be zero, we can rewrite the variance of these series of measurements as (equating $M = N$)

$$\sigma_N^2 \approx \frac{1}{N}\left(\langle f^2 \rangle - \langle f \rangle^2\right) = \frac{\sigma_f^2}{N}. \tag{8.14}$$

We note that the standard deviation is proportional with the inverse square root of the number of measurements

$$\sigma_N \sim \frac{1}{\sqrt{N}}. \tag{8.15}$$

The aim of Monte Carlo calculations is to have σ_N as small as possible after N samples. The results from one sample represents, since we are using concepts from statistics, a 'measurement'.

The scaling in the previous equation is clearly unfavorable compared even with the trapezoidal rule. In the previous chapter we saw that the trapezoidal rule carries a truncation error $O(h^2)$, with h the step length. In general, methods based on a Taylor expansion such as the trapezoidal rule or Simpson's rule have a truncation error which goes like $\sim O(h^k)$, with $k \geq 1$. Recalling that the step size is defined as $h = (b - a)/N$, we have an error which goes like $\sim N^{-k}$.

However, Monte Carlo integration is more efficient in higher dimensions. To see this, let us assume that our integration volume is a hypercube with side L and dimension d. This cube contains hence

$N = (L/h)^d$ points and therefore the error in the result scales as $N^{-k/d}$ for the traditional methods. The error in the Monte carlo integration is however independent of d and scales as $\sigma \sim 1/\sqrt{N}$, always! Comparing this error with that of the traditional methods, shows that Monte Carlo integration is more efficient than an order-k algorithm when $d > 2k$. In order to expose this, consider the definition of the quantum mechanical energy of a system consisting of 10 particles in three dimensions. The energy is the expectation value of the Hamiltonian H and reads

$$E = \frac{\int d\mathbf{R}_1 d\mathbf{R}_2 \dots d\mathbf{R}_N \Psi^*(\mathbf{R_1}, \mathbf{R}_2, \dots, \mathbf{R}_N) H(\mathbf{R_1}, \mathbf{R}_2, \dots, \mathbf{R}_N) \Psi(\mathbf{R_1}, \mathbf{R}_2, \dots, \mathbf{R}_N)}{\int d\mathbf{R}_1 d\mathbf{R}_2 \dots d\mathbf{R}_N \Psi^*(\mathbf{R_1}, \mathbf{R}_2, \dots, \mathbf{R}_N) \Psi(\mathbf{R_1}, \mathbf{R}_2, \dots, \mathbf{R}_N)},$$

where Ψ is the wave function of the system and $\mathbf{R}_i$ are the coordinates of each particle. If we want to compute the above integral using for example Gaussian quadrature and use for example ten mesh points for the ten particles, we need to compute a ten-dimensional integral with a total of 10^{30} mesh points. As an amusing exercise, assume that you have access to today's fastest computer with a theoretical peak capacity of more than 100 Teraflops, that is 10^{14} floating point operations per second. Assume also that every mesh point corresponds to one floating operation per second. Estimate then the time needed to compute this integral with a traditional method like Gaussian quadrature and compare this number with the estimated lifetime of the universe, $T \approx 4.7 \times 10^{17}$s. Do you have the patience to wait?

We end this first part with a discussion of a brute force Monte Carlo program which integrates

$$\int_0^1 dx \frac{4}{1+x^2} = \pi, \tag{8.16}$$

where the input is the desired number of Monte Carlo samples. Note that we transfer the variable $idum$ in order to initialize the random number generator from the function $ran0$. The variable $idum$ gets changed for every sampling. This variable is called the *seed*.

What we are doing is to employ a random number generator to obtain numbers x_i in the interval $[0, 1]$ through a call to one of the library functions $ran0$, $ran1$, $ran2$ or $ran3$ which generate random numbers in the interval $x \in [0, 1]$. These functions will be discussed in the next section. Here we simply employ these functions in order to generate a random variable. All random number generators produce pseudo-random numbers in the interval $[0, 1]$ using the so-called uniform probability distribution $p(x)$ defined as

$$p(x) = \frac{1}{b-a}\Theta(x-a)\Theta(b-x), \tag{8.17}$$

with $a = 0$ og $b = 1$. If we have a general interval $[a, b]$, we can still use these random number generators through a change of variables

$$z = a + (b-a)x, \tag{8.18}$$

with x in the interval $[0, 1]$.

The present approach to the above integral is often called 'crude' or 'Brute-Force' Monte-Carlo. Later on in this chapter we will study refinements to this simple approach. The reason for doing so is that a random generator produces points that are distributed in a homogenous way in the interval $[0, 1]$. If our function is peaked around certain values of x, we may end up sampling function values where $f(x)$ is small or near zero. Better schemes which reflect the properties of the function to be integrated are thence needed.

The algorithm is as follows

– Choose the number of Monte Carlo samples N.

- Perform a loop over N and for each step generate a a random number x_i in the interval $[0,1]$ trough a call to a random number generator.
- Use this number to evaluate $f(x_i)$.
- Evaluate the contributions to the mean value and the standard deviation for each loop.
- After N samples calculate the final mean value and the standard deviation.

The following C/C++ program[1] implements the above algorithm using the library function $ran0$ to compute π. Note again the inclusion of the $lib.h$ file which has the random number generator function **ran0**.

http://folk.uio.no/mhjensen/fys3150/2005/programs/chapter8/program1.cpp

```
#include <iostream>
#include "lib.h"
using namespace std;

//      Here we define various functions called by the main program
//      this function defines the function to integrate

double func(double x);

//      Main function begins here
int main()
{
     int i, n;
     long idum;
     double crude_mc, x, sum_sigma, fx, variance;
     cout << "Read in the number of Monte-Carlo samples" << endl;
     cin >> n;
     crude_mc = sum_sigma=0. ; idum=-1 ;
//     evaluate the integral with the a crude Monte-Carlo method
      for ( i = 1;  i <= n; i++){
           x=ran0(&idum);
           fx=func(x);
           crude_mc += fx;
           sum_sigma += fx*fx;
      }
      crude_mc = crude_mc/((double) n );
      sum_sigma = sum_sigma/((double) n );
      variance=sum_sigma-crude_mc*crude_mc;

//     final output
      cout << " variance= " << variance << " Integral = "
           << crude_mc << " Exact= " << M_PI << endl;
}  // end of main program
// this function defines the function to integrate
double func(double x)
{
  double value;
```

[1]The Fortran 90/95 programs are not listed in the main text, they are found under the corresponding chapter as programs/chapter8/program*n*.f90.

Table 8.1: Results for $I = \int_0^1 dx 1/(1+x^2)$ as function of number of Monte Carlo samples N. The exact answer is $3.14159E+00$ for the integral and $4.13581E-01$ for the variance with six leading digits.

N	I	σ_N
10	3.10263E+00	3.98802E-01
100	3.02933E+00	4.04822E-01
1000	3.13395E+00	4.22881E-01
10000	3.14195E+00	4.11195E-01
100000	3.14003E+00	4.14114E-01
1000000	3.14213E+00	4.13838E-01
10000000	3.14177E+00	4.13523E-01
10^9	3.14162E+00	4.13581E-01

```
  value = 4/(1.+x*x);
  return value;
} // end of function to evaluate
```

We note that as N increases, the integral itself never reaches more than an agreement to the fourth or fifth digit. The variance also oscillates around its exact value $4.13581E-01$. Note well that the variance need not be zero but one can, with appropriate redefinitions of the integral be made smaller. A smaller variance yields also a smaller standard deviation. Improvements to this crude Monte Carlo approach will be discussed in the coming sections.

As an alternative, we could have used the random number generator provided by the C/C++ compiler through the functions *srand* and *rand*. In this case we initialise it via the function *srand*. The random number generator is called via the function *rand*, which returns an integer from 0 to its the maximum value, defined by the variable RAND_MAX as demonstrated in the next few lines of code.

```
  invers_period = 1./RAND_MAX;
  // initialise the random number generator
  srand(time(NULL));
  // obtain a floating number x in [0,1]
      x = double(rand())*invers_period;
```

8.1.2 Second illustration, particles in a box

We give here an example of how a system evolves towards a well defined equilibrium state.

Consider a box divided into two equal halves separated by a wall. At the beginning, time $t = 0$, there are N particles on the left side. A small hole in the wall is then opened and one particle can pass through the hole per unit time.

After some time the system reaches its equilibrium state with equally many particles in both halves, $N/2$. Instead of determining complicated initial conditions for a system of N particles, we model the system by a simple statistical model. In order to simulate this system, which may consist of $N \gg 1$ particles, we assume that all particles in the left half have equal probabilities of going to the right half. We introduce the label n_l to denote the number of particles at every time on the left side, and $n_r = N - n_l$ for those on the right side. The probability for a move to the right during a time step Δt is n_l/N. The algorithm for simulating this problem may then look like as follows

– Choose the number of particles N.

- Make a loop over time, where the maximum time should be larger than the number of particles N.
- For every time step Δt there is a probability n_l/N for a move to the right. Compare this probability with a random number x.
- If $x \leq n_l/N$, decrease the number of particles in the left half by one, i.e., $n_l = n_l - 1$. Else, move a particle from the right half to the left, i.e., $n_l = n_l + 1$.
- Increase the time by one unit (the external loop).

In this case, a Monte Carlo sample corresponds to one time unit Δt.

