LECTURE 9: Monte Carlo Methods I

November 7, 2011

1 Introduction

1.1 Brief history of the Monte Carlo method

The idea of the Monte Carlo (MC) method is a lot older than the computer. The name *Monte Carlo* is relatively new - it was suggested by Nicolas Metropolis in 1949 because of the similarity of statistical simulation to games of chance (Monaco was the center of gambling). Under the name *statistical sampling*, the MC method stretches back to times when numerical calculations were performed using pencil and paper.

At first, Monte Carlo was a method for estimating integrals which could not be solved by other means. Integrals over poorly-behaved functions and multidimensional integrals were profitable subjects of the MC method.

The famous physicist Richard Feynman realized around the time of the Second World War that the time of electronic computing was just around the corner. He created what could be described as a highly pipelined human CPU, by employing a large number of people to use mechanical adding machines in an arithmetic assembly line. A number of crucial calculations to the design of the atomic bomb were performed in this way.

It was in the last months of the Second World War when the new ENIAC electronic computer was used for the first time to perform numerical calculations. The technology that went into ENIAC had existed even before but the war had slowed down the construction of the machine. The idea of using randomness for calculations occurred to Stan Ulam while he was playing a game of cards. He realized that he could calculate the probability of a certain event simply by repeating the game over and over again. From there it was a simple step to realize that the computer could play the games for him.

This seems obvious now, but it is actually a subtle question that a physical problem with an exact answer can be approximately solved by studying a suitable random process.

Nowadays Monte Carlo has grown to become the most powerful method for solving problems in statistical physics - among many other applications.

The name Monte Carlo is used as a general term for a wide class of stochastic methods. The common factor is that random numbers are used for sampling.

2 Monte Carlo integration - simple sampling

One of the simplest but also effective uses of the Monte Carlo method is the evaluation of integrals which are intractable by analytic techniques. In the simplest case, we wish to

obtain the one-dimensional integral of $f(x)$ over some fixed interval $[a,b]$:

$$I = \int_a^b f(x)dx$$

One-dimensional integrals can be effectively calculated using discrete approximations such as the trapezoidal rule or Simpson's rule (discussed in Chapter 5). In order to illustrate the MC integration technique, we apply it first to the one-dimensional case and then extend the discussion to multidimensional integrals (where the other methods become computationally very expensive and thus less effective).

2.1 Hit-or-miss method

In the so-called *"hit-or-miss"* Monte Carlo integration, the definite integral is estimated by drawing a box which bounds the function $f(x)$ in the interval $[a,b]$; i.e. the box extends from a to b and from 0 to f_{max} where $f_{max} > f(x)$ throughout the interval. Then N points are dropped randomly into the box. An estimate for the integral can now be obtained by calculating the total number of points N_0 which fall under the curve of $f(x)$; i.e.

$$I_{est} = \frac{N_0}{N} A$$

where $A = (b-a) f_{max}$ is the area of the box.

Each of N random points is obtained by generating 2 uniformly distributed random numbers s_1 and s_2 and taking

$$x = a + s_1 * (b-a)$$
$$y = s_2 * f_{max}$$

as the x and y coordinates of the point.

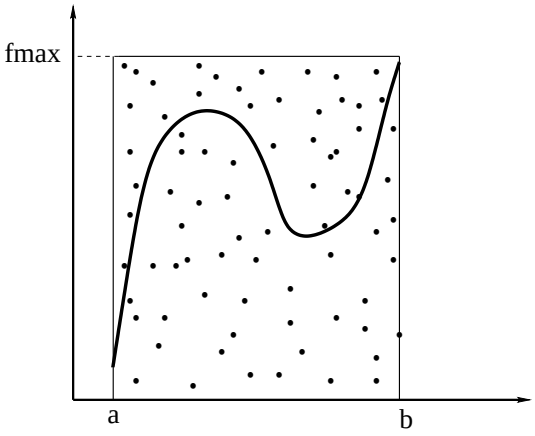


Figure 1: Hit-or-miss MC. Points are generated randomly inside a rectangular area which bounds the function $f(x)$ in the interval $[a,b]$. The number of points under the curve is calculated to obtain an estimate of the integral.

The estimate of the integral becomes increasingly precise as $N \to \infty$ and will eventually converge to the correct answer. Obviously, the quality of the answer depends on the quality of the random number generator sequence which is used. We can obtain independent estimates by repeating the calculation using a different random number sequence. Comparing the values gives an idea of the precision of the calculation.

Example 1: Estimating the value of π

As a simple illustration of the "hit-or-miss" method, we can consider the estimation of the numerical value of π. We generate N random points in the square $x \in [-1,1]$ and $y \in [-1,1]$. Then we calculate how many of those points landed inside the circle $x^2+y^2=1$. Denote this number by N_0. The ratio of the two areas is

$$\frac{A_{\text{circle}}}{A_{\text{square}}} = \frac{\pi R^2}{4R^2} = \frac{\pi}{4} \approx \frac{N_0}{N}$$

Thus an estimate of π is obtained from

$$\pi \approx \frac{4N_0}{N}$$

The following C code can be used to calculate an estimate of π for $N = 1000 - 10000$.

```
void   init_random_generator(long iseed);
double random_generator();

int main(void)
{
  long int iseed=23455;
  int i, j, n, n_tot, n_circle;
  double x, y, ratio;

  /* Initialize random number generator */
  init_random_generator(iseed);
  n_tot=1000;

  /* Loop over different values of N */
  for(n=0; n<10; n++) {
    /* Throw N points within a box [-1,1][-1,1] */
    n_circle = 0;
    for(i=0; i<n_tot; i++) {
      x = 2.0*random_generator()-1.0;
      y = 2.0*random_generator()-1.0;
      /* Determine if inside the circle */
      if(x*x+y*y < 1.0) n_circle ++;
    }
    ratio = (double)n_circle/(double)n_tot;
    /* Print out the obtained value of PI */
    fprintf(stdout,"N =  %d  PI = %10.8lf\n",n_tot,ratio*4.0);
    /* increase the number of points */
    n_tot += 1000;
  }
}
```

Figure 2 shows the distribution of 1000 random points inside the square.

