A.1 Introduction to Monte Carlo Method

Monte Carlo (MC) methods are stochastic techniques which are based on the use of random numbers and probability statistics to investigate problems. MC methods can be used in everything from economics to nuclear physics to regulating the flow of traffic. Of course the way they are applied varies widely from field to field, and there are dozens of subsets of MC even within chemistry. But, strictly speaking, to call something a "Monte Carlo" experiment, there is a need to use random numbers to examine some problem. That MC methods are used to model physical problems to examine more complex systems then solving equations which describe the interactions between two atoms is fairly simple; but solving the same equations for hundreds or thousands of atoms is impossible. With MC methods, a large system can be sampled in a number of random configurations, and that data can be used to describe the system as a whole. "Hit and miss" integration is the simplest type of MC method to understand, and it is the type of experiment used to determine the various type of distributions. The MC experiment which calculates the value of pi based on a "hit and miss" integration is discussed below.

A.2 Monte Carlo Calculation of Pi

The figure as shown below is simply a unit circle circumscribed by a square. The problem could be examined in terms of the full circle and square, but it's easier to examine just one quadrant of the circle, as in the figure below. If someone is a very poor dart player, it is easy to imagine throwing darts randomly at the figure, and it should be
apparent that of the total number of darts that hit within the square, the number of darts that hit the shaded part (circle quadrant) is proportional to the area of that part. In other words, by geometry, it’s easy to show that if each dart thrown lands somewhere inside the square, the ratio of "hits" (in the shaded area) to "throws" will be one-fourth the value of pi. If someone actually doing this experiment then soon realizes that it takes a very large number of throws to get a decent value of pi...well over 1,000. To make things easy, computers should be used to generate random numbers.

![Figure A.1. Circle inscribing the square.](image)

\[
\frac{\text{Number of darts hitting shaded area}}{\text{Number of darts hitting the square}} = \frac{1}{4} \pi r^2
\]

\[
\pi = 4 \frac{\text{Number of darts hitting shaded area}}{\text{Number of darts hitting the square}}
\]

Let the radius of the circle be 1.0, for each throw, it can generate two random numbers, an x and a y coordinate, whose distance from the origin (0, 0) can be calculated using the Pythagorean theorem. If the distance from the origin is less than or equal to 1.0, it is within the shaded area and counts as a hit. This should be repeated thousands (or millions) of times, and one will wind up with an estimate of the value of pi...
pi. How good it is depends on how many iterations (throws) are done, and to a lesser extent on the quality of the random number generator.

Matlab Code

********************************************************************************
1. hits=0;
2. m=100000;
3. for n= 1: m
4.    x=rand(1);
5.    y=rand(1);
6.    dist=sqrt(x^2 +y^2);
7.    if dist<=1.0
8.        hits=hits+1.0;
9.    end
10. end
11. value_pi=4*hits/n
********************************************************************************

Table A.1. Value of Pi by using different values of m.

<table>
<thead>
<tr>
<th>m</th>
<th>( \pi )</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>3.6</td>
</tr>
<tr>
<td>100</td>
<td>3</td>
</tr>
<tr>
<td>1000</td>
<td>3.12</td>
</tr>
<tr>
<td>10000</td>
<td>3.1268</td>
</tr>
<tr>
<td>100000</td>
<td>3.1504</td>
</tr>
<tr>
<td>1000000</td>
<td>3.1451</td>
</tr>
<tr>
<td>2000000</td>
<td>3.1417</td>
</tr>
</tbody>
</table>

This is a simple code for obtaining the values of Pi using Matlab. When this code is run, the value of pi comes out to be 3.1451. It is worth while noting that the maximum value of n which is equal to m significantly affects the result of the
simulation as shown in Table A.1. However, it can also be noted from the table as we reach to a value of \( n = 10^6 \) the results are very accurate.

### A.3 Random Numbers

At the heart of any Monte Carlo method is a random number generator: a procedure that produces an infinite stream \( U_1, U_2, U_3, \ldots \) of random variables that are independent and identically distributed according to some probability distribution \( \text{Dist} \). When this distribution is the uniform distribution on the interval \( (0, 1) \) (that is, \( \text{Dist} = U(0, 1) \)), the generator is said to be a uniform random number generator. Most computer languages already contain a built-in uniform random number generator. The user is typically requested only to input an initial number, called the seed, and upon invocation the random number generator produces a sequence of independent uniform random variables on the interval \( (0, 1) \). In MATLAB, for example, this is provided by the \text{rand} function.

The concept of an infinite identically distributed sequence of random variables is a mathematical abstraction that may be impossible to implement on a computer. The best one can hope to achieve in practice is to produce a sequence of random numbers with statistical properties that are indistinguishable from those of a true sequence of identically distributed random variables. Although physical generation methods based on universal background radiation or quantum mechanics seem to offer a stable source of such true randomness, the vast majority of current random number generators are based on simple algorithms that can be easily implemented on a computer. Such algorithms can usually be represented as a tuple \( (S, f, \mu, U, g) \), where

- \( S \) is a finite set of states.
A random number generator has the following structure:

Algorithm (Generic Random Number Generator)
1. Initialize: Draw the seed \( S_0 \) from the distribution \( \mu \) on \( S \). Set \( t = 1 \).
2. Transition: Set \( S_t = f(S_{t-1}) \).
3. Output: Set \( U_t = g(S_t) \).
4. Repeat: Set \( t = t + 1 \) and return to Step 2.