The following simple C/C++-program illustrates this model.

http://folk.uio.no/mhjensen/fys3150/2005/programs/chapter8/program2.cpp

```
// Particles in a box
#include <iostream>
#include <fstream>
#include <iomanip>
#include "lib.h"
using namespace  std;

ofstream ofile;
int main(int argc, char* argv[])
{
  char *outfilename;
  int initial_n_particles, max_time, time, random_n, nleft;
  long idum;
  // Read in output file, abort if there are too few command-line arguments
  if( argc <= 1 ){
    cout << "Bad Usage: " << argv[0] <<
      " read also output file on same line" << endl;
    exit(1);
  }
  else{
    outfilename=argv[1];
  }
  ofile.open(outfilename);
  // Read in data
  cout << "Initial number of particles = " << endl ;
  cin >> initial_n_particles;
  // setup of initial conditions
  nleft = initial_n_particles;
  max_time = 10*initial_n_particles;
  idum = -1;
  // sampling over number of particles
  for( time=0; time <= max_time; time++){
    random_n = ((int) initial_n_particles*ran0(&idum));
    if ( random_n <= nleft){
      nleft -= 1;
    }
    else{
      nleft += 1;
    }
```

```
    ofile << setiosflags(ios::showpoint | ios::uppercase);
    ofile << setw(15) << time;
    ofile << setw(15) << nleft << endl;
  }
  return 0;
} // end main function
```

The enclosed figure shows the development of this system as function of time steps. We note that for $N = 1000$ after roughly 2000 time steps, the system has reached the equilibrium state. There are however noteworthy fluctuations around equilibrium.

If we denote $\langle n_l \rangle$ as the number of particles in the left half as a time average after *equilibrium is reached*, we can define the standard deviation as

$$\sigma = \sqrt{\langle n_l^2 \rangle - \langle n_l \rangle^2}. \tag{8.19}$$

This problem has also an analytic solution to which we can compare our numerical simulation. If $n_l(t)$ are the number of particles in the left half after t moves, the change in $n_l(t)$ in the time interval Δt is

$$\Delta n = \left(\frac{N - n_l(t)}{N} - \frac{n_l(t)}{N} \right) \Delta t, \tag{8.20}$$

and assuming that n_l and t are continuous variables we arrive at

$$\frac{dn_l(t)}{dt} = 1 - \frac{2n_l(t)}{N}, \tag{8.21}$$

whose solution is

$$n_l(t) = \frac{N}{2} \left(1 + e^{-2t/N} \right), \tag{8.22}$$

with the initial condition $n_l(t = 0) = N$.

8.1.3 Radioactive decay

Radioactive decay is among one of the classical examples on use of Monte-Carlo simulations. Assume that a the time $t = 0$ we have $N(0)$ nuclei of type X which can decay radioactively. At a time $t > 0$ we are left with $N(t)$ nuclei. With a transition probability ω, which expresses the probability that the system will make a transition to another state during a time step of one second, we have the following first-order differential equation

$$dN(t) = -\omega N(t) dt, \tag{8.23}$$

whose solution is

$$N(t) = N(0) e^{-\omega t}, \tag{8.24}$$

where we have defined the mean lifetime τ of X as

$$\tau = \frac{1}{\omega}. \tag{8.25}$$

If a nucleus X decays to a daugther nucleus Y which also can decay, we get the following coupled equations

$$\frac{dN_X(t)}{dt} = -\omega_X N_X(t), \tag{8.26}$$

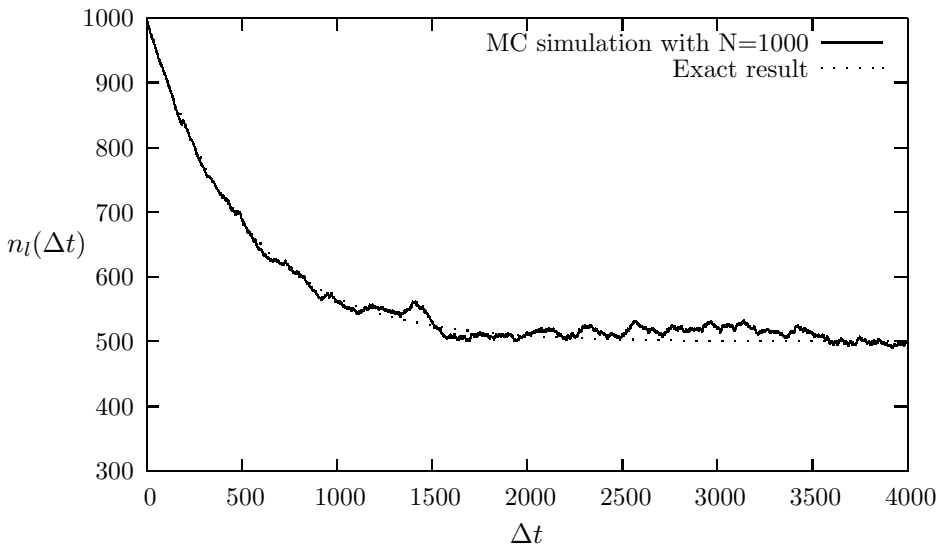

Figure 8.1: Number of particles in the left half of the container as function of the number of time steps. The solution is compared with the analytic expression. $N = 1000$.

and

$$\frac{dN_Y(t)}{dt} = -\omega_Y N_Y(t) + \omega_X N_X(t). \tag{8.27}$$

The program example in the next subsection illustrates how we can simulate such the decay process of one type of nuclei through a Monte Carlo sampling procedure.

8.1.4 Program example for radioactive decay of one type of nucleus

The program is split in four tasks, a main program with various declarations,

http://folk.uio.no/mhjensen/fys3150/2005/programs/chapter8/program3.cpp

```
// Radioactive decay of nuclei
#include <iostream>
#include <fstream>
#include <iomanip>
#include "lib.h"
using namespace  std;
ofstream ofile;
// Function to read in data from screen
void initialise(int&, int&, int&, double& ) ;
// The Mc sampling for nuclear decay
void  mc_sampling(int, int, int, double, int*);
// prints to screen the results of the calculations
void  output(int, int, int *);
int main(int argc, char* argv[])
{
  char *outfilename;
  int initial_n_particles, max_time, number_cycles;
```

```
  double decay_probability;
  int *ncumulative;
  // Read in output file, abort if there are too few command-line arguments
  if( argc <= 1 ){
    cout << "Bad Usage: " << argv[0] <<
      " read also output file on same line" << endl;
    exit(1);
  }
  else{
    outfilename=argv[1];
  }
  ofile.open(outfilename);
  // Read in data
  initialise(initial_n_particles, max_time, number_cycles,
             decay_probability) ;
  ncumulative = new int [max_time+1];
  // Do the mc sampling
  mc_sampling(initial_n_particles, max_time, number_cycles,
              decay_probability, ncumulative);
  // Print out results
  output(max_time, number_cycles, ncumulative);
  delete [] ncumulative;
  return 0;
} // end of main function
```

followed by a part which performs the Monte Carlo sampling

```
void mc_sampling(int initial_n_particles, int max_time,
                 int number_cycles, double decay_probability,
                 int *ncumulative)
{
  int cycles, time, np, n_unstable, particle_limit;
  long idum;

  idum=-1;  // initialise random number generator
  // loop over monte carlo cycles
  // One monte carlo loop is one sample
  for (cycles = 1; cycles <= number_cycles; cycles++){
    n_unstable = initial_n_particles;
    //  accumulate the number of particles per time step per trial
    ncumulative[0] += initial_n_particles;
    // loop over each time step
    for (time=1; time <= max_time; time++){
      // for each time step, we check each particle
      particle_limit = n_unstable;
      for ( np = 1; np <=  particle_limit; np++) {
        if( ran0(&idum) <= decay_probability) {
          n_unstable=n_unstable-1;
        }
      }  // end of loop over particles
      ncumulative[time] += n_unstable;
    }  // end of loop over time steps
  }    // end of loop over MC trials
```

```
}    // end mc_sampling function
```

and finally functions for reading input and writing output data. The latter are not listed here, see under program/chapter8/program3.cpp for a full listing. The input variables are the number of Monte Carlo cycles, the maximum number of time steps, the initial number of particles and the decay probability. The output consists of the number of remaining nuclei at each time step.

8.1.5 Brief summary

In essence the Monte Carlo method contains the following ingredients

- A PDF which characterizes the system
- Random numbers which are generated so as to cover in a as uniform as possible way on the unity interval [0,1].
- A sampling rule
- An error estimation
- Techniques for improving the errors

In the next section we discuss various PDF's which may be of relevance here, thereafter we discuss how to compute random numbers. Section 8.4 discusses Monte Carlo integration in general, how to choose the correct weighting function and how to evaluate integrals with dimensions $d > 1$.

8.2 Probability distribution functions

Hitherto, we have tacitly used properties of probability distribution functions in our computation of expectation values. Here and there we have referred to the uniform PDF. It is now time to present some general features of PDFs which we may encounter when doing physics and how we define various expectation values. In addition, we derive the central limit theorem and discuss its meaning in the light of properties of various PDFs.

The following table collects properties of probability distribution functions. In our notation we reserve the label $p(x)$ for the probability of a certain event, while $P(x)$ is the cumulative probability.

Table 8.2: Important properties of PDFs.

	Discrete PDF	Continuous PDF
Domain	$\{x_1, x_2, x_3, \ldots, x_N\}$	$[a, b]$
Probability	$p(x_i)$	$p(x)dx$
Cumulative	$P_i = \sum_{l=1}^{i} p(x_l)$	$P(x) = \int_a^x p(t)dt$
Positivity	$0 \leq p(x_i) \leq 1$	$p(x) \geq 0$
Positivity	$0 \leq P_i \leq 1$	$0 \leq P(x) \leq 1$
Monotonic	$P_i \geq P_j$ if $x_i \geq x_j$	$P(x_i) \geq P(x_j)$ if $x_i \geq x_j$
Normalization	$P_N = 1$	$P(b) = 1$

With a PDF we can compute expectation values of selected quantities such as

$$\langle x^k \rangle = \frac{1}{N}\sum_{i=1}^{N} x_i^k p(x_i), \tag{8.28}$$

if we have a discrete PDF or

$$\langle x^k \rangle = \int_a^b x^k p(x) dx, \tag{8.29}$$

in the case of a continuous PDF. We have already defined the mean value μ and the variance σ^2.

The expectation value of a quantity $f(x)$ is then given by for example

$$\langle f \rangle = \int_a^b f(x)p(x)dx. \tag{8.30}$$

We have already seen the use of the last equation when we applied the crude Monte Carlo approach to the evaluation of an integral.