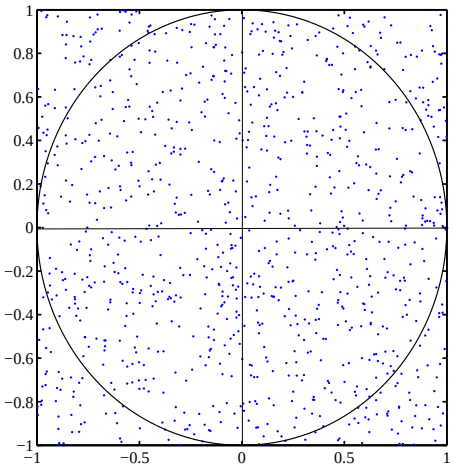


Figure 2: Distribution of 1000 random points.

Increasing the number of points inside the square, we get the following sequence of estimates.

```
N =  1000   PI = 3.13200000
N =  2000   PI = 3.14200000
N =  3000   PI = 3.10800000
N =  4000   PI = 3.11800000
N =  5000   PI = 3.13040000
N =  6000   PI = 3.12266667
N =  7000   PI = 3.15142857
N =  8000   PI = 3.12450000
N =  9000   PI = 3.15333333
N =  10000  PI = 3.15600000

EXACT VALUE    = 3.14159265
```

The estimate can be improved by choosing a certain value of N (e.g. $N = 10000$) and calculating the average value obtained from several independent calculations (where in each N points are placed inside the square using different random number sequences).

The following result was obtained using 1000 independent measurements.

```
N =  10000   m = 1000  PI = 3.14140240
```

This illustrates a *general feature* of the Monte Carlo method: The precision of the calculation is determined by the statistics of the sampling.

A single measurement gives us a crude estimate of the answer but repeating the process many times and calculating the average of the independent measurements improves the precision, and moreover, the *errors can be analyzed* using principles of statistics.

Example 2: Integration of the function $\sin^2 \frac{1}{x}$

To give an example of a difficult integration task, consider the function (shown in Fig. 3)

$$f(x) = sin^2 \frac{1}{x}$$

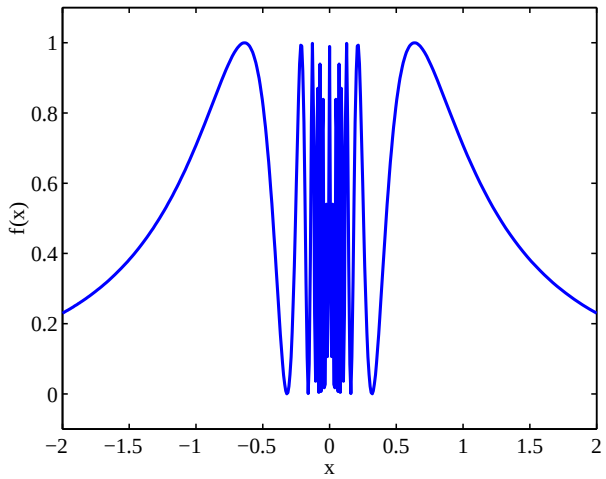


Figure 3: Plot of function $\sin^2(1/x)$.

The integral of this function is hard to solve analytically, but can be evaluated in a straightforward manner using MC integration. There are a number of more sophisticated MC integration techniques (which will be discussed later), but we use this example here to illustrate the idea of the simple "hit-or-miss" -algorithm.

The aim is to calculate the integral

$$I = \int_a^b f(x)dx = \int_a^b sin^2 \frac{1}{x} dx$$

We select $a = 0$ and $b = 1$. The maximum value of $f(x)$ on this interval is $f_{max} = 1$. Thus the rectangular area where we will randomly drop N points (x,y) is $x \in [0,1]$ and $y \in [0, f_{max}]$. Then we count the number of points for which $y < f(x)$. Denote this number by N_0. An estimate of the value of the integral is now given by

$$I_{est} = \frac{N_0}{N}(b-a) f_{max} = \frac{N_0}{N}$$

We obtain the following results:

```
N =  100000     I = 0.6706900
N =  1000000    I = 0.6725830
N =  10000000   I = 0.6734556

correct result: I = 0.6734568
```

With ten million points, we get the correct answer to the fifth decimal. The following Matlab commands

```
syms x
i = int(sin(1/x)*sin(1/x),0,1)
eval(i)
```

can be used to calculate the definite integral of $f(x)$ from 0 to 1.

Figure 4 shows the absolute error of the estimated answer obtained by MC integration as a function of total number of points used N.

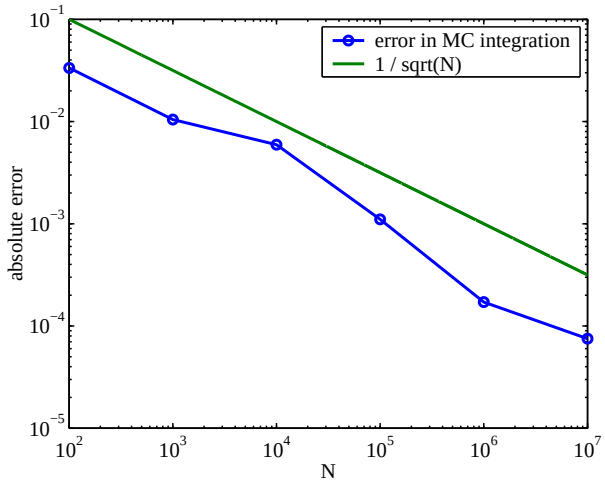


Figure 4: Absolute error of the MC estimate.

