SIMPLE MONTE CARLO SIMULATION USING MATLAB

M. Indralingam
Department of Mathematics
University of Moratuwa

SIMPLE MONTE CARLO SIMULATION USING MATLAB

M. Indralingam
Department of Mathematics
University of Moratuwa

Abstract:

Many computer programming languages are currently available for various computing needs. MATLAB is one language which is most suitable for all scientific programming. Many engineers and scientists have not understood or unaware of its potential. In this paper we have presented the simple Monte Carlo simulation using MATLAB, which could be extended for large complex problems. Two simple examples in Queuing theory and Integration are discussed and compared with theoretical results.

1. INTRODUCTION:

Simulation[1] is one of the popular techniques used in Operational Research(OR). Generally simulation techniques are used where mathematical modeling techniques cannot be applied to any given problem. In practice simulation is done using the computer. There are various packages available for this purpose, for examples GPSSPC, SLAM, SIMSCRIPT, etc.. Recent version of MATLAB includes many toolboxes for specialized computing in the fields such as Neural Networks, Control theory, Image Processing. This paper illustrates a simple method of discrete event stochastic simulation and integration using MATLAB. Two examples are discussed with relevant MATLAB codes .

2. MATLAB

MATLAB is a vector-oriented language. All operations are generally carried out on vectors, or matrices, and not on the individual elements on the matrix. Most matrix operations are 'built-in' 'to MATLAB, and are extremely fast and accurate. Other operations are implemented using subroutines of commands which are interpreted-these may be executed very quickly as well. There are groups of subroutines for simulating nonlinear dynamical and elementary signal processing. The next section describes using the random number generation procedure, how a simple queue system can be simulated.

3.1 QUEUE MODELING

Most people have a fairly good idea of what constitutes a queue, since most have to queue for some kind of service every day. A good example of the type of queue we are interested in simulating is the queuing system of bank. A bank has customers who join a line, or several lines, in order to obtain a service. There may be many constraints on the queue. For example, the queue cannot get infinitely large in a bank because of limited room. There are many varieties of queuing mechanisms [2] and its application in various can be found in various fields. We shall look at some theory for possibly the simplest queue. The 'M/M/1' queue, which stands for 'Poisson arrivals, Exponential service and 1 server' has the following properties: time between arrivals are independent and exponentially distributed with parameter λ, service times are independent and exponentially distributed with parameter μ. The mean time between arrivals is thus λ^{-1} and the mean service time is μ^{-1}. We would like to know, among the other things, the distribution of the length of the queue at any time and the mean of the times that customers queue or wait (see appendix 1 for some theoretical results).

Matlab implementation of an M/M/1 queue

The following MATLAB function [2] implements an M/M/1 queue. It should be typed into a file 'qsim.m'. the main part of the program, the 'while loop' represents the transaction between 'busy periods'. That is each loop corresponds to one busy period.

```
function [T,N,W,busy,idle,x,y]=qsim(lam,mu,n)
%usage [T,N,W,busy,idle,x,y]=qsim(lam,mu,n)
x=-log(rand(n,1))/lam;
y=-log(rand(n,1))/mu;
W=zeros(n,1);
idle=[x(1)];
busy=[];
k=1;
j=[1];
while j(k)<=n
    jk=j(k);
    a=cumsum(x(jk+1:n));
    s=cumsum(y(jk:n-1));
    b=find(a>s);
    if isempty(b)
        j=[j;n+1];
    else
        mb=min(b);
        j=[j;jk+mb];
        idle=[idle;a(mb)-s(mb)];
    end
    jkp=j(k+1)-1;
    busy=[busy;sum(y(jk:jkp))];
    W(jk:jkp)=cumsum(y(jk:jkp)-[0;x(jk+1:jkp)]);
    k=k+1;
end

%reconstrction of queue size
k=k-1;
s=zeros(2*k,1);
s(:)=[idle';busy'];
s=cumsum(s);
N=zeros(2*n+1,1);
T=N;
N(1)=0;
T(1)=0;
for p=1:k
    jp=j(p);
    jpp=j(p+1)-1;
    z=[0;cumsum(x(jp+1:jpp));cumsum(y(jp:jpp))];
    t=kron([1;-1],ones(jpp+1-jp,1));
    [u,v]=sort(z);
    N(2*jp:2*jpp+1)=cumsum(t(v));
    T(2*jp:2*jpp+1)=u+s(2*p-1);
end
```

The following program takes the output from 'qsim.m' And computes summary statistics. It should be typed into a file called 'qsum.m'.

```
function [Nbar,Wbar,pidle,xbar,ybar]=qsum(T,N,W,busy,idle,x,y)
%usage [Nbar,Wbar,pidle,xbar,ybar]=qsim(T,N,W,busy,idle,x,y)
n=length(x);
Nbar=diff(T)'*N(1:2*n)/T(2*n);
Wbar=mean(W);
pidle=sum(idle)/(sum(busy)+sum(idle));
xbar=mean(x);
ybar=mean(y);
```

```
where Nbar: mean number people in the queue
      Wbar: mean waiting time
```