There are at least three PDFs which one may encounter. These are the

1. uniform distribution
$$p(x) = \frac{1}{b-a}\Theta(x-a)\Theta(b-x), \tag{8.31}$$
yielding probabilities different from zero in the interval $[a, b]$. The mean value and the variance for this distribution are discussed in section 8.3.

2. The exponential distribution
$$p(x) = \alpha e^{-\alpha x}, \tag{8.32}$$
yielding probabilities different from zero in the interval $[0, \infty)$ and with mean value
$$\mu = \int_0^\infty xp(x)dx = \int_0^\infty x\alpha e^{-\alpha x}dx = \frac{1}{\alpha} \tag{8.33}$$
and variance
$$\sigma^2 = \int_0^\infty x^2 p(x)dx - \mu^2 = \frac{1}{\alpha^2}. \tag{8.34}$$

3. Finally, we have the so-called univariate normal distribution, or just the normal distribution
$$p(x) = \frac{1}{b\sqrt{2\pi}}\exp\left(-\frac{(x-a)^2}{2b^2}\right) \tag{8.35}$$
with probabilities different from zero in the interval $(-\infty, \infty)$. The integral $\int_{-\infty}^{\infty}\exp\left(-(x^2\right)dx$ appears in many calculations, its value is $\sqrt{\pi}$, a result we will need when we compute the mean value and the variance. The mean value is
$$\mu = \int_0^\infty xp(x)dx = \frac{1}{b\sqrt{2\pi}}\int_{-\infty}^{\infty} x\exp\left(-\frac{(x-a)^2}{2b^2}\right)dx, \tag{8.36}$$
which becomes with a suitable change of variables
$$\mu = \frac{1}{b\sqrt{2\pi}}\int_{-\infty}^{\infty} b\sqrt{2}(a+b\sqrt{2}y)\exp -y^2 dy = a. \tag{8.37}$$

Similarly, the variance becomes

$$\sigma^2 = \frac{1}{b\sqrt{2\pi}} \int_{-\infty}^{\infty} (x-\mu)^2 \exp\left(-\frac{(x-a)^2}{2b^2}\right) dx, \tag{8.38}$$

and inserting the mean value and performing a variable change we obtain

$$\sigma^2 = \frac{1}{b\sqrt{2\pi}} \int_{-\infty}^{\infty} b\sqrt{2}(b\sqrt{2}y)^2 \exp\left(-y^2\right) dy = \frac{2b^2}{\sqrt{\pi}} \int_{-\infty}^{\infty} y^2 \exp\left(-y^2\right) dy, \tag{8.39}$$

and performing a final integration by parts we obtain the well-known result $\sigma^2 = b^2$. It is useful to introduce the standard normal distribution as well, defined by $\mu = a = 0$, viz. a distribution centered around zero and with a variance $\sigma^2 = 1$, leading to

$$p(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right). \tag{8.40}$$

The exponential and uniform distributions have simple cumulative functions, whereas the normal distribution does not, being proportional to the so-called error function $erf(x)$, given by

$$P(x) = .\frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} \exp\left(-\frac{t^2}{2}\right) dt, \tag{8.41}$$

which is difficult to evaluate in a quick way. Later in this chapter we will present an algorithm by Box and Mueller which allows us to compute the cumulative distribution using random variables sampled from the uniform distribution.

Some other PDFs which one encounters often in the natural sciences are the binomial distribution

$$p(x) = \left(\begin{array}{c} n \\ x \end{array} \right) y^x (1-y)^{n-x} \; x = 0, 1, \ldots, n, \tag{8.42}$$

where y is the probability for a specific event, such as the tossing of a coin or moving left or right in case of a random walker. Note that x is a discrete stochastic variable.

The sequence of binomial trials is characterized by the following definitions

- Every experiment is thought to consist of N independent trials.
- In every independent trial one registers if a specific situation happens or not, such as the jump to the left or right of a random walker.
- The probability for every outcome in a single trial has the same value, for example the outcome of tossing a coin is always $1/2$.

In the next chapter we will show that the probability distribution for a random walker approaches the binomial distribution.

In order to compute the mean and variance we need to recall Newton's binomial formula

$$(a+b)^m = \sum_{n=0}^{m} \left(\begin{array}{c} m \\ n \end{array} \right) a^n b^{m-n},$$

which can be used to show that

$$\sum_{x=0}^{n} \left(\begin{array}{c} n \\ x \end{array} \right) y^x (1-y)^{n-x} = (y+1-y)^n = 1, \tag{8.43}$$

the PDF is normalized to one. The mean value is

$$\mu = \sum_{x=0}^{n} x \left(\begin{array}{c} n \\ x \end{array} \right) y^x (1-y)^{n-x} = \sum_{x=0}^{n} x \frac{n!}{x!(n-x)!} y^x (1-y)^{n-x}, \tag{8.44}$$

resulting in

$$\mu = \sum_{x=0}^{n} x \frac{(n-1)!}{(x-1)!(n-1-(x-1))!} y^{x-1} (1-y)^{n-1-(x-1)}, \tag{8.45}$$

which we rewrite as

$$\mu = ny \sum_{\nu=0}^{n} \left(\begin{array}{c} n-1 \\ \nu \end{array} \right) y^{\nu} (1-y)^{n-1-\nu} = ny(y+1-y)^{n-1} = ny. \tag{8.46}$$

The variance is slightly trickier to get, see the next exercises. It reads $\sigma^2 = ny(1-y)$.

Another important distribution with discrete stochastic variables x is the Poisson model, which resembles the exponential distribution and reads

$$p(x) = \frac{\lambda^x}{x!} e^{-\lambda} \; x = 0, 1, \ldots, ; \lambda > 0. \tag{8.47}$$

In this case both the mean value and the variance are easier to calculate,

$$\mu = \sum_{x=0}^{\infty} x \frac{\lambda^x}{x!} e^{-\lambda} = \lambda e^{-\lambda} \sum_{x=1}^{\infty} \frac{\lambda^{x-1}}{(x-1)!} = \lambda, \tag{8.48}$$

and the variance is $\sigma^2 = \lambda$. Example of applications of the Poisson distribution is the counting of the number of α-particles emitted from a radioactive source in a given time interval. In the limit of $n \to \infty$ and for small probabilities y, the binomial distribution approaches the Poisson distribution. Setting $\lambda = ny$, with y the probability for an event in the binomial distribution we can show that

$$\lim_{n\to\infty} \left(\begin{array}{c} n \\ x \end{array} \right) y^x (1-y)^{n-x} e^{-\lambda} \sum_{x=1}^{\infty} = \frac{\lambda^x}{x!} e^{-\lambda}, \tag{8.49}$$

see for example Refs. [43, 44] for a proof.

Exercise 8.1
Calculate the cumulative functions $P(x)$ for the binomial and the Poisson distributions and their variances.

8.2.1 Multivariable Expectation Values

Let us recapitulate some of the above concepts using a discrete PDF (which is what we end up doing anyway on a computer). The mean value of a random variable X with range $x_1, x_2, \ldots, N$ is

$$\langle x \rangle = \mu = \frac{1}{N} \sum_{i=1}^{N} x_i p(x_i),$$

and the variance is

$$\langle \sigma^2 \rangle = \frac{1}{N}\sum_{i=1}^{N}(x_i - \langle x \rangle)^2 p(x_i) = \frac{1}{N}\sum_{i=1}^{N}\langle (x_i - \mu_i)^2 \rangle.$$

Assume now that we have two independent sets of measurements X_1 and X_2 with corresponding mean and variance μ_1 and μ_2 and $\langle \sigma^2 \rangle_{X_1}$ and $\langle \sigma^2 \rangle_{X_2}$. It follows that if we define the new stochastic variable

$$Y = X_1 + X_2, \tag{8.50}$$

we have

$$\mu_Y = \mu_1 + \mu_2, \tag{8.51}$$

and

$$\langle \sigma^2 \rangle_Y = \langle (Y - \mu_Y)^2 \rangle = \langle (X_1 - \mu_1)^2 \rangle + \langle (X_2 - \mu_2)^2 \rangle + 2\langle (X_1 - \mu_1)(X_2 - \mu_2) \rangle. \tag{8.52}$$

It is useful to define the so-called covariance, given by

$$\langle cov(X_1, X_2) \rangle = \langle (X_1 - \mu_1)(X_2 - \mu_2) \rangle \tag{8.53}$$

where we consider the averages μ_1 and μ_2 as the outcome of two separate measurements. The covariance measures thus the degree of correlation between variables. We can then rewrite the variance of Y as

$$\langle \sigma^2 \rangle_Y = \sum_{j=1}^{2}\langle (X_j - \mu_j)^2 \rangle + 2cov(X_1, X_2), \tag{8.54}$$

which in our notation becomes

$$\langle \sigma^2 \rangle_Y = \langle \sigma^2 \rangle_{X_1} + \langle \sigma^2 \rangle_{X_2} + 2cov(X_1, X_2). \tag{8.55}$$

If X_1 and X_2 are two independent variables we can show that the covariance is zero, but one cannot deduce from a zero covariance whether the variables are independent or not. If our random variables which we generate are truely random numbers, then the covariance should be zero. We will see tests of standard random number generators in the next section. A way to measure the correlation between two sets of stochastic variables is the so-called correlation function $\rho(X_1, X_2)$ defined as

$$\rho(X_1, X_2) = \frac{\langle cov(X_1, X_2) \rangle}{\sqrt{\langle \sigma^2 \rangle_{X_1} \langle \sigma^2 \rangle_{X_2}}}. \tag{8.56}$$

Obviously, if the covariance is zero due to the fact that the variables are independent, then the correlation is zero. This quantity is often called the correlation coefficient between X_1 and X_2. We can extend this analysis to a set of stochastic variables $Y = (X_1 + X_2 + \cdots + X_N)$. We now assume that we have N different measurements of the mean and variance of a given variable. Each measurement consists again of N measurements, although we could have chosen the latter to be different from N. As an example, every evening for N days you measure N throws of two dice. The mean and variance are defined as above. The total mean value is defined as

$$\langle \mu_Y \rangle = \sum_{i=1}^{N}\langle \mu_i \rangle. \tag{8.57}$$

The total variance is however now defined as

$$\langle\sigma^2\rangle_Y = \langle(Y-\mu_Y)^2\rangle = \sum_{j=1}^{N}\langle(X_j-\mu_j)\rangle^2 = \sum_{j=1}^{N}\langle\sigma^2\rangle_{X_j} + 2\sum_{j<k}^{N}\langle(X_j-\mu_j)\rangle\langle(X_k-\mu_k)\rangle, \quad (8.58)$$

or

$$\langle\sigma^2\rangle_Y = \sum_{j=1}^{N}\langle\sigma^2\rangle_{X_j} + 2\sum_{j<k}^{N} cov(X_j,X_k). \quad (8.59)$$

If the variables are independent, the covariance is zero and the variance is reduced to

$$\langle\sigma^2\rangle_Y = \sum_{j=1}^{N}\langle\sigma^2\rangle_{X_j}, \quad (8.60)$$

and if we assume that all sets of measurements produce the same variance $\langle\sigma^2\rangle$, we end up with

$$\langle\sigma^2\rangle_Y = N\langle\sigma^2\rangle. \quad (8.61)$$

In the next subsection we combine these results with the central limit theorem in order to obtain the classical expression for the standard deviation.