The error goes as $1/\sqrt{N}$. This is not at all competitive with algorithms such as the Romberg method, but in higher dimensions the MC method becomes much more efficient.

2.2 Sample-mean method

Another simple Monte Carlo integration technique is termed as the "sample-mean" method (also called the "crude" method). In this approach we select N values of x randomly from the interval $[0,1]$ and then approximate the integral by

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

Thus we approximate the integral by the average of N numbers $f(x_1), f(x_2), \ldots, f(x_N)$. Again the estimated answer eventually converges to the correct result as $N \to \infty$.

In a general case where the interval is $[a,b]$, the estimate of the integral is given by

$$I_{est} = (b-a)\langle f \rangle = (b-a)\frac{1}{N}\sum_{i=1}^{N} f(x_i)$$

The **statistical error** involved in MC integration is given by

$$\sigma \approx \frac{\sigma_N}{\sqrt{N}}$$

where

$$\sigma_N^2 = \langle f^2 \rangle - \langle f \rangle^2 = \frac{1}{N}\sum_{i=1}^{N} f^2(x_i) - \left[\frac{1}{N}\sum_{i=1}^{N} f(x_i)\right]^2$$

Higher-dimensional integrals

Monte Carlo methods become particularly attractive when we consider integration in higher dimensions. For example,

$$I = \int_0^1 \int_0^1 \int_0^1 f(x,y,z)\,dx\,dy\,dz \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i,y_i,z_i)$$

where (x_i,y_i,z_i) is a random sequence of points in the unit cube $x,y,z \in [0,1]$. In total, we need $3N$ random numbers in order to generate the N random points.

An integral over an arbitrary integration area is obtained from

$$\begin{aligned} I &= \int_{a_1}^{a_2} \int_{b_1}^{b_2} \int_{c_1}^{c_2} f(x,y,z)\,dx\,dy\,dz \\ &= \int_{a_1}^{a_2} \left[\int_{b_1}^{b_2} \left(\int_{c_1}^{c_2} f(x,y,z)\,dx \right) dy \right] dz \\ &\approx (a_2-a_1)(b_2-b_1)(c_2-c_1)\frac{1}{N}\sum_{i=1}^{N} f(x_i,y_i,z_i) \end{aligned}$$

In general, the sample-mean method is given by

$$I = \int_A f \approx (\text{measure of } A) \times (\text{average of } f \text{ over } n \text{ random points in } A)$$

Errors

In *standard numerical integration*, the error decreases as n^{-a} for $d = 1$ (one-dimensional integral); e.g. the trapezoid rule has n^{-2}. In d dimensions, the error decreases as $n^{-a/d}$. (Here n denotes the total number of subintervals; i.e. the number of subintervals per dimension is $n^{1/d}$).

The error in *MC integration* decreases *always* as $N^{-1/2}$.

The computational cost is relative to the total number of subintervals n or the total number of points N. From this we can deduce the following:

MC integration is more efficient for $d > 2a$.

Example 1: Two-dimensional integral

Consider the integral of $f(x,y) = xye^{-x^2y}$ over the square $x,y \in [0,1]$.

Using the MC sample mean method, the value of this integral is estimated by generating N points randomly in the square and calculating the average of $f(x,y)$ over these points. Thus

$$I_{MC} = \frac{1}{N}\sum_{i=1}^{N} f(x_i,y_i)$$

Here the area factor is $A = 1$.

The exact value of the integral is

$$I = \int_0^1 \int_0^1 xye^{-x^2y}dxdy = \frac{1}{2e} \approx 0.18393972$$

Figure 5 below shows the absolute error $|I_{MC} - I|$.

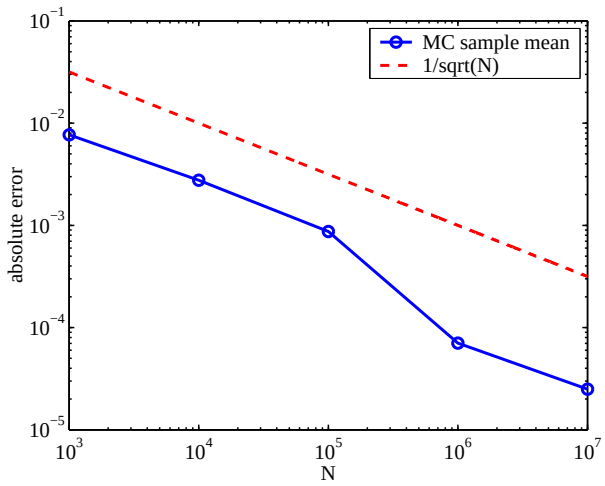


Figure 5: Absolute error of MC estimate.

The error decreases as $1/\sqrt{N}$ as expected from theory, but notice that there is *uncertainty* associated with each estimate (the line is not straight). This is due to the fact that we are using random numbers to calculate the estimates. If we use a sequence of N random numbers and then repeat the calculation with another sequence of N random numbers, we get two different answers. The distribution of answers will become narrower as we increase N and the sequence will eventually converge to the correct answer.

The error in a Monte Carlo calculation is fundamentally different from that in the other methods of integration. For example, consider the trapezoid rule. In this case, the error is due to the linear approximation that is made in each subinterval. By making the subintervals smaller, the fit is made better and the error decreases.

As an example, the Figure 6 shows the error when the trapezoid rule is applied in the one-dimensional case. The error decreases as n^{-2} and there are no fluctuations when the uniform spacing is used in all calculations.

