Department of Physics
University of Dhaka
Computational Physics
Random Number

- The numbers that generates in a sequence which have no relation with each other. Random numbers are useful for a variety of purposes, such as generating data encryption keys, simulating and modeling complex phenomena and for selecting random samples from larger data sets. They have also been used aesthetically, for example in literature and music, and are of course ever popular for games and gambling. When discussing single numbers, a random number is one that is drawn from a set of possible values, each of which is equally probable, i.e., a uniform distribution. When discussing a sequence of random numbers, each number drawn must be statistically independent of the others. **We do not get actually random but a Pseudo Random Number**.

- **Pseudo Random Number Generator:** As the word pseudo suggests, pseudo-random numbers are not random in the way you might expect, at least not if you're used to dice rolls or lottery tickets. Essentially, PRNGs are algorithms that use mathematical formulae or simply precalculated tables to produce sequences of numbers that appear random. A good example of a PRNG is the linear congruential method. A good deal of research has gone into pseudo-random number theory, and modern algorithms for generating pseudo-random numbers are so good that the numbers look exactly like they were really random. It usually use the following algorithm $X_{n+1} = (aX_n + b)\%m$,
  Where X is a sequence of random value,
  m is a positive number,
  a is a multiplier $0 < a < m$,
  c is the increment $0 \leq c < m$,
  X_0 is the seed $0 \leq X_0 < m$
  For example if one use a=7, b=0, m=11, and X_0=1 he/she is suppose to get the numbers {7, 5, 2, 3, 10, 4, 6, 9, 8, 1}, Once he/she get 1 which is same as our seed, the number sequence will repeat. Since all integers are less than m the sequence must repeat after at least m-1 iterations, i.e. the maximal period is m -1. ($x_0 = 0$ is a fixed point and cannot be used.)

  - ⋆ The numbers follow a sequence.
  - ⋆ It follow same sequence if started from the same seed.
  - ⋆ By changing the seed one can change the randomness.
  - ⋆ No correlations
  - ⋆ Long periods
  - ⋆ Follow well-defined distribution
  - ⋆ Fast implementation
  - ⋆ Reproductibility

  We can generate a pseudo-random number in the range from 0.0 to 32,767 using —bf rand() function from < *cstdlib* > library. The maximum value is library-dependent, but is guaranteed to be at least 32767 on any standard library implementation. We can check it from RAND_MAX.

```
#include< iostream >
#include< cstdlib >
using namespace std;
int main()  {
int x;
x=rand();
cout<<x;
x=rand();
cout<<x;
cout<<RAND_MAX;   //To check the maximum value of the random number.
return 0;}
```

- We can set the range of generated numbers using % (modulus) operator by specifying a maximum value. For instance, to generate a whole number within the range of **1 to 100:**

```
#include < iostream >
#include < cstdlib >
using namespace std;

int main()
{
int i = 0;
while(i++ < 10) {
int r = (rand() % 100) + 1;
cout << r <<" "; }
return 0;
}
```

- One can also generate random number in a certain range, for example if one wants to generate number between -9 to +9 he may write the following code:

```
#include < iostream >
#include < cstdlib >
using namespace std;

int main()
{
int i = 0;
while(i++ < 10) {
int r = (rand() % 19) + (-9);
cout << r <<" "; }
return 0;
}
```

There are 19 distinct integers between -9 and +9 (including both), that is why you need to get the modulous of 19)

- To increase the randomness of the generated numbers one shoul use **SEEDING**, is will increase the randomness of the gnerated numbers.

```
#include < iostream >
#include < cstdlib >
using namespace std;

int main()
{
int i = 0, seed=12345;
srand(seed);
while(i++ < 10) {
int r = (rand() % 100) + 1;
cout << r <<" "; }
return 0;
}
```

By changing the seed to a different value you can get different sequence of number.

- To increase the randomness further, people use the computer time as seed. For this one need to use the library < *ctime* >, Please see the code below:

```
#include < iostream >
#include < cmath >
#include < cstdlib >
#include < ctime >
using namespace std;

int main()
{
int i = 0;
srand(time(NULL));
while(i++ < 10) {
int r = (rand() % 100) + 1;
cout << r << " ";
}
return 0;
}
```

- **Monte Carlo Integration Method**

  - ⋆ **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.

⋆ **Monte Carlo integration - simple sampling (Hit or Miss method):** 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 $f(x)$ within the interval [a,b], i. e $I = \int_a^b f(x)dx$ One-dimensional integrals can be effectively calculated using discrete approximations such as the trapezoidal rule or Simpsons rule. 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). 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 **fmax** where **fmax** $>$ **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 **N0** which fall under the curve of f(x); i.e.

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

where $R = (b-a) * fmax$ be the total 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 $x = a + s_1 * (b-a)$
$y = s_2 * fmax$
as the x and y coordinates of the point.