8.2.2 The central limit theorem

Suppose we have a PDF $p(x)$ from which we generate a series N of averages $\langle x_i\rangle$. Each mean value $\langle x_i\rangle$ is viewed as the average of a specific measurement, e.g., throwing dice 100 times and then taking the average value, or producing a certain amount of random numbers. For notational ease, we set $\langle x_i\rangle = x_i$ in the discussion which follows.

If we compute the mean z of N such mean values x_i

$$z = \frac{x_1+x_2+\cdots+x_N}{N}, \quad (8.62)$$

the question we pose is which is the PDF of the new variable z.

The probability of obtaining an average value z is the product of the probabilities of obtaining arbitrary individual mean values x_i, but with the constraint that the average is z. We can express this through the following expression

$$\tilde{p}(z) = \int dx_1 p(x_1)\int dx_2 p(x_2)\ldots\int dx_N p(x_N)\delta(z-\frac{x_1+x_2+\cdots+x_N}{N}), \quad (8.63)$$

where the δ-function enbodies the constraint that the mean is z. All measurements that lead to each individual x_i are expected to be independent, which in turn means that we can express $\tilde{p}$ as the product of individual $p(x_i)$.

If we use the integral expression for the δ-function

$$\delta(z-\frac{x_1+x_2+\cdots+x_N}{N}) = \frac{1}{2\pi}\int_{-\infty}^{\infty} dq e^{\left(iq(z-\frac{x_1+x_2+\cdots+x_N}{N})\right)}, \quad (8.64)$$

and inserting $e^{i\mu q-i\mu q}$ where μ is the mean value we arrive at

$$\tilde{p}(z) = \frac{1}{2\pi}\int_{-\infty}^{\infty} dq e^{(iq(z-\mu))}\left[\int_{-\infty}^{\infty} dx p(x) e^{(iq(\mu-x)/N)}\right]^N, \quad (8.65)$$

with the integral over x resulting in

$$\int_{-\infty}^{\infty} dxp(x)\exp\left(iq(\mu-x)/N\right) = \int_{-\infty}^{\infty} dxp(x)\left[1+\frac{iq(\mu-x)}{N}-\frac{q^2(\mu-x)^2}{2N^2}+\ldots\right]. \tag{8.66}$$

The second term on the rhs disappears since this is just the mean and employing the definition of σ^2 we have

$$\int_{-\infty}^{\infty} dxp(x)e^{(iq(\mu-x)/N)} = 1-\frac{q^2\sigma^2}{2N^2}+\ldots, \tag{8.67}$$

resulting in

$$\left[\int_{-\infty}^{\infty} dxp(x)\exp\left(iq(\mu-x)/N\right)\right]^N \approx \left[1-\frac{q^2\sigma^2}{2N^2}+\ldots\right]^N, \tag{8.68}$$

and in the limit $N\rightarrow\infty$ we obtain

$$\tilde{p}(z) = \frac{1}{\sqrt{2\pi}(\sigma/\sqrt{N})}\exp\left(-\frac{(z-\mu)^2}{2(\sigma/\sqrt{N})^2}\right), \tag{8.69}$$

which is the normal distribution with variance $\sigma_N^2=\sigma^2/N$, where σ is the variance of the PDF $p(x)$ and μ is also the mean of the PDF $p(x)$.

Thus, the central limit theorem states that the PDF $\tilde{p}(z)$ of the average of N random values corresponding to a PDF $p(x)$ is a normal distribution whose mean is the mean value of the PDF $p(x)$ and whose variance is the variance of the PDF $p(x)$ divided by N, the number of values used to compute z.

The theorem is satisfied by a large class of PDFs. Note however that for a finite N, it is not always possible to find a closed expression for $\tilde{p}(x)$. The central limit theorem leads then to the well-known expression for the standard deviation, given by

$$\sigma_N = \frac{\sigma}{\sqrt{N}}. \tag{8.70}$$

The latter is true only if the average value is known exactly. This is obtained in the limit $N\rightarrow\infty$ only. Because the mean and the variance are measured quantities we obtain the familiar expression in statistics

$$\sigma_N \approx \frac{\sigma}{\sqrt{N-1}}. \tag{8.71}$$

8.3 Random numbers

Uniform deviates are just random numbers that lie within a specified range (typically 0 to 1), with any one number in the range just as likely as any other. They are, in other words, what you probably think random numbers are. However, we want to distinguish uniform deviates from other sorts of random numbers, for example numbers drawn from a normal (Gaussian) distribution of specified mean and standard deviation. These other sorts of deviates are almost always generated by performing appropriate operations on one or more uniform deviates, as we will see in subsequent sections. So, a reliable source of random uniform deviates, the subject of this section, is an essential building block for any sort of stochastic modeling or Monte Carlo computer work. A disclaimer is however appropriate. It should be fairly obvious that something as deterministic as a computer cannot generate purely random numbers.

Numbers generated by any of the standard algorithm are in reality pseudo random numbers, hopefully abiding to the following criteria:

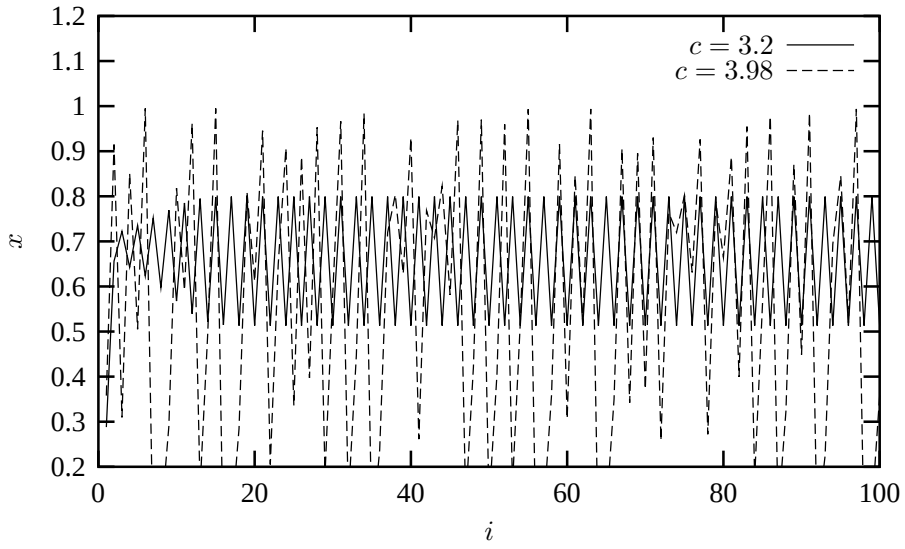

Figure 8.2: Plot of the logistic mapping $x_{i+1} = cx_i(1 - x_i)$ for $x_0 = 0.1$ and $c = 3.2$ and $c = 3.98$.

1. they produce a uniform distribution in the interval [0,1].

2. correlations between random numbers are negligible

3. the period before the same sequence of random numbers is repeated is as large as possible and finally

4. the algorithm should be fast.

That correlations, see below for more details, should be as small as possible resides in the fact that every event should be independent of the other ones. As an example, a particular simple system that exhibits a seemingly random behavior can be obtained from the iterative process

$$x_{i+1} = cx_i(1 - x_i), \tag{8.72}$$

which is often used as an example of a chaotic system. c is constant and for certain values of c and x_0 the system can settle down quickly into a regular periodic sequence of values $x_1, x_2, x_3, \ldots$. For $x_0 = 0.1$ and $c = 3.2$ we obtain a periodic pattern as shown in Fig. 8.2. Changing c to $c = 3.98$ yields a sequence which does not converge to any specific pattern. The values of x_i seem purely random. Although the latter choice of c yields a seemingly random sequence of values, the various values of x harbor subtle correlations that a truly random number sequence would not possess.

The most common random number generators are based on so-called Linear congruential relations of the type

$$N_i = (aN_{i-1} + c)\mathrm{MOD}(M), \tag{8.73}$$

which yield a number in the interval [0,1] through

$$x_i = N_i/M \tag{8.74}$$

The number M is called the period and it should be as large as possible and N_0 is the starting value, or seed. The function MOD means the remainder, that is if we were to evaluate $(13)\mathrm{MOD}(9)$, the outcome is the remainder of the division $13/9$, namely 4.

The problem with such generators is that their outputs are periodic; they will start to repeat themselves with a period that is at most M. If however the parameters a and c are badly chosen, the period may be even shorter.

Consider the following example

$$N_i = (6N_{i-1} + 7)\mathrm{MOD}(5), \tag{8.75}$$

with a seed $N_0 = 2$. This generator produces the sequence $4, 1, 3, 0, 2, 4, 1, 3, 0, 2, \ldots\ldots$, i.e., a sequence with period 5. However, increasing M may not guarantee a larger period as the following example shows

$$N_i = (27N_{i-1} + 11)\mathrm{MOD}(54), \tag{8.76}$$

which still, with $N_0 = 2$, results in $11, 38, 11, 38, 11, 38, \ldots$, a period of just 2.