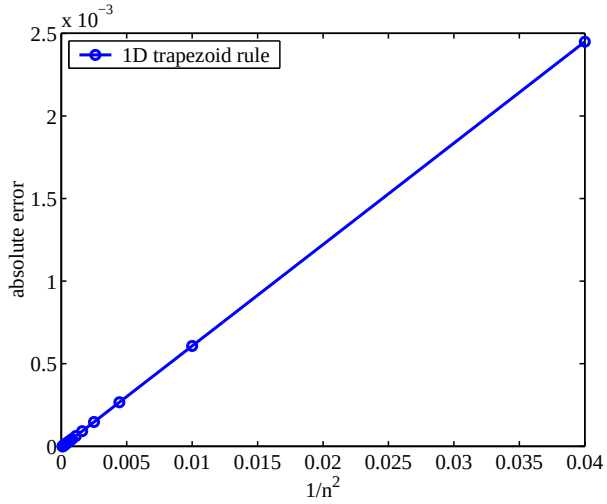


Figure 6: Absolute error of trapezoidal estimate.

Example 2: Computing volumes

The volume of a complicated region in three-dimensional space can be computed by a Monte Carlo technique. Taking a relatively simple case as an example, consider the volume of the region where points satisfy the condition

$$\begin{cases} 0 \leq x \leq 1 \quad 0 \leq y \leq 1 \quad 0 \leq z \leq 1 \\ x^2 + \sin y \leq z \\ x - z + e^y \leq 1 \end{cases}$$

The first line defines a cube of volume 1. The two other inequalities define a volume which is a subset of the cube. We can therefore apply the "hit-or-miss" method by generating N points inside the cube and determining how many of those points satisfy the two last inequalities (N_0). Then an estimate of the desired volume is given by N_0/N.

In the following segment of C code, the estimate of the volume is obtained by taking m independent calculations of the value (where in each N random points are dropped in the square). An estimate of the precision is given by variance

$$\sigma_N^2 = \langle V_N^2 \rangle - \langle V_N \rangle^2$$

where N is fixed and V denotes an estimate of the volume.

```
/* Loop over independent calculations */
for(j=0; j<m; j++) {

  n_0 = 0;  /* Points in given volume */

  /* Throw n_tot points within the unit cube */
  for(i=0; i<n_tot; i++) {
    x = random_generator();
    y = random_generator();
    z = random_generator();

    /* Determine if inside the volume */
    if(f1(x,y) <= z && f2(x,y,z) <= 1.0) n_0++;
  }

  /* Calculate the ratio n_0 / n_tot */
  ratio = (double)n_0/(double)n_tot;

  /* Update averages */
  vol += ratio;
  vol2 += ratio*ratio;
}

vol /= (double)m;
vol2 /= (double)m;

std = vol2 - vol*vol;
```

Figure 7 shows the estimated volume (the average volume obtained from $m = 20$ independent calculations) as a function of N (the number of random points in the square per independent measurement).

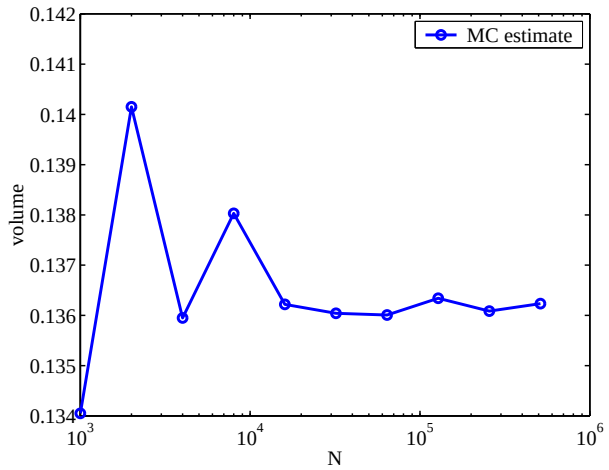


Figure 7: Estimated volume as a function of N.

The variance is shown in Fig. 8.

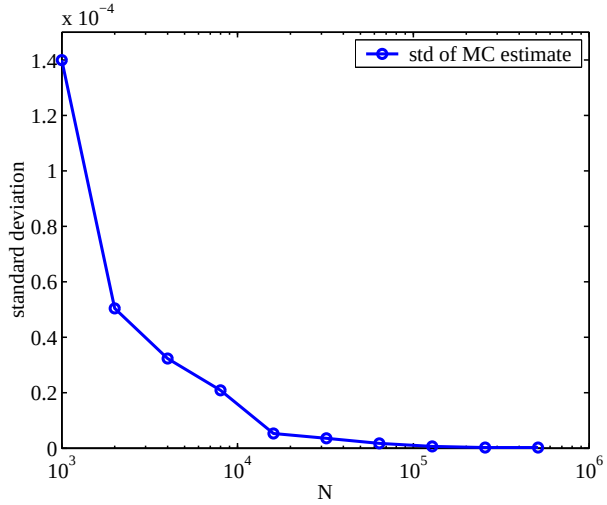


Figure 8: Variance of the estimated volume.

3 Monte Carlo integration - intelligent methods

3.1 Importance sampling

The accuracy of the Monte Carlo integration technique can be improved if we have some *a priori* knowledge about the integrand. In particular, if we can identify those regions of the function that have the greatest impact on the integral, then we can ensure that those regions receive the majority of random points. This means that we choose the points according to their anticipated importance to the value of the integral and then weight the contribution of each point by the inverse of the probability by which it was chosen.