Simulation output:

The following is the output from a sample run of the above programs. The first three lines contain the calls from the MATLAB command window. The figure depicts queue length against time. Note that, although $\rho = \frac{1}{2}$ and thus the theoretical mean queue length should be 1 the queue gets quite large. For this reason, the simulation has been carried out a second time. The result show behavior which is more like that expected from the theory (Appendix1).

```
» [T,N,W,busy,idle,x,y]=qsim(1,2,100);
» [Nbar,Wbar,pidle,xbar,ybar]=qsum(T,N,W,busy,idle,x,y);
» [Nbar,Wbar,pidle,xbar,ybar]

ans =

   2.5790   2.3842   0.4021   0.9157   0.5528

» [T,N,W,busy,idle,x,y]=qsim(1,2,100);
» [Nbar,Wbar,pidle,xbar,ybar]=qsum(T,N,W,busy,idle,x,y);
» [Nbar,Wbar,pidle,xbar,ybar]

ans =

   0.8073   0.7208   0.5133   0.8929   0.4367
```

The following codes produce a graph of queue length Vs time (figure3.1) . Similarly other relevant graphs can be produced.

```
» stairs(T,N)
» title('Queue simulation\lambda=1, \mu=2')
» xlabel('time'),ylabel('Queue length')
```

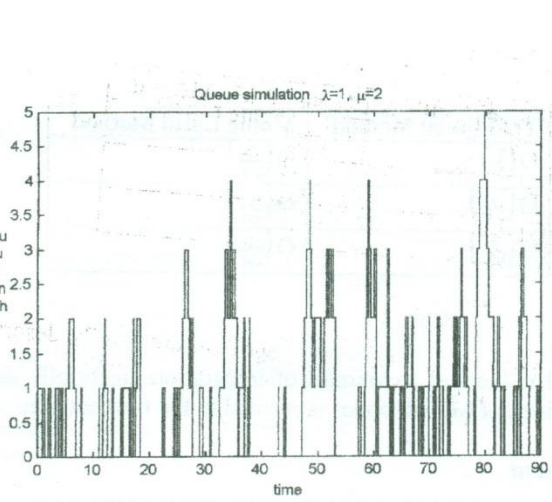

Figure 3.1

To simulate G/G/1 (Queue model with general arrival and service distribution), only we have to write function procedures to get the values for **x** (arrival rate) and **y** (service rate) from a given empirical distribution. This would be possible using many features provided by the MATLAB programming language.

4. Monte Carlo Integration

Suppose we wish to approximate the integral

$$I = \int_{a_1}^{b_1} \dots\dots \int_{a_2}^{b_2} g(x_1, \dots\dots x_p)\, dx_1 \dots . dx_p$$

Again without loss of generality let us assume that $a_i = 0$ and $b_i = 1$ $\forall$i=0 . Let $\{X_i : i = 1,2,\dots N\}$ be N points uniformly distributed in the p-dimensional unit hypercube $[0,1]^p$. This can be simulated by generating Np uniform random numbers and forming them into N p-dimensional vectors). Monte-Carlo approximation of the above integral I is defined by:

$$\hat{I} = \frac{1}{N} \sum_{i=1}^{N} g(X_i)$$

In the appendix 2 ,order of the error of the M-C integration is shown to be $1/\sqrt{N}$. Error order of traditional method and Monte Carlo methods are given in the table 4.1. It shows for three or more dimensions, Monte Carlo integration should achieve smaller errors than the traditional techniques.

P	Traditional Method	Monte Carlo Method
1	$O\left(\frac{1}{N}\right)$	$O\left(\frac{1}{\sqrt{N}}\right)$
2	$O\left(\frac{1}{\sqrt{N}}\right)$	$O\left(\frac{1}{\sqrt{N}}\right)$
3	$O\left(\frac{1}{\sqrt[3]{N}}\right)$	$O\left(\frac{1}{\sqrt{N}}\right)$

Table 4.1

In Appendix 3 traditional numerical techniques of integration are briefly described. It may help the reader to get the idea of Matlab codes presented in the next section.

4.1 Matlab Implementation

In this section, the following example is selected to demonstrate the Monte Carlo integration procedure.

$$\int_0^1 \cdots \int_0^1 e^{-\sum_{i=1}^{p} x_i} dx_1 dx_2 \ldots dx_p$$

The Matlab function given below are easily adopted to much more general integrands. Although it is easy to write Monte Carlo routines to do this for general p, it is not as easy to write general numerical integration programs. For this reason, the code presented below will work for only $p \leq 3$. The first four segments of code contains three Matlab functions: the first implements the Monte Carlo technique, the second the 'left' integrals, the third the 'right' integrals and the fourth the 'midpoint' rule, where areas are approximated