Typical periods for the random generators provided in the program library are of the order of $\sim 10^9$ or larger. Other random number generators which have become increasingly popular are so-called shift-register generators. In these generators each successive number depends on many preceding values (rather than the last values as in the linear congruential generator). For example, you could make a shift register generator whose lth number is the sum of the $l - i$th and $l - j$th values with modulo M,

$$N_l = (aN_{l-i} + cN_{l-j})\mathrm{MOD}(M). \tag{8.77}$$

Such a generator again produces a sequence of pseudorandom numbers but this time with a period much larger than M. It is also possible to construct more elaborate algorithms by including more than two past terms in the sum of each iteration. One example is the generator of Marsaglia and Zaman [46] which consists of two congruential relations

$$N_l = (N_{l-3} - N_{l-1})\mathrm{MOD}(2^{31} - 69), \tag{8.78}$$

followed by

$$N_l = (69069N_{l-1} + 1013904243)\mathrm{MOD}(2^{32}), \tag{8.79}$$

which according to the authors has a period larger than 2^{94}.

Moreover, rather than using modular addition, we could use the bitwise exclusive-OR ($\oplus$) operation so that

$$N_l = (N_{l-i}) \oplus (N_{l-j}) \tag{8.80}$$

where the bitwise action of $\oplus$ means that if $N_{l-i} = N_{l-j}$ the result is 0 whereas if $N_{l-i} \neq N_{l-j}$ the result is 1. As an example, consider the case where $N_{l-i} = 6$ and $N_{l-j} = 11$. The first one has a bit representation (using 4 bits only) which reads 0110 whereas the second number is 1011. Employing the $\oplus$ operator yields 1101, or $2^3 + 2^2 + 2^0 = 13$.

In Fortran90, the bitwise $\oplus$ operation is coded through the intrinsic function $\mathrm{IEOR}(m, n)$ where m and n are the input numbers, while in C it is given by $m \wedge n$. The program below (from Numerical Recipes, chapter 7.1) shows the function $ran0$ implements

$$N_i = (aN_{i-1})\mathrm{MOD}(M). \tag{8.81}$$

However, since a and N_{i-1} are integers and their multiplication could become greater than the standard 32 bit integer, there is a trick via Schrage's algorithm which approximates the multiplication of large integers through the factorization

$$M = aq + r,$$

where we have defined

$$q = [M/a],$$

and

$$r = M \text{ MOD } a.$$

where the brackets denote integer division. In the code below the numbers q and r are chosen so that $r < q$. To see how this works we note first that

$$(aN_{i-1})\text{MOD}(M) = (aN_{i-1} - [N_{i-1}/q]M)\text{MOD}(M), \tag{8.82}$$

since we can add or subtract any integer multiple of M from aN_{i-1}. The last term $[N_{i-1}/q]M\text{MOD}(M)$ is zero since the integer division $[N_{i-1}/q]$ just yields a constant which is multiplied with M. We can now rewrite Eq. (8.82) as

$$(aN_{i-1})\text{MOD}(M) = (aN_{i-1} - [N_{i-1}/q](aq + r))\text{MOD}(M), \tag{8.83}$$

which results in

$$(aN_{i-1})\text{MOD}(M) = (a(N_{i-1} - [N_{i-1}/q]q) - [N_{i-1}/q]r))\,\text{MOD}(M), \tag{8.84}$$

yielding

$$(aN_{i-1})\text{MOD}(M) = (a(N_{i-1}\text{MOD}(q)) - [N_{i-1}/q]r))\,\text{MOD}(M). \tag{8.85}$$

The term $[N_{i-1}/q]r$ is always smaller or equal $N_{i-1}(r/q)$ and with $r < q$ we obtain always a number smaller than N_{i-1}, which is smaller than M. And since the number $N_{i-1}\text{MOD}(q)$ is between zero and $q - 1$ then $a(N_{i-1}\text{MOD}(q)) < aq$. Combined with our definition of $q = [M/a]$ ensures that this term is also smaller than M meaning that both terms fit into a 32-bit signed integer. None of these two terms can be negative, but their difference could. The algorithm below adds M if their difference is negative. Note that the program uses the bitwise $\oplus$ operator to generate the starting point for each generation of a random number. The period of $ran0$ is $\sim 2.1 \times 10^9$. A special feature of this algorithm is that is should never be called with the initial seed set to 0.

```
    /*
     ** The function
     **            ran0()
     ** is an "Minimal" random number generator of Park and Miller
     ** (see Numerical recipe page 279). Set or reset the input value
     ** idum to any integer value (except the unlikely value MASK)
     ** to initialize the sequence; idum must not be altered between
     ** calls for sucessive deviates in a sequence.
     ** The function returns a uniform deviate between 0.0 and 1.0.
     */
double ran0(long &idum)
{
   const int a = 16807, m = 2147483647, q = 127773;
   const int r = 2836, MASK = 123459876;
   const double am = 1./m;
```

```
    long      k;
    double    ans;
    idum ^= MASK;
    k = (*idum)/q;
    idum = a*(idum - k*q) - r*k;
    // add m if negative difference
    if(idum < 0) idum += m;
    ans=am*(idum);
    idum ^= MASK;
    return ans;
} // End: function ran0()
```

The other random number generators $ran1$, $ran2$ and $ran3$ are described in detail in chapter 7.1 of Numerical Recipes. Here we limit ourselves to study selected properties of these generators.

8.3.1 Properties of selected random number generators

As mentioned previously, the underlying PDF for the generation of random numbers is the uniform distribution, meaning that the probability for finding a number x in the interval [0,1] is $p(x) = 1$.

A random number generator should produce numbers which uniformly distributed in this interval. Table 8.3 shows the distribution of $N = 10000$ random numbers generated by the functions in the program library. We note in this table that the number of points in the various intervals $0.0 - 0.1$, $0.1 - 0.2$ etc are fairly close to 1000, with some minor deviations.

Two additional measures are the standard deviation σ and the mean $\mu = \langle x \rangle$.

For the uniform distribution with N points we have that the average $\langle x^k \rangle$ is

$$\langle x^k \rangle = \frac{1}{N}\sum_{i=1}^{N} x_i^k p(x_i), \tag{8.86}$$

and taking the limit $N \rightarrow \infty$ we have

$$\langle x^k \rangle = \int_0^1 dx p(x) x^k = \int_0^1 dx x^k = \frac{1}{k+1}, \tag{8.87}$$

since $p(x) = 1$. The mean value μ is then

$$\mu = \langle x \rangle = \frac{1}{2} \tag{8.88}$$

while the standard deviation is

$$\sigma = \sqrt{\langle x^2 \rangle - \mu^2} = \frac{1}{\sqrt{12}} = 0.2886. \tag{8.89}$$

The various random number generators produce results which agree rather well with these limiting values. In the next section, in our discussion of probability distribution functions and the central limit theorem, we are to going to see that the uniform distribution evolves towards a normal distribution in the limit $N \rightarrow \infty$.

There are many other tests which can be performed. Often a picture of the numbers generated may reveal possible patterns.

Table 8.3: Number of x-values for various intervals generated by 4 random number generators, their corresponding mean values and standard deviations. All calculations have been initialized with the variable $idum = -1$.

x-bin	ran0	ran1	ran2	ran3
0.0-0.1	1013	991	938	1047
0.1-0.2	1002	1009	1040	1030
0.2-0.3	989	999	1030	993
0.3-0.4	939	960	1023	937
0.4-0.5	1038	1001	1002	992
0.5-0.6	1037	1047	1009	1009
0.6-0.7	1005	989	1003	989
0.7-0.8	986	962	985	954
0.8-0.9	1000	1027	1009	1023
0.9-1.0	991	1015	961	1026
μ	0.4997	0.5018	0.4992	0.4990
σ	0.2882	0.2892	0.2861	0.2915

Since our random numbers, which are typically generated via a linear congruential algorithm, are never fully independent, we can then define an important test which measures the degree of correlation, namely the so-called auto-correlation function C_k

$$C_k = \frac{\langle x_{i+k}x_i\rangle - \langle x_i\rangle^2}{\langle x_i^2\rangle - \langle x_i\rangle^2}, \tag{8.90}$$

with $C_0 = 1$. Recall that $\sigma^2 = \langle x_i^2\rangle - \langle x_i\rangle^2$. The non-vanishing of C_k for $k \neq 0$ means that the random numbers are not independent. The independence of the random numbers is crucial in the evaluation of other expectation values. If they are not independent, our assumption for approximating σ_N in Eq. (8.14) is no longer valid.

The expectation values which enter the definition of C_k are given by

$$\langle x_{i+k}x_i\rangle = \frac{1}{N-k}\sum_{i=1}^{N-k} x_i x_{i+k}. \tag{8.91}$$

Fig. 8.3 compares the auto-correlation function calculated from $ran0$ and $ran1$. As can be seen, the correlations are non-zero, but small. The fact that correlations are present is expected, since all random numbers do depend in some way on the previous numbers.

Exercise 8.2
Make a program which computes random numbers according to the algorithm of Marsaglia and Zaman, Eqs. (8.78) and (8.79). Compute the correlation function C_k and compare with the auto-correlation function from the function $ran0$.