In practice, we introduce a positive normalized *weight function* $w(x)$ for which

$$\int_a^b w(x)dx = 1$$

and then rewrite the integral as

$$I = \int_a^b \left[\frac{f(x)}{w(x)}\right] w(x)dx$$

The integral is evaluated by generating random numbers *according to the weight function* $w(x)$ (which is a probability distribution function). Either the *analytic transformation of probability densities* or the *rejection method* can be used for generating random numbers which follow the distribution $w(x)$.

Using importance sampling, the *estimate* for the integral is given by

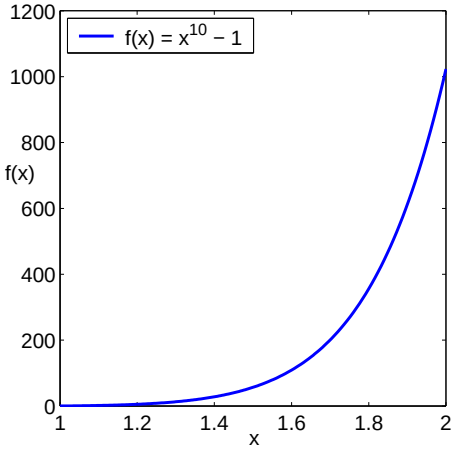
$$I_{est} = \frac{1}{N}\sum_{i=1}^{N}\frac{f(x_i)}{w(x_i)}$$

where the points x_i are generated randomly according to the distribution $w(x)$. Note that the function $w(x)$ must be normalized according to the given condition so that all the numbers x_i are in $[a,b]$.

In general, $w(x)$ should be chosen to mimic the form of $f(x)$ as closely as possible in order to make $f(x)/w(x)$ slowly varying.

Example

Let us consider calculating the integral of $f(x) = x^{10} - 1$ between 1 and 2 using both the sample mean method and importance sampling. The following figure shows the graph of $f(x)$ in the integration area:



We notice that most of the contribution to the integral comes from the area near $x = 2$, thus the importance sampling method is expected to improve the accuracy of the result compared to simple sampling.

We begin by choosing a weight function $w(x)$ which is similar to $f(x)$. In order to make analytic transformation of probability densities easy, we choose the following form

$$w(x) = Cx^{10}$$

where C is a constant which is determined by the normalization condition:

$$\int_a^b w(x)dx = 1 \;\Rightarrow\; \int_1^2 Cx^{10}dx = 1$$
$$\Rightarrow\; C = \frac{11}{2^{11}-1} \approx 0.0054$$

The cumulative distribution function $F(x)$ is given by

$$F(x) = \int_a^x w(x)dx = \frac{11}{2^{11}-1}\int_1^x x^{10}dx = \frac{x^{11}-1}{2^{11}-1} = y$$

The formula for the transformation between probability densities gives

$$x = F^{-1}(y) = [(2^{11}-1)y+1]^{1/11}$$

where the numbers y are uniformly distributed in $[0,1]$.

Figure 9 shows how 1000 random numbers x_i are distributed in $[1,2]$. This is what we wanted: the area near $x = 2$ is clearly favored.

We are now ready to compute the estimate for the integral by calculating the average of $f(x)/w(x)$ over randomly distributed points x_i which follow $w(x)$:

$$I_{est} = \frac{1}{N}\sum_{i=1}^{N}\frac{f(x_i)}{w(x_i)}$$

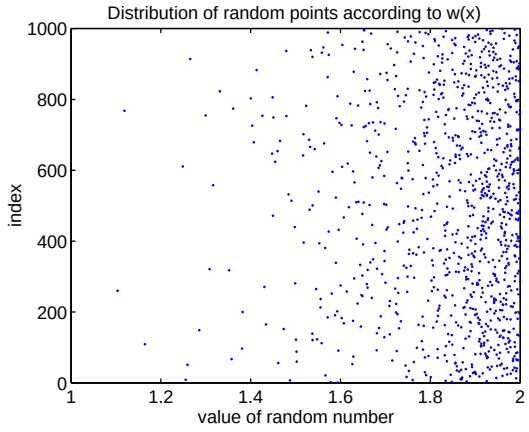


Figure 9: Distribution of random points.

We obtain the following results:

```
SAMPLE MEAN METHOD:
N          I              error
1000       197.78778405   12.69687496
10000      183.23575316   1.85515593
100000     184.46495114   0.62595795
1000000    185.48523664   0.39432755
10000000   184.90758678   0.18332232

IMPORTANCE SAMPLING:
N          I              error
1000       185.14836846   0.05745937
10000      185.12627991   0.03537082
100000     185.08113864   0.00977046
1000000    185.09466645   0.00375735
10000000   185.08989139   0.00101771
```

The convergence is much faster when importance sampling is used. Figure 10 shows the absolute error as a function of N (number of random points) on a logarithmic scale.

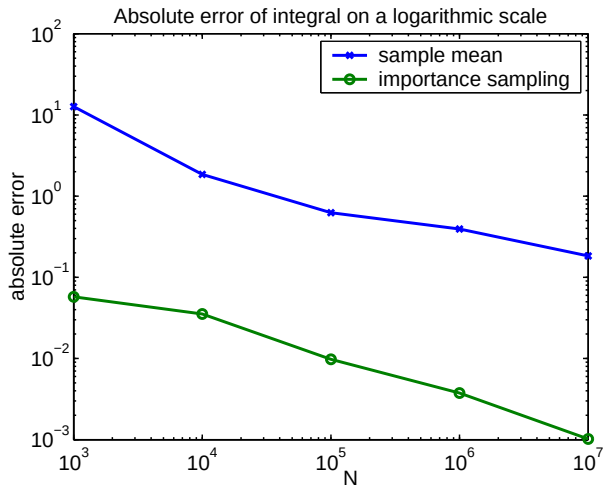


Figure 10: Absolute error.