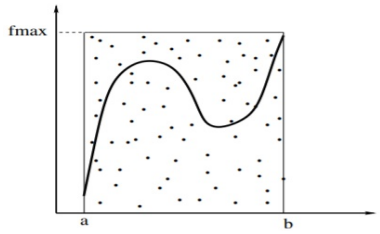

Figure 1: The hit and miss method, Points are generated randomly inside a rectangular area which bounds the function f(x) in the interval [a,b]. The number of points under the curve is calculated to obtain an estimate of the integral.

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 different random number sequences. Comparing the values gives an idea of the precision of the calculation.

- **Calculation of PI:** A simple example of the use of random numbers is the calculation of π. If we put points within a square, with center at the origin and sides having length two times unity at random, then the number of point within a unit circle with center at the origin to that within the square, will be equal to the ratio of the areas of the unit circle to that of the square. i. e. $\pi/4$. 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

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

Thus

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

The following program calculates π using this algorithm.

#include < *iostream* >
#include < *cmath* >
#include < *cstdlib* >
#include< *ctime* >
using namespace std;

double getmePi(int N){ int i=0, seed, count=0;
double x=0.0, y=0.0;
seed= time(NULL);
srand(seed); for(i=1;i<=N;i++) {
x=(double) rand()/(double) RAND_MAX;
y=(double) rand()/(double) RAND_MAX;
RAND_MAX is the maximum random integer number created by rand ().
if(sqrt(x*x+y*y)<=1.0) count++;
}
return 4.0*(double) count/(double)N;

```
}
int main(){
int i=0; double val=0.0;
for(i=1000;i<=1000000;i*=10){
val=getmePi(i);
cout<<val<<endl;}
return 0;}
```

- Calculate the integral

$$I = \int_{\pi}^{\pi} \frac{(1 + cosx)sin|2x|}{1 + |sin(2x)|} dx$$

```
#include < iostream >
#include < cmath >
#include < cstdlib >
#include< ctime >
using namespace std;


double func(double x){
return (1+cos(x))*sin(abs(2*x))/(1+abs(sin(2*x)));
}
double MC1D(double a, double b, double c, double d, int N){
int i;
double x, y, area;
int S=0;
for(i=1;i¡=N;i++){
x = a + (b - a) * (rand()/(double)RAND_MAX);
y = c + (d - c) * (rand()/(double)RAND_MAX);
if(y <= func(x))S++;
area=((double)S/(double)N)*(b-a)*(d-c);}
return area;
}


int main() {
srand(time(NULL));
cout << MC1D(-M_PI, M_PI, -0.3, 0.9, 10000) << endl;
return 0;}
```

- **Running various functions at the same time:** If you need to integrate various different functions at the same tieme with the same method, for example with mote carlo method, you can proceed as follows:

  - ⋆ define all your functions with their return types one by one with a common name and some extension. For Example,
    double func_1(double x){ return .....}
    double func_2(double x){ return .....}
    and so on .....

- ⋆ Define the integration function (monte carlo function here) with the guest function(function to be integrated) as the first argument with the common name. Example:
  double MC_1D(**double func(double x),** double a, double b, double c, double d, int N).
- ⋆ Call the integration function in the main function, with the **full name** of the function to be integrated as the first argumnet. For example: Now this time if you want to get the func_1 to be integrated,
  MC_1D(func_1, a, b, c, d, N).

- See the example below: Here we are calculating

$$I = \int_{\pi}^{\pi} \frac{(1 + cosx)sin|2x|}{1 + |sin(2x)|} dx$$

and

$$\int_{0}^{\pi/2} \frac{1}{1 + sin^2(x)} dx$$

#include < *iostream* >
#include < *cmath* >
#include < *cstdlib* >
#include< *ctime* >
using namespace std;

double func_1(double x){
return (1+cos(x))*sin(abs(2*x))/(1+abs(sin(2*x)));
}
double func_2(double x){
return 1/(1+sin(x)*sin(x));
}
double MC1D(**double func(double x),** double a, double b, double c, double d, int N){
int i;
double x, y, area;
int S=0;
for(i=1;i¡=N;i++){
$x = a + (b - a) * (rand()/(double)RAND_MAX)$;
$y = c + (d - c) * (rand()/(double)RAND_MAX)$;
if(y <= func(x))S++;
area=((double)S/(double)N)*(b-a)*(d-c);}
return area;
}

int main() {
srand(time(NULL));
$cout << MC1D(\textbf{func\_1}, -M\_PI, M\_PI, -0.3, 0.9, 10000) << endl$;
$cout << MC1D(\textbf{func\_2}, 0, M\_PI/2, -0.3, 0.9, 10000) << endl$;
return 0;}