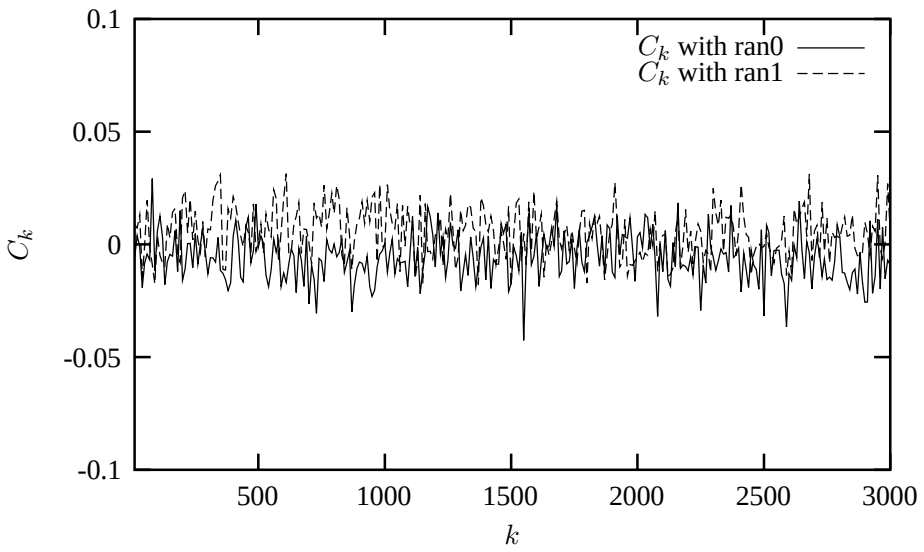

Figure 8.3: Plot of the auto-correlation function C_k for various k-values for $N = 10000$ using the random number generators $ran0$ and $ran1$.

8.4 Improved Monte Carlo integration

In section 8.1 we presented a simple brute force approach to integration with the Monte Carlo method. There we sampled over a given number of points distributed uniformly in the interval $[0, 1]$

$$I = \int_0^1 f(x)dx \approx \sum_{i=1}^{N} \omega_i f(x_i) = \frac{1}{N}\sum_{i=1}^{N} f(x_i) = \langle f \rangle,$$

with the weights $\omega_i = 1$.

Here we introduce two important topics which in most cases improve upon the above simple brute force approach with the uniform distribution $p(x) = 1$ for $x \in [0, 1]$. With improvements we think of a smaller variance and the need for fewer Monte Carlo samples, although each new Monte Carlo sample will most likely be more times consuming than corresponding ones of the brute force method.

- The first topic deals with change of variables, and is linked to the cumulative function $P(x)$ of a PDF $p(x)$. Obviously, not all integration limits go from $x = 0$ to $x = 1$, rather, in physics we are often confronted with integration domains like $x \in [0, \infty)$ or $x \in (-\infty, \infty)$ etc. Since all random number generators give numbers in the interval $x \in [0, 1]$, *we need a mapping from this integration interval to the explicit one under consideration.*

- The next topic deals with the shape of the integrand itself. Let us for the sake of simplicity just assume that the integration domain is again from $x = 0$ to $x = 1$. If the function to be integrated $f(x)$ has sharp peaks and is zero or small for many values of $x \in [0, 1]$, most samples of $f(x)$ give contributions to the integral I which are negligible. As a consequence we need many N samples to have a sufficient accuracy in the region where $f(x)$ is peaked. What do we do then? We try to find a new PDF $p(x)$ chosen so as to match $f(x)$ in order to render the integrand smooth. The new PDF

$p(x)$ has in turn an x domain which most likely has to be mapped from the domain of the uniform distribution.

Why care at all and not be content with just a change of variables in cases where that is needed? Below we show several examples of how to improve a Monte Carlo integration through smarter choices of PDFs which render the integrand smoother. However one classic example from quantum mechanics illustrates the need for a good sampling function.

In quantum mechanics, the probability distribution function is given by $p(x) = \Psi(x)^*\Psi(x)$, where $\Psi(x)$ is the eigenfunction arising from the solution of e.g., the time-independent Schrödinger equation. If $\Psi(x)$ is an eigenfunction, the corresponding energy eigenvalue is given by

$$H(x)\Psi(x) = E\Psi(x), \tag{8.92}$$

where $H(x)$ is the hamiltonian under consideration. The expectation value of H, assuming that the quantum mechanical PDF is normalized, is given by

$$\langle H\rangle = \int dx\Psi(x)^*H(x)\Psi(x). \tag{8.93}$$

We could insert $\Psi(x)/\Psi(x)$ right to the left of H and rewrite the last equation as

$$\langle H\rangle = \int dx\Psi(x)^*\Psi(x)\frac{H(x)}{\Psi(x)}\Psi(x), \tag{8.94}$$

or

$$\langle H\rangle = \int dxp(x)\tilde{H}(x), \tag{8.95}$$

which is on the form of an expectation value with

$$\tilde{H}(x) = \frac{H(x)}{\Psi(x)}\Psi(x). \tag{8.96}$$

The crucial point to note is that if $\Psi(x)$ is the exact eigenfunction itself with eigenvalue E, then $\tilde{H}(x)$ reduces just to the constant E and we have

$$\langle H\rangle = \int dxp(x)E = E, \tag{8.97}$$

since $p(x)$ is normalized.

However, *in most cases of interest we do not have the exact* Ψ. But if we have made a clever choice for $\Psi(x)$, the expression $\tilde{H}(x)$ exhibits a smooth behavior in the neighbourhood of the exact solution. The above example encompasses the main essence of the Monte Carlo philosophy. It is a trial approach, where intelligent guesses lead to hopefully better results.

8.4.1 Change of variables

The starting point is always the uniform distribution

$$p(x)dx = \begin{cases} dx & 0 \leq x \leq 1 \\ 0 & else \end{cases} \tag{8.98}$$

with $p(x) = 1$ and satisfying

$$\int_{-\infty}^{\infty} p(x)dx = 1. \tag{8.99}$$

All random number generators provided in the program library generate numbers in this domain.

When we attempt a transformation to a new variable $x \rightarrow y$ we have to conserve the probability

$$p(y)dy = p(x)dx, \tag{8.100}$$

which for the uniform distribution implies

$$p(y)dy = dx. \tag{8.101}$$

Let us assume that $p(y)$ is a PDF different from the uniform PDF $p(x) = 1$ with $x \in [0, 1]$. If we integrate the last expression we arrive at

$$x(y) = \int_0^y p(y')dy', \tag{8.102}$$

which is nothing but the cumulative distribution of $p(y)$, i.e.,

$$x(y) = P(y) = \int_0^y p(y')dy'. \tag{8.103}$$

This is an important result which has consequences for eventual improvements over the brute force Monte Carlo.

To illustrate this approach, let us look at some examples.

Example 1

Suppose we have the general uniform distribution

$$p(y)dy = \begin{cases} \frac{dy}{b-a} & a \leq y \leq b \\ 0 & else \end{cases} \tag{8.104}$$

If we wish to relate this distribution to the one in the interval $x \in [0, 1]$ we have

$$p(y)dy = \frac{dy}{b-a} = dx, \tag{8.105}$$

and integrating we obtain the cumulative function

$$x(y) = \int_a^y \frac{dy'}{b-a}, \tag{8.106}$$

yielding

$$y = a + (b-a)x, \tag{8.107}$$

a well-known result!

Example 2, the exponential distribution

Assume that

$$p(y) = e^{-y}, \tag{8.108}$$

which is the exponential distribution, important for the analysis of e.g., radioactive decay. Again, $p(x)$ is given by the uniform distribution with $x \in [0,1]$, and with the assumption that the probability is conserved we have

$$p(y)dy = e^{-y}dy = dx, \tag{8.109}$$

which yields after integration

$$x(y) = P(y) = \int_0^y \exp(-y')dy' = 1 - \exp(-y), \tag{8.110}$$

or

$$y(x) = -ln(1-x). \tag{8.111}$$

This gives us the new random variable y in the domain $y \in [0,\infty)$ determined through the random variable $x \in [0,1]$ generated by functions like $ran0$.

This means that if we can factor out $\exp(-y)$ from an integrand we may have

$$I = \int_0^\infty F(y)dy = \int_0^\infty \exp(-y)G(y)dy \tag{8.112}$$

which we rewrite as

$$\int_0^\infty \exp(-y)G(y)dy = \int_0^\infty \frac{dx}{dy}G(y)dy \approx \frac{1}{N}\sum_{i=1}^{N} G(y(x_i)), \tag{8.113}$$

where x_i is a random number in the interval [0,1]. Note that in practical implementations, our random number generators for the uniform distribution never return exactly 0 or 1, but we we may come very close. We should thus in principle set $x \in (0,1)$.

The algorithm for the last example is rather simple. In the function which sets up the integral, we simply need to call one of the random number generators like $ran0$, $ran1$, $ran2$ or $ran3$ in order to obtain numbers in the interval [0,1]. We obtain y by the taking the logarithm of $(1-x)$. Our calling function which sets up the new random variable y may then include statements like

```
.....
idum=-1;
x=ran0(&idum);
y=-log(1.-x);
.....
```

Exercise 8.4
Make a function exp_random which computes random numbers for the exponential distribution $p(y) = e^{-\alpha y}$ based on random numbers generated from the function $ran0$.

Example 3

Another function which provides an example for a PDF is

$$p(y)dy = \frac{dy}{(a+by)^n}, \tag{8.114}$$

with $n > 1$. It is normalizable, positive definite, analytically integrable and the integral is invertible, allowing thereby the expression of a new variable in terms of the old one. The integral

$$\int_0^\infty \frac{dy}{(a+by)^n} = \frac{1}{(n-1)ba^{n-1}}, \tag{8.115}$$

gives

$$p(y)dy = \frac{(n-1)ba^{n-1}}{(a+by)^n}dy, \tag{8.116}$$

which in turn gives the cumulative function

$$x(y) = P(y) = \int_0^y \frac{(n-1)ba^{n-1}}{(a+bx)^n}dy' =, \tag{8.117}$$

resulting in

$$x(y) = 1 - \frac{1}{(1+b/ay)^{n-1}}, \tag{8.118}$$

or

$$y = \frac{a}{b}\left((1-x)^{-1/(n-1)} - 1\right). \tag{8.119}$$

With the random variable $x \in [0,1]$ generated by functions like $ran0$, we have again the appropriate random variable y for a new PDF.