3.2 Control variate method

As another example of a more "intelligent" Monte Carlo technique, we discuss the so-called control variate method. In this method, also known as *"variate reduction"*, we use the fact that the regular MC integration (e.g. the sample mean method) works well if the function does not vary much over the area of the integral. Since most functions are not very flat, improved methods are needed.

We already discussed the importance sampling method where the idea was to use nonuniformly distributed points which mostly fall into regions that make the largest contribution to the integral. In the control variate method, we try to replace the integral over the original function $f(x)$ by an integral over a slowly-varying function.

This can be done as follows,

$$\begin{aligned} I &= \int_a^b f(x)dx = \int_a^b [f(x)-g(x)]dx + \int_a^b g(x)dx \\ &= \int_a^b [f(x)-g(x)]dx + I_0 \end{aligned}$$

where $g(x)$ is some function relatively similar to $f(x)$ so that $f(x)-g(x)$ is slowly varying in $[a,b]$, and the integral I_0 of $g(x)$ can be calculated easily.

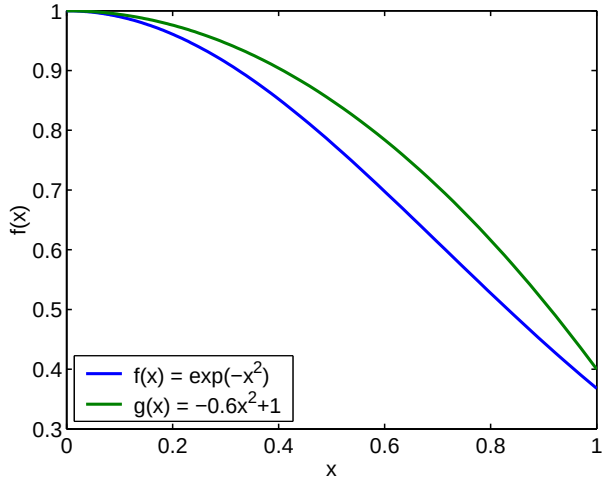
An improved estimate of the original integral can now be computed from

$$I_{est} = \frac{1}{N}\sum_{i=0}^{N}[f(x_i)-g(x_i)] + I_0$$

Example

The task is to estimate the value of the integral of $f(x) = e^{-x^2}$ from 0 to 1. By looking at the graph of $f(x)$, we note that the function $g(x) = -0.6x^2 + 1$ is similar to f in the range $0 \leq x \leq 1$. Thus we will use this function in the control variate method.

The following figure shows the graphs of the functions $f(x)$ and $g(x)$:



The integral of $g(x)$ is given by

$$I_0 = \int_0^1 g(x)dx = \int_0^1 (-0.6x^2 + 1)dx = \frac{4}{5}$$

An estimate of the original integral is obtained from

$$I_{est} = \frac{1}{N}\sum_{i=0}^{N}[f(x_i) - g(x_i)] + I_0$$

where the points x_i are uniformly distributed in the integration area (in $[0, 1]$).

Figure 11 shows the absolute error of the estimate as a function of N for the regular sample mean method and control variate method.

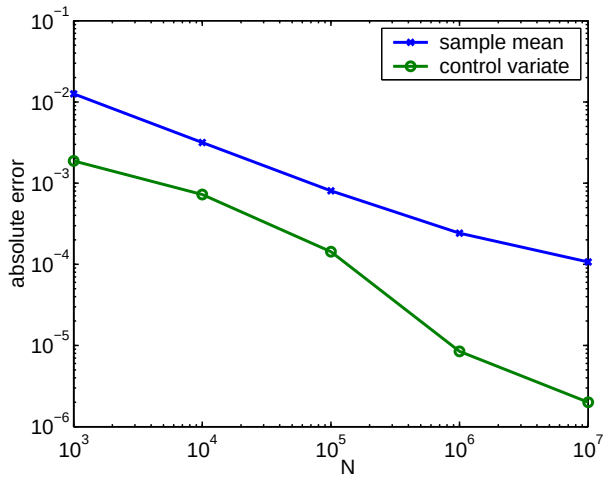


Figure 11: Absolute error of the MC estimate.

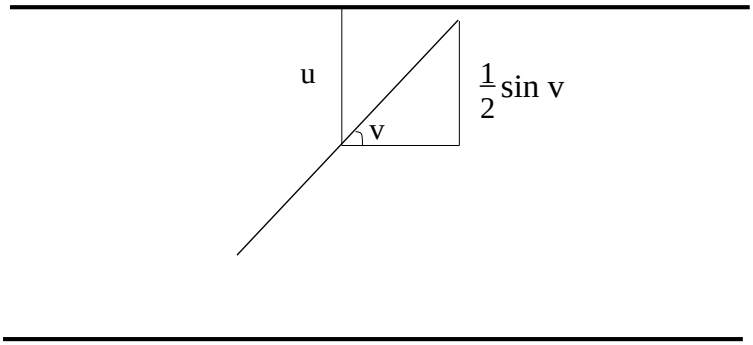
The control variate method gives a clear improvement.

4 Monte Carlo simulation

We will now illustrate another type of application of the Monte Carlo method - simulation. We begin here by simple examples - a description of more advanced techniques is given in the next chapter.

4.1 Buffon's needle problem

The next example is a very old problem known as the Buffon's needle problem. Imagine that a needle of unit length is dropped between two parallel lines that are 1 unit apart. What is the probability that the needle intersects one of the lines?