The algorithm produces a sequence \( U_1, U_2, U_3, \ldots \) of pseudorandom numbers and they are called as random numbers. Starting from a certain seed, the sequence of states (and hence of random numbers) must repeat itself, because the state space is finite. The smallest number of steps taken before entering a previously visited state is called the period length of the random number generator.

**A.4 Bernoulli Distribution**

The probability distribution function of the Bernoulli distribution is given by \( f(x; p) = p^x (1 - p)^{1-x}, \ x \in \{0, 1\} \), where \( p \in [0, 1] \) is the success parameter. The distribution is represented as \( \text{Ber}(p) \). The Bernoulli distribution is used to describe experiments with only two outcomes: 1 (success) or 0 (failure). Such an experiment is called a Bernoulli trial. A sequence of identically distributed Bernoulli random variables, \( X_1, X_2, \ldots \) identically distributed \( \text{Ber}(p) \), is called a Bernoulli process. This
process is used to model a random experiment. The inverse-transform method leads to the following generation algorithm.

Algorithm 3.1 (Ber(p) Generator)

1. Generate $U \approx U(0, 1)$.
2. If $U \leq p$, return $X = 1$; otherwise, return $X = 0$.

A.5 Binomial Distribution

The probability distribution function of the binomial distribution is given by

$$f(x; n, p) = P(X = x) = \binom{n}{x} p^x (1-p)^{n-x}, x=0,1,2,\ldots,n,$$

Where, $0 \leq p \leq 1$. The distribution is represented as Bin(n, p). The binomial distribution is used to describe the total number of successes in a sequence of n independent Bernoulli trials. That is, a Bin(n, p) distributed random variable $X$ can be written as the sum $X = B_1 + \cdots + B_n$ of independent Ber(p) random variables $\{B_i\}$.

A.6 Derivation of the Poisson Distribution

A Poisson distribution can sometimes be used to approximate the binomial distribution. For the approximation to be reasonable the value of $n$ needs to be large and the value of $p$ should be small:

![Figure A.2. Binomial to Poisson distribution.](image-url)
Proof:

A better way of describing $\lambda$ is as a probability per unit time that an event will occur. That is

$$dP = \lambda dt$$ \hspace{2cm} (A.1)

Where, $dP$ is the differential probability that an event will occur in the infinitesimal time interval $dt$. Of course, some care must be taken when translating a rate to a probability per unit time. For example, if $\lambda = 10/s$, it is obviously not true that the probability is 10 that an event will occur in any particular second. However, if that same rate is expressed $\lambda = 0.01/ms$, it is roughly true that the probability is 0.01 that an event will happen in any particular millisecond. The equation A.1 only becomes exact in the limit of infinitesimal $dt$. It is approximately correct for any finite $\Delta t$.

$$\Delta P = \lambda \Delta t$$ \hspace{2cm} (A.2)

to the extent that $\Delta P \ll 1$.

There are several possible derivations of the Poisson probability distribution. It is often derived as a limiting case of the binomial probability distribution. The derivation to follow relies on equation A.1 and begins by determining the probability $P(0; t)$ that there will be no events in some finite interval $t$. The first step is to break the interval from 0 to $t$ into $N$ intervals of equal length $\Delta t = t/N$ (the limit as $N \to \infty$ will be performed at the end). The probability of an event in a small enough but finite interval $\Delta t$ will be given by the equation A.2 and thus the probability of no event in this same interval is given by $1 - \lambda \Delta t$. For the full interval from 0 to $t$ to have no events, each and every one of the $N$ subintervals must have no events. Consequently, the probability of no events in a time $t$ is the product of the $N$ probabilities for no events in the subintervals.
\[ P(0: t) = (1 - \lambda \Delta t)^N \]  \hspace{1cm} \text{A.3}

Substituting \( \Delta t = t/N \)

\[ P(0: t) = \left(1 - \frac{t}{N} \right)^N \]  \hspace{1cm} \text{A.4}

Taking the limit \( N \to \infty \) and recognizing that

\[ \lim_{N \to \infty} \left(1 - \frac{x}{N}\right)^N = e^{-x} \]  \hspace{1cm} \text{A.5}

gives

\[ P(0: t) = e^{-\lambda t} \]  \hspace{1cm} \text{A.6}

Next, a relation is derived for the probability, denoted \( P(n + 1; t) \), for there to be \( n + 1 \) events in a time \( t \). It will be a recursion relation because it will be based on the probability \( P(n; t) \) of one less event. For there to be \( n + 1 \) events in \( t \), three independent events must happen in the following order (their probabilities given in parentheses).

