MONTE CARLO METHODS

Introduction

The deterministic systems are described by some mathematical rule. But some systems are not deterministic known as random or stochastic.

Monte Carlo refers to any procedure that makes use of random numbers and it is opposed to deterministic algorithms.

There are lots of applications:

- Data analysis
- Simulation of physics events, which are based on random processes/probabilities
- Detector design, optimization and simulation

- Mathematics, Numerical Analysis
- Physics, Simulation of Natural Phenomena
- Engineering, Simulation of Experimental Apparatus
- Biology, Cell Simulations
- Statistics, Distribution Functions
- Economy, Modeling Stock Exchange

Random Number

A Random Number is a number chosen as if by chance from some specified distribution. In a uniform distribution of Random Numbers in the range [0,1], every number have the same chance of turning up.

As an example:
When you throw a dice the numbers incoming are uniformly distributed over the range between 1 and 6. Because every number have the same chance of turning up. Try!

Random Number Generators

In principle, the best way to obtain a series of random numbers

\{x_1, x_2, x_3, ... x_n\}

is to use some process in nature:

- throw a coin or dice
- take the decay time of radioactive nuclei
- take the soccer results of last sunday
- Lotto 6-49
This of course is not very efficient. Therefore computer algorithms have been developed. These algorithms are known as: Pseudo-Random Number Generators.

A Random Number Generator is a computer sub-program that produce sequence of random numbers. These should really be termed pseudo-random numbers, because for each run, you will obtain same sequence of random numbers for the same seed.

In general, these functions uses Linear Conguential Method (Lehmer, 1948) using 32-bit integers have a period of at most $2^{31} \sim 10^9$. This many random numbers can be generated in seconds on a modern workstation. The method is employed by an equation of the form:

$$x_{i+1} = (a \cdot x_i + b) \mod m$$

where mod means modulo. Constants a, b and m are chosen carefully such that the sequence of numbers becomes chaotic and evenly distributed.

Rules:

- First initial number, $x_0$, called seed, is selected
- $m > x_0$, $a, b \geq 0$
- The range of values is 0 to m (divide by m to convert to 0. to 1.). The period of this generator is $m-1$. So m should be as large as possible since the period can never be longer than m.

A popular random number generator called RANDU was distributed by IBM in 1960 with the algorithm:

$$x_{i+1} = (69069 \cdot x_i) \mod 2^{31}-1$$

and then Park and Miller proposed a minimal standard:

$$x_{i+1} = (16807 \cdot x_i) \mod 2^{31}-1$$

Here is the C++ function that uses Park-Miller algorithm:

```cpp
// Park-Miller algorithm
// Returns a random real number in the range [0,1]
double random_pm(int &seed) {
    const int   im = 2147483647, ia = 16807;
    const int   iq = 127773,     ir = 2836;
    const double m = 128.0/im;
    int    k;
    double r;
    k = seed / iq;
    seed = ia*(seed - k*iq) - ir*k;
    if(seed < 0) seed += im;
    r = m * (seed/128);
    return r;
}
```
### Usage of random_pm() function in a main program:

<table>
<thead>
<tr>
<th>Code</th>
<th>Output</th>
</tr>
</thead>
</table>
| #include <iostream>  
using namespace std;  

double random_pm(int &);  

int main(){  
int Xseed, i;  
double r;  
Xseed = 123456789;  

for(i=1; i<=10; i++){  
r = random_pm(Xseed);  
cout << r << endl;  
}  
} // main  
// Park-Miller algorithm  
// returns a random real number in the range [0,1]  
//----------------------------------------------------------------------  

double random_pm(int &seed)  
{  
const int   im = 2147483647, ia = 16807;  
const int   iq = 127773,     ir = 2836;  
const double m = 128.0/im;  
int    k;  
double r;  
k = seed / iq;  
seed = ia*(seed - k*iq) - ir*k;  
if(seed < 0) seed += im;  
r = m * (seed/128);  
return r;  
}

Today, all computer languages contain a mechanism for producing a sequence of random numbers which are uniformly distributed in [0,1]. Here is the example built-in functions to assign a random number in the range [0,1] to a floating-point variable x:

**Fortran**  
CALL RANDOM_NUMBER(x)

**C/C++**  
x = rand() / (RAND_MAX+1.0);

**MATLAB**  
x = rand;
Basic Simulations

<table>
<thead>
<tr>
<th>Tossing a coin</th>
<th>Output</th>
</tr>
</thead>
</table>
| `#include <iostream>`
`#include <cstdlib>`
using namespace std;
int main(){
    int m = 0; // number of heads
    int n = 5000; // number of experiments
    for(int i=1; i<=n; i++){
        double x = rand()/(RAND_MAX+1.0);
        if (x < 0.5) // head
            m++;
    }
    double p = double(m)/n;
    cout << n << m << p << endl;
} |
| 5000 2522 0.5044 |

<table>
<thead>
<tr>
<th>Tossing two dices</th>
<th>Output</th>
</tr>
</thead>
</table>
| `#include <iostream>`
`#include <cstdlib>`
using namespace std;
int main(){
    int m = 0; // number of 6
    int n = 5000; // number of experiments
    for(int i=1; i<=n; i++){
        int x = 1 + int(6*rand()/(RAND_MAX+1.0));
        if (x == 6) // six
            m++;
    }
    double p = double(m)/n;
    cout << n << m << p << endl;
} |
| 5000 840 0.168 |

MC Integration

In Monte Carlo integration, numerical integration of a function is performed by making use of random numbers.