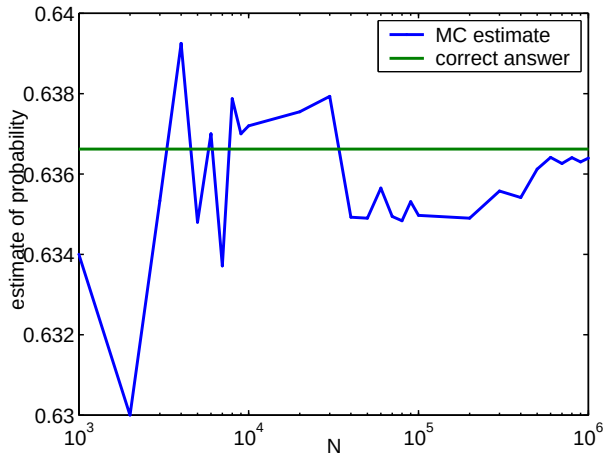
Assume that the center of the needle lands at a random point between the two lines. Denote the distance from the nearest line by u. The angular orientation of the needle is another random variable. Denote the angle measured from the axis parallel to the two lines by v. Now we can say that the needle intersects the two lines if $u \leq \frac{1}{2}\sin v$.

In practice, we select u from a uniform random distribution on the interval $[0, 1/2]$ and v from a uniform random distribution on the interval $[0, \pi/2]$. We perform the experiment N times and determine how many times $2u \leq \sin v$. Thus we can use a random variable $w = 2u$ which is distributed uniformly in $[0, 1]$.

The Buffon’s needle problem is implemented in the following C code:

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

#define PI 3.141592654

/* Combination RNG in a separate file */
void   init_random_generator(long iseed);
double random_generator();

int main(void)
{
  long int iseed=27489;
  int i, n, m=0;
  double w, v, prob;

  /* Initialize random number generator */
  init_random_generator(iseed);

  n = 1000000; /* Number of throws */

  /* Throw n needles */
  for(i=1; i<=n; i++) {

    w = random_generator();
    v = 0.5*PI*random_generator();

    /* Check if needle touches a line */
    if(w <= sin(v)) m++;

    /* Output results every 1000 throws */
    if(i%100000==0)
      printf("%d  %lf  %lf\n",
              i,(double)m/(double)i,2.0/PI);
  }
}
```

The following figure shows the probability calculated by the MC simulation as a function of N. The horizontal line is the correct answer.



At first the fluctuations are large but the result converges toward the correct answer as N becomes larger. If the correct result was not known, we could analyze the errors by doing multiple independent runs for a given number N. The subject of statistical errors is discussed in more detail shortly.

4.2 Game of cards

In this example, we want to simulate the game of poker and do an experiment to estimate the probability of getting three of a kind. We randomly draw a hand of five cards and repeat this N times. From these N hands we calculate the number of those hands, M, which contain three of a kind.

In order to implement this experiment, we need arrays for the card pack (containing 52 cards, numbered from 1 to 13, no jokers) and the hand (five cards). It is also helpful to use an extra array which indicates whether a given card has been drawn already (the hand cannot contain the same card twice). Then for each hand, we have to go through the cards and check if three of the five cards have the same value.

Typical hands from the output:

```
9 11 11 9 4
12 13 5 4 8
6 7 11 5 9
7 1 3 5 10
9 3 2 2 7
7 7 8 5 11
8 11 3 12 8
11 7 12 13 12
9 1 11 7 13
9 2 5 4 4
7 2 2 8 3
2 7 7 12 11
```

Running the code and dealing 5000 hands, we get $m = 121$ hands which contain three of a kind. This gives an estimate of 0.024 for the probability of getting three of a kind. The correct answer is 0.029 (within three decimals).

The following is one possible implementation of the program:

```
int main(void)
{
  long int iseed=27489;
  int i, j, k, x, n, m=0;
  int pack[52], drawn[52], hand[5], num;

  /* Initialize random number generator */
  init_random_generator(iseed);

  /* Set the values of the cards */
  j=1;
  for(i=0; i<52; i++) {
    pack[i]=j;  /* value = 1-13 */
    j++;
    if(j==14) j=1;
  }

  n = 5000; /* Number of hands */

  /* Deal n hands */
  for(i=0; i<n; i++) {

    /* Initilize help pack */
    for(j=0; j<52; j++) drawn[j]=0;

    /* Draw five cards */
    for(j=0; j<5; j++) {
      /* Uniform random number in [0,51]*/
      x = 52*random_generator();
      while(drawn[x]==1)  x = 52*random_generator();
      hand[j] = pack[x]; drawn[x]=1;
    }