Example 4, the normal distribution

For the normal distribution, expressed here as

$$g(x,y) = \exp\left(-(x^2+y^2)/2\right)dxdy. \tag{8.120}$$

it is rather difficult to find an inverse since the cumulative distribution is given by the error function $erf(x)$.

If we however switch to polar coordinates, we have for x and y

$$r = \left(x^2+y^2\right)^{1/2} \qquad \theta = tan^{-1}\frac{x}{y}, \tag{8.121}$$

resulting in

$$g(r,\theta) = r\exp\left(-r^2/2\right)drd\theta, \tag{8.122}$$

where the angle θ could be given by a uniform distribution in the region $[0, 2\pi]$. Following example 1 above, this implies simply multiplying random numbers $x \in [0,1]$ by 2π. The variable r, defined for $r \in [0,\infty)$ needs to be related to to random numbers $x' \in [0,1]$. To achieve that, we introduce a new variable

$$u = \frac{1}{2}r^2, \tag{8.123}$$

and define a PDF

$$\exp(-u)du, \tag{8.124}$$

with $u \in [0, \infty)$. Using the results from example 2, we have that

$$u = -ln(1 - x'), \tag{8.125}$$

where x' is a random number generated for $x' \in [0, 1]$. With

$$x = rcos(\theta) = \sqrt{2u}cos(\theta), \tag{8.126}$$

and

$$y = rsin(\theta) = \sqrt{2u}sin(\theta), \tag{8.127}$$

we can obtain new random numbers x, y through

$$x = \sqrt{-2ln(1 - x')}cos(\theta), \tag{8.128}$$

and

$$y = \sqrt{-2ln(1 - x')}sin(\theta), \tag{8.129}$$

with $x' \in [0, 1]$ and $\theta \in 2\pi[0, 1]$.

A function which yields such random numbers for the normal distribution would include statements like

```
.....
idum=-1;
radius=sqrt(-2*ln(1.-ran0(idum)));
theta=2*pi*ran0(idum);
x=radius*cos(theta);
y=radius*sin(theta);
.....
```

Exercise 8.4
Make a function $normal\_random$ which computes random numbers for the normal distribution based on random numbers generated from the function $ran0$.

8.4.2 Importance sampling

With the aid of the above variable transformations we address now one of the most widely used approaches to Monte Carlo integration, namely importance sampling.

Let us assume that $p(y)$ is a PDF whose behavior resembles that of a function F defined in a certain interval $[a, b]$. The normalization condition is

$$\int_a^b p(y)dy = 1. \tag{8.130}$$

We can rewrite our integral as

$$I = \int_a^b F(y)dy = \int_a^b p(y)\frac{F(y)}{p(y)}dy. \tag{8.131}$$

This integral resembles our discussion on the evaluation of the energy for a quantum mechanical system in Eq. (8.94).

Since random numbers are generated for the uniform distribution $p(x)$ with $x \in [0,1]$, we need to perform a change of variables $x \to y$ through

$$x(y) = \int_a^y p(y')dy', \tag{8.132}$$

where we used

$$p(x)dx = dx = p(y)dy. \tag{8.133}$$

If we can invert $x(y)$, we find $y(x)$ as well.

With this change of variables we can express the integral of Eq. (8.131) as

$$I = \int_a^b p(y)\frac{F(y)}{p(y)}dy = \int_a^b \frac{F(y(x))}{p(y(x))}dx, \tag{8.134}$$

meaning that a Monte Carlo evalutaion of the above integral gives

$$\int_a^b \frac{F(y(x))}{p(y(x))}dx = \frac{1}{N}\sum_{i=1}^{N}\frac{F(y(x_i))}{p(y(x_i))}. \tag{8.135}$$

The advantage of such a change of variables in case $p(y)$ follows closely F is that the integrand becomes smooth and we can sample over relevant values for the integrand. It is however not trivial to find such a function p. The conditions on p which allow us to perform these transformations are

1. p is normalizable and positive definite,
2. it is analytically integrable and
3. the integral is invertible, allowing us thereby to express a new variable in terms of the old one.

The variance is now with the definition

$$\tilde{F} = \frac{F(y(x))}{p(y(x))}, \tag{8.136}$$

given by

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}\left(\tilde{F}\right)^2 - \left(\frac{1}{N}\sum_{i=1}^{N}\tilde{F}\right)^2. \tag{8.137}$$

The algorithm for this procedure is

- Use the uniform distribution to find the random variable y in the interval [0,1]. $p(x)$ is a user provided PDF.
- Evaluate thereafter

$$I = \int_a^b F(x)dx = \int_a^b p(x)\frac{F(x)}{p(x)}dx, \tag{8.138}$$

by rewriting

$$\int_a^b p(x)\frac{F(x)}{p(x)}dx = \int_a^b \frac{F(x(y))}{p(x(y))}dy, \tag{8.139}$$

since

$$\frac{dy}{dx} = p(x). \tag{8.140}$$

– Perform then a Monte Carlo sampling for

$$\int_a^b \frac{F(x(y))}{p(x(y))}dy, \approx \frac{1}{N}\sum_{i=1}^{N}\frac{F(x(y_i))}{p(x(y_i))}, \tag{8.141}$$

with $y_i \in [0,1]$,

– and evaluate the variance as well according to Eq. (8.137).

Exercise 8.5

(a) Calculate the integral

$$I = \int_0^1 e^{-x^2}dx,$$

using brute force Monte Carlo with $p(x) = 1$ and importance sampling with $p(x) = ae^{-x}$ where a is a constant.

(b) Calculate the integral

$$I = \int_0^\pi \frac{1}{x^2 + cos^2(x)}dx,$$

with $p(x) = ae^{-x}$ where a is a constant. Determine the value of a which minimizes the variance.

8.4.3 Acceptance-Rejection method

This is rather simple and appealing method after von Neumann. Assume that we are looking at an interval $x \in [a,b]$, this being the domain of the PDF $p(x)$. Suppose also that the largest value our distribution function takes in this interval is M, that is

$$p(x) \leq M \qquad x \in [a,b]. \tag{8.142}$$

Then we generate a random number x from the uniform distribution for $x \in [a,b]$ and a corresponding number s for the uniform distribution between $[0,M]$. If

$$p(x) \geq s, \tag{8.143}$$

we accept the new value of x, else we generate again two new random numbers x and s and perform the test in the latter equation again.

As an example, consider the evaluation of the integral

$$I = \int_0^3 \exp(x)dx.$$

Obviously to derive it analytically is much easier, however the integrand could pose some more difficult challenges. The aim here is simply to show how to implent the acceptance-rejection algorithm. The integral is the area below the curve $f(x) = \exp(x)$. If we uniformly fill the rectangle spanned by $x \in [0,3]$ and $y \in [0, \exp(3)]$, the fraction below the curve obatained from a uniform distribution, and multiplied by the area of the rectangle, should approximate the chosen integral. It is rather easy to implement this numerically, as shown in the following code.

Acceptance-Rejection algorithm

```
//    Loop over Monte Carlo trials n
      integral =0.;
      for ( int i = 1;  i <= n; i++){
//    Finds a random value for x in the interval [0,3]
           x = 3*ran0(&idum);
//    Finds y-value between [0,exp(3)]
           y = exp(3.0)*ran0(&idum);
//    if the value of y at exp(x) is below the curve, we accept
           if ( y  < exp(x)) s = s+ 1.0;
//    The integral is area enclosed below the line f(x)=exp(x)
     }
//   Then we multiply with the area of the rectangle and divide by the number
     of cycles
     Integral = 3.*exp(3.)*s/n
```

8.5 Monte Carlo integration of multidimensional integrals

When we deal with multidimensional integrals of the form

$$I = \int_0^1 dx_1 \int_0^1 dx_2 \ldots \int_0^1 dx_d g(x_1, \ldots, x_d), \tag{8.144}$$

with x_i defined in the interval $[a_i, b_i]$ we would typically need a transformation of variables of the form

$$x_i = a_i + (b_i - a_i)t_i,$$

if we were to use the uniform distribution on the interval $[0, 1]$. In this case, we need a Jacobi determinant

$$\prod_{i=1}^{d}(b_i - a_i),$$

and to convert the function $g(x_1, \ldots, x_d)$ to

$$g(x_1, \ldots, x_d) \rightarrow g(a_1 + (b_1 - a_1)t_1, \ldots, a_d + (b_d - a_d)t_d).$$

As an example, consider the following sixth-dimensional integral

$$\int_{-\infty}^{\infty} \mathbf{dxdy}g(\mathbf{x}, \mathbf{y}), \tag{8.145}$$

where

$$g(\mathbf{x}, \mathbf{y}) = \exp\left(-\mathbf{x}^2 - \mathbf{y}^2 - (\mathbf{x} - \mathbf{y})^2/2\right), \tag{8.146}$$

with $d = 6$.

We can solve this integral by employing our brute force scheme, or using importance sampling and random variables distributed according to a gaussian PDF. For the latter, if we set the mean value $\mu = 0$ and the standard deviation $\sigma = 1/\sqrt{2}$, we have

$$\frac{1}{\sqrt{\pi}} \exp\left(-x^2\right), \tag{8.147}$$

and through

$$\pi^3 \int \prod_{i=1}^{6} \left(\frac{1}{\sqrt{\pi}} \exp\left(-x_i^2\right) \right) \exp\left(-(\mathbf{x}-\mathbf{y})^2/2\right) dx_1 \ldots dx_6, \tag{8.148}$$

we can rewrite our integral as

$$\int f(x_1, \ldots, x_d) F(x_1, \ldots, x_d) \prod_{i=1}^{6} dx_i, \tag{8.149}$$

where f is the gaussian distribution.

Below we list two codes, one for the brute force integration and the other employing importance sampling with a gaussian distribution.