- There must be \( n \) events up to some point \( t' \) in the interval from 0 to \( t \) (\( P(n, t') \) by definition)
- An event must occur in the infinitesimal interval from \( t' \) to \( t' + dt' \) (\( \lambda dt' \) by A.1).
- There must be no events in the interval from \( t' \) to \( t \) (\( P(0, t - t') \) by definition).

The probability of \( n + 1 \) events in the interval from 0 to \( t \) would be the product of the three probabilities above integrated over all \( t' \) from 0 to \( t \) to take into account that the last event may occur at any time in the interval. That is,

\[ P(n + 1; t) = \int_0^t P(n; t') \lambda dt' P(0; t - t') \]  \hspace{1cm} \text{A.7}

From Equation A.6 we already have \( P(0; t - t') = e^{\lambda (t-t')} \) and with the following definition
\[ P(n; t) = e^{-\lambda t} \bar{P}(n; t) \]  

A.8

Substituted, A.7 becomes (after canceling \( e^{-\lambda t} \) from both sides):

\[ \bar{P}(n+1; t) = \lambda \int_0^t \bar{P}(n; t') \, dt' \]  

A.9

From A.6 and A.8, \( \bar{P}(0; t) = 1 \) and then \( \bar{P}(1; t) \) can be found from an application of A.9

\[ \bar{P}(1; t) = \lambda \int_0^t \bar{P}(0, t') \, dt' = \lambda t \]  

A.10

Applying A.8 for the next few terms … the pattern clearly emerges that

\[ \bar{P}(n; t) = \frac{(\lambda t)^n}{n!} \]  

A.11

Thus with A.9, the Poisson probabilities result

\[ P(n; t) = e^{-\lambda t} \frac{(\lambda t)^n}{n!} \]  

A.12

Using \( \mu = \lambda t \) as the “expected” number of events

\[ P(n) = e^{-\mu} \frac{\mu^n}{n!} \]  

A.13

The mean value is given by

\[ \langle n \rangle = \sum_{n=0}^{\infty} nP(n) = \mu \]  

A.14

and

\[ \langle n^2 \rangle = \sum_{n=0}^{\infty} n^2P(n) = \mu^2 + \mu \]  

A.15

Leading to variance as
\[ \sigma^2 = \langle (n - \mu)^2 \rangle = \langle n^2 \rangle - \mu^2 = \mu \]

**Poisson Distribution and Random Variable**

A Poisson random variable \( X = \) number of occurrences in specified time or space unit. The assumptions we make about the process generating \( X \) are as follows:

1. Occurrences are independent.
2. Any number of occurrences is possible in a given time/space unit.
3. Probability of single occurrence in a given interval is proportional to the length/size of the interval.
4. No simultaneous occurrences.
5. The expected number of occurrences during any one space/time unit is denoted by \( \mu \). This is the same for all space/time units.

Another case where the Poisson distribution is used to model when the number of occurrences is in a binomial experiment situation where the probability of success \( (p) \) is small and the number trials \( (n) \) is big. In this case we can treat \( X = \) # of successes in \( n \) trials as having a Poisson distribution with \( \mu = np \).

**Poisson Probability Function**

\[ f(x) = P(X = x) = \frac{e^{-\mu} \mu^x}{x!}, \quad x = 0, 1, 2, ... \]

**Matlab Code**

```
%*********************************************************************
*****
%This is the matlab program of generating Poisson Process.
%Using the Algorithm:
%(1)Set T(0)=0 and n=1.
%(2)Gnerate an independent random variable U(n)~U(0,1).
%(3)Set T(n)=T(n-1)-(1/lamda)ln(U(n)).
%(4)if T(n)>Tmax, Stop; otherwise, set n=n+1 and go to Step (2).
```
lambda=input('Enter The arrival Rate:');    % arrival rate
Tmax=input('Enter maximum time:');         % maximum time

clear T;
T(1)= 0;
i=1;

while T(i) < Tmax,
    U(i)=rand(1,1);
    T(i+1)=T(i)-(1/lambda)*(log(U(i)));
    i=i+1;
end

T(i)=Tmax;
stairs(T(1:i), 0:(i-1));
title(['A Sample path of the Poisson process with arrival rate ', num2str(lambda)])
xlabel('Time')
ylabel(['Number of Arrivals in [0,', num2str(Tmax), ',]'])
APPENDIX B
PUBLISHED PAPERS

Publications in International Journals

   http://mecs-press.org/ijmecs/ijmecs-v5-n9/v5n9-1.html

   http://mecs-press.org/ijmecs/ijmecs-v5-n12/v5n12-7.html

   http://www.ijcaonline.org/archives/volume84/number6/14584-2816

Paper Communicated:

1. **Sanjay Kumar Yadav, Gurmit Singh, Divakar Singh Yadav**, “Analysis of a Database Replication Algorithm under Load Sharing in Networks”, Journal of Engineering Science and Technology, School of Engineering (JESTEC), Taylor’s University, Malaysia, ISSN: 1823-4690