    /* Look for a three of a kind */
    for(j=0; j<3; j++) {
      num = 1;
      for(k=j+1; k<5; k++)
        if(hand[j] == hand[k]) num++;
      if(num>=3) { m++; break; }
    }
  }
  fprintf(stdout,"\n n=%d  m=%d   p(3 of kind) = %10.8lf\n",
                 n,m,(double)m/(double)n);
}
```

4.3 Calculation of errors

Statistical errors are inherent to the Monte Carlo method since it is a stochastic method which utilizes random numbers. For this reason, it is extremely important to have the proper tools for analyzing the errors. Otherwise we cannot say much about the quality of an obtained answer.

Statistical errors arise from random changes in the system from one measurement to another. They can be estimated by taking many independent measurements of the quantity of interest and calculating the spread of the values. We can estimate the true value by taking the mean of several independent measurements, and the error on that estimate is simply the error on the mean.

Denote the quantity of interest by A. A single Monte Carlo run produces a value A_i, which is considered as the result of a single 'measurement'. Taking m such independent measurements, we can calculate the **mean** of the measured values from

$$\bar{A} = \frac{1}{m}\sum_{i=1}^{m} A_i$$

This is the estimate of the true value $\langle A \rangle$. The standard deviation, or error, of this estimate is $\sigma = \sigma_p/\sqrt{m}$, where $\sigma_p^2 = \langle A^2 \rangle - \langle A \rangle^2$ is the inherent standard deviation or error. Denote deviation $\delta A_i = A_i - \bar{A}$. Trivially, $\delta\bar{A}_i = 0$, $\langle \delta A \rangle = 0$. Hence, $\delta\bar{A}^2 = 1/m \sum_{i=1}^{m} (\delta A_i)^2 = \bar{A^2} - (\bar{A})^2$.

$\langle \delta A^2 \rangle = \sigma_p^2 -$ error in the estimate $= \sigma_p^2 - \sigma_p^2/m$. The estimate of the **standard deviation** on the mean of the m independent measurements is hence given by

$$\sigma = \frac{\sigma_p}{\sqrt{m}} == \sqrt{\frac{\frac{1}{m}\sum_{i=1}^{m}(A_i - \langle A \rangle)^2}{m-1}} = \sqrt{\frac{1}{m-1}(\langle A^2 \rangle - \langle A \rangle^2)}$$

This an estimate of the **error**.

The above expression for error is based on the assumption that the m measurements are statistically independent. This is valid in the case of simple sampling, but in more advanced MC techniques, successive samples are often correlated which means that the above expression underestimates the error.

4.4 Simulation of radioactive decay

One of the simplest examples of a physical process for which the Monte Carlo method can be applied is the study of radioactive decay. Here we begin with a sample of N nuclei which decay at a rate λ s^{-1}. The physics of the situation specifies that the rate of decay is given by

$$\frac{dN}{dt} = -\lambda N$$

where the nuclei which decay during the time dt can be chosen randomly. Note that this is an example of a physical process which is *random* by nature. The Monte Carlo method is especially well suited for studies of such processes.

The time dependence of the number of undecayed nuclei specifies that the rate of decay is given by

$$N = N_0 e^{-\lambda N}$$

where N_0 is the initial number of nuclei and λ is related to the 'half-life' of the system.

In the simplest approach, the positions of the nuclei play no role and only the number of undecayed nuclei is monitored. Time is divided into *discrete* intervals, and each undecayed nucleus is tested for decay. Time is then incremented by one unit and the process is repeated so that the number of undecayed nuclei can be determined as a function of time.

The time discretization must be done intelligently so that a reasonable number of decays occur in each time step. Otherwise the simulation requires too much cpu time to be effective. On the other hand, if the time step is chosen to be too large, then time resolution is lost because too many decays occur during each time step.

The entire process can be repeated many times to obtain a series of independent 'measurements' and the mean value of N as well as an error estimate may be determined for each value of time. Note that in this case the measurements are uncorrelated since we begin a new simulation with a different random number sequence each time.

The extension of this approach to multiple decay paths is straightforward.

Example

Given a sample of 10000 radioactive nuclei each of which decays at rate p per second, what is the half-life of the sample if $p = 0.2$?

The most accurate way to solve this problem using the Monte Carlo approach, is to take many such samples and simply determine the time which it takes for each sample to decay to half of its original size. The suitable length of the time step dt can be obtained by doing a few short test runs with different values of the time step. A suitable value is $dt = 0.01$ s. The decay probability per time step is then $p = 0.002$ s^{-1}. In each individual measurement, the half-time is estimated by the elapsed time when the number of undecayed nuclei N is less or equal to $N_0/2$.

The following segment of C code performs the *m* measurements:

```
  N0 = 10000;  /* Size of sample */
  dt = 0.01;   /* time step = 0.1 sec */
  p = 0.002;   /* Decay probability per time step */
  m = 100;     /* number of independent measurements */

  /* Loop over independent measurements */
  for(k=0; k<m; k++) {

    /* Begin a new simulation */
    N=N0; time = dt;

    /* Loop over time steps*/
    for(step=0; step<MAXSTEPS; step++) {
      /* Number of undecayed nuclei */
      n = N;
      /* Loop over undecayed nuclei */
      for(i=0; i<n; i++) {
        /* Test for decay */
        r = random_generator();
        if(r < p) N--;  /* Decay event */
      }
      /* Check for half-life */
      if(2*N <= N0) break;
      /* Increment time */
      time = time+dt;
    }

    /* Update averages */
    thalf  += time;
    thalf2 += time*time;
}
```

After the averages $\langle t_{1/2} \rangle$ and $\langle t_{1/2}^2 \rangle$ have been calculated from the m measurements, we can analyze the error on the mean value by calculating the standard deviation:

$$\sigma = \sqrt{\frac{1}{m-1}(\langle t_{1/2}^2 \rangle - \langle t_{1/2} \rangle^2)}$$

This is done in the following segment of code:

```
 /* Calculate the final estimate */
  thalf = thalf/(double)m;
  thalf2 = thalf2/(double)m;

  /* Standard deviation (error) */
  std = sqrt((thalf2-thalf*thalf)/(double)(m-1));
```

Typical values of N obtained from 5 independent measurements are:

```
k=0  N0 = 10000  N =  4999  time = 3.49000000
k=1  N0 = 10000  N =  4999  time = 3.41000000
k=2  N0 = 10000  N =  4986  time = 3.54000000
k=3  N0 = 10000  N =  4988  time = 3.41000000
k=4  N0 = 10000  N =  4996  time = 3.52000000
```

This gives us the following final estimates:

```
m=5     thalf = 3.47400000  error = 0.02731300
```

The accuracy can be increased by taking more samples:

```
m=100  thalf = 3.46630000  error = 0.00485685
```

Thus the MC simulation with $m = 100$ gives the answer
$t_{1/2,MC} = 3.466 \pm 0.005$.
The result is within the given error limits of the correct answer
which is is $t_{1/2} = \ln 0.5/(-0.2) \approx 3.4657359$.