8.5.1 Brute force integration

http://folk.uio.no/mhjensen/fys3150/2005/programs/chapter8/program4.cpp

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include "lib.h"
using namespace std;

double brute_force_MC(double *);
//      Main function begins here
int main()
{
     int n;
     double x[6], y, fx;
     double int_mc = 0.;  double variance = 0.;
     double sum_sigma= 0. ; long idum=-1 ;
     double length=5.; // we fix the max size of the box to L=5
     double volume=pow((2*length),6);
     cout << "Read in the number of Monte-Carlo samples" << endl;
     cin >> n;
//   evaluate the integral with importance sampling
     for ( int i = 1;  i <= n; i++){
//   x[] contains the random numbers for all dimensions
       for (int j = 0; j< 6; j++) {
           x[j]=-length+2*length*ran0(&idum);
       }
       fx=brute_force_MC(x);
       int_mc += fx;
       sum_sigma += fx*fx;
     }
     int_mc = int_mc/((double) n );
     sum_sigma = sum_sigma/((double) n );
     variance=sum_sigma-int_mc*int_mc;
//   final output
      cout << setiosflags(ios::showpoint | ios::uppercase);
      cout << " Monte carlo result= " << setw(10) << setprecision(8) <<
         volume*int_mc;
```

```
      cout << " Sigma= " << setw(10) << setprecision(8) << volume*sqrt(
         variance/((double) n )) << endl;
      return 0;
}  // end of main program

// this function defines the integrand to integrate

double  brute_force_MC(double *x)
{
   double a = 1.; double b = 0.5;
// evaluate the different terms of the exponential
   double xx=x[0]*x[0]+x[1]*x[1]+x[2]*x[2];
   double yy=x[3]*x[3]+x[4]*x[4]+x[5]*x[5];
   double xy=pow((x[0]-x[3]),2)+pow((x[1]-x[4]),2)+pow((x[2]-x[5]),2);
   return exp(-a*xx-a*yy-b*xy);
} // end function for the integrand
```

8.5.2 *Importance sampling*

This code includes a call to the function $normal\_random$, which produces random numbers from a gaussian distribution.

http://folk.uio.no/mhjensen/fys3150/2005/programs/chapter8/program5.cpp

```
// importance sampling with gaussian deviates
#include <iostream>
#include <fstream>
#include <iomanip>
#include "lib.h"
using namespace std;

double gaussian_MC(double *);
double gaussian_deviate(long *);
//      Main function begins here
int main()
{
     int n;
     double x[6], y, fx;
     cout << "Read in the number of Monte-Carlo samples" << endl;
     cin >> n;
     double int_mc = 0.;  double variance = 0.;
     double sum_sigma= 0. ; long idum=-1 ;
     double length=5.; // we fix the max size of the box to L=5
     double volume=pow(acos(-1.),3.);
     double sqrt2 = 1./sqrt(2.);
//   evaluate the integral with importance sampling
     for ( int i = 1;  i <= n; i++){
//   x[] contains the random numbers for all dimensions
       for (int j = 0; j < 6; j++) {
         x[j] = gaussian_deviate(&idum)*sqrt2;
       }
       fx=gaussian_MC(x);
       int_mc += fx;
```

```
        sum_sigma += fx*fx;
      }
      int_mc = int_mc/((double) n );
      sum_sigma = sum_sigma/((double) n );
      variance=sum_sigma-int_mc*int_mc;
//    final output
       cout << setiosflags(ios::showpoint | ios::uppercase);
       cout << " Monte carlo result= " << setw(10) << setprecision(8) <<
           volume*int_mc;
       cout << " Sigma= " << setw(10) << setprecision(8) << volume*sqrt(
           variance/((double) n )) << endl;
      return 0;
}  // end of main program

// this function defines the integrand to integrate

double  gaussian_MC(double *x)
{
   double a = 0.5;
// evaluate the different terms of the exponential
   double xy=pow((x[0]-x[3]),2)+pow((x[1]-x[4]),2)+pow((x[2]-x[5]),2);
   return exp(-a*xy);
} // end function for the integrand

// random numbers with gaussian distribution
double gaussian_deviate(long * idum)
{
  static int iset = 0;
  static double gset;
  double fac, rsq, v1, v2;

  if ( idum < 0) iset =0;
  if (iset == 0) {
    do {
      v1 = 2.*ran0(idum) -1.0;
      v2 = 2.*ran0(idum) -1.0;
      rsq = v1*v1+v2*v2;
    } while (rsq >= 1.0 || rsq == 0.);
    fac = sqrt(-2.*log(rsq)/rsq);
    gset = v1*fac;
    iset = 1;
    return v2*fac;
  } else {
    iset =0;
    return gset;
  }
} // end function for gaussian deviates
```

The following table lists the results from the above two programs as function of the number of Monte Carlo samples. The suffix *cr* stands for the brute force approach while *gd* stands for the use of a Gaussian distribution function. One sees clearly that the approachwith a Gaussian distribution function yields a much improved numerical result, with fewer samples.

Table 8.4: Results for as function of number of Monte Carlo samples N. The exact answer is $I \approx 10.9626$ for the integral. The suffix cr stands for the brute force approach while gd stands for the use of a Gaussian distribution function. All calculations use ran0 as function to generate the uniform distribution.

N	I_{cr}	I_{gd}
10000	1.15247E+01	1.09128E+01
100000	1.29650E+01	1.09522E+01
1000000	1.18226E+01	1.09673E+01
10000000	1.04925E+01	1.09612E+01

8.6 Physics Project: Decay of ^{210}Bi and ^{210}Po

In this project we are going to simulate the radioactive decay of these nuclei using sampling through random numbers. We assume that at $t = 0$ we have $N_X(0)$ nuclei of the type X which can decay radioactively. At a given time t we are left with $N_X(t)$ nuclei. With a transition rate ω_X, which is the probability that the system will make a transition to another state during a time step of one second, we get the following differential equation

$$dN_X(t) = -\omega_X N_X(t)dt,$$

whose solution is

$$N_X(t) = N_X(0)e^{-\omega_X t},$$

and where the mean lifetime of the nucleus X is

$$\tau = \frac{1}{\omega_X}.$$

If the nucleus X decays to Y, which can also decay, we get the following coupled equations

$$\frac{dN_X(t)}{dt} = -\omega_X N_X(t),$$

and

$$\frac{dN_Y(t)}{dt} = -\omega_Y N_Y(t) + \omega_X N_X(t).$$

We assume that at $t = 0$ we have $N_Y(0) = 0$. In the beginning we will have an increase of N_Y nuclei, however, they will decay thereafter. In this project we let the nucleus ^{210}Bi represent X. It decays through β-decay to ^{210}Po, which is the Y nucleus in our case. The latter decays through emision of an α-particle to ^{206}Pb, which is a stable nucleus. ^{210}Bi has a mean lifetime of 7.2 days while ^{210}Po has a mean lifetime of 200 days.

a) Find analytic solutions for the above equations assuming continuous variables and setting the number of ^{210}Po nuclei equal zero at $t = 0$.

b) Make a program which solves the above equations. What is a reasonable choice of timestep Δt? You could use the program on radioactive decay from the web-page of the course as an example and make your own for the decay of two nuclei. Compare the results from your program with the exact answer as function of $N_X(0) = 10$, 100 and 1000. Make plots of your results.

c) When ^{210}Po decays it produces an α particle. At what time does the production of α particles reach its maximum? Compare your results with the analytic ones for $N_X(0) = 10$, 100 and 1000.

8.7 Physics project: Numerical integration of the correlation energy of the helium atom

The task of this project is to integrate in a brute force manner a six-dimensional integral which is used to determine the ground state correlation energy between two electrons in a helium atom. We will employ both Gauss-Legendre quadrature and Monte-Carlo integration. Furthermore, you will need to parallelize your code for the Monte-Carlo integration.

We assume that the wave function of each electron can be modelled like the single-particle wave function of an electron in the hydrogen atom. The single-particle wave function for an electron i in the $1s$ state is given in terms of a dimensionless variable (the wave function is not properly normalized)

$$\mathbf{r}_i = x_i\mathbf{e}_x + y_i\mathbf{e}_y + z_i\mathbf{e}_z,$$

as

$$\psi_{1s}(\mathbf{r}_i) = e^{-\alpha r_i},$$

where α is a parameter and

$$r_i = \sqrt{x_i^2 + y_i^2 + z_i^2}.$$

We will fix $\alpha = 2$, which should correspond to the charge of the helium atom $Z = 2$.

The ansatz for the wave function for two electrons is then given by the product of two $1s$ wave functions as

$$\Psi(\mathbf{r}_1, \mathbf{r}_2) = e^{-\alpha(r_1+r_2)}.$$

Note that it is not possible to find an analytic solution to Schrödinger's equation for two interacting electrons in the helium atom.

The integral we need to solve is the quantum mechanical expectation value of the correlation energy between two electrons, namely

$$\langle \frac{1}{|\mathbf{r}_1 - \mathbf{r}_2|} \rangle = \int_{-\infty}^{\infty} d\mathbf{r}_1 d\mathbf{r}_2 e^{-2\alpha(r_1+r_2)} \frac{1}{|\mathbf{r}_1 - \mathbf{r}_2|}. \tag{8.150}$$

Note that our wave function is not normalized. There is a normalization factor missing, but for this project we don't need to worry about that.

a) Use Gauss-Legendre quadrature and compute the integral by integrating for each variable x_1, y_1, z_1, x_2, y_2, z_2 from $-\infty$ to ∞. How many mesh points do you need before the results converges at the level of the fourth leading digit? Hint: the single-particle wave function $e^{-\alpha r_i}$ is more or less zero at $r_i \approx 10 - 15$. You can therefore replace the integration limits $-\infty$ and ∞ with -10 and 10, respectively. You need to check that this approximation is satisfactory.

b) Compute the same integral but now with brute force Monte Carlo and compare your results with those from the previous point. Discuss the differences. With bruce force we mean that you should use the uniform distribution.

c) Improve your brute force Monte Carlo calculation by using importance sampling. Hint: use the exponential distribution. Does the variance decrease? Does the CPU time used compared with the brute force Monte Carlo decrease in order to achieve the same accuracy? Comment your results.

d) Parallelize your code from the previous point and compare the CPU time needed with that from point [c)]. Do you achieve a good speedup?

e) The integral of Eq. (8.150) has an analytical expression. Can you find it?