- one-dimensional integrals is not as effective as other numerical integration methods (such as Trapezoidal or Simpson’s method)
- Though for high-dimensional integrals the MC method can be more efficient

Algorithm:

1. Enclose the function in a box area $y_{\text{max}}$.
2. Uniformly populate the box with $N$ random points:
   - Generate two random numbers $R_1$, $R_2$, a random point in the box is then
   - $x = a + (b-a)*R_1$
   - $y = y_{\text{max}}*R_2$
3. Count the number of points $M$ that lie below the curve $f(x)$.
4. The integral is then estimated from:
   - $I/A = M/N$ and so
   - $I = (M/N)*A$ and so
   - $I = (b-a)*y_{\text{max}} M/N$
EXAMPLE: Computing $\pi$

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

int main(){
    int i,j,Tosses,Hits;
    double x,y,Pi,error;
    for(i=1;i<=9;i++){
        Tosses = pow(10.0,i);
        Hits   = 0;
        for(j=1; j<=Tosses; j++){
            x = -1 + 2*double(rand())/RAND_MAX;
            y = -1 + 2*double(rand())/RAND_MAX;
            if( (x*x + y*y)<1.0 ) Hits++;
        }
        Pi = 4.0*Hits/Tosses;
        error = fabs(M_PI-Pi);
        cout << Tosses << Pi << error << endl;
    }
}
// main
```

<table>
<thead>
<tr>
<th>Computing $\pi$</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>#include &lt;iostream&gt;</code></td>
<td>10  3.200000  0.058407</td>
</tr>
<tr>
<td><code>#include &lt;cstdlib&gt;</code></td>
<td>100 3.120000  0.021593</td>
</tr>
<tr>
<td>using namespace std;</td>
<td>1,000 3.156000  0.014407</td>
</tr>
<tr>
<td>int main(){</td>
<td>10,000 3.166800  0.025207</td>
</tr>
<tr>
<td>int i,j,Tosses,Hits;</td>
<td>100,000 3.138880  0.002713</td>
</tr>
<tr>
<td>double x,y,Pi,error;</td>
<td>1,000,000 3.141920  0.000327</td>
</tr>
<tr>
<td>for(i=1;i&lt;=9;i++){</td>
<td>10,000,000 3.140899  0.000694</td>
</tr>
<tr>
<td>Tosses = pow(10.0,i);</td>
<td>100,000,000 3.141710  0.000117</td>
</tr>
<tr>
<td>Hits = 0;</td>
<td>1,000,000,000 3.141604  0.000012</td>
</tr>
<tr>
<td>} // main</td>
<td></td>
</tr>
</tbody>
</table>
EXAMPLE: Computing area under the curve: \( y = 10 \sin(x) \) between 0 and \( \pi \).

```cpp
#include <iostream>
#include <cstdlib>
#include <cmath>

using namespace std;

double f(double x)
{
    return 10*sin(x);
}

int main()
{
    int i, j, m, n;
    double area, r1, r2;
    double x, y, a, b, y_max, error;
    y_max = 10.0;  a = 0.0;  b = M_PI;

    for(i=1; i<=9; i++) {
        n = pow(10.0,i);
        m = 0;

        for(j=1; j<=n; j++) {
            r1 = (double) rand()/RAND_MAX;
            r2 = (double) rand()/RAND_MAX;
            x = a + (b-a)*r1;
            y = y_max*r2;
            if( y < f(x) ) m++;
        }

        area  = (b-a)*y_max*m/n;
        error = fabs(20.0 - area);
        cout << n << area << error << endl;
    }

    return 0;
}
```

<table>
<thead>
<tr>
<th>Computing integral</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>#include &lt;iostream&gt;</td>
<td>10   18.84956  1.1504420</td>
</tr>
<tr>
<td>#include &lt;cstdlib&gt;</td>
<td>100  20.42035  0.4203548</td>
</tr>
<tr>
<td>#include &lt;cmath&gt;</td>
<td>10,000 19.94912  0.0508842</td>
</tr>
<tr>
<td>using namespace std;</td>
<td>100,000 20.08106  0.0810623</td>
</tr>
<tr>
<td></td>
<td>1,000,000 19.94000  0.0599956</td>
</tr>
<tr>
<td>double f(double x){</td>
<td>1,000,000 20.00874  0.0087432</td>
</tr>
<tr>
<td>return 10*sin(x);</td>
<td>10,000,000 19.99852  0.0014839</td>
</tr>
<tr>
<td>}</td>
<td>100,000,000 20.00074  0.0007362</td>
</tr>
<tr>
<td></td>
<td>1,000,000,000 20.00051  0.0005044</td>
</tr>
</tbody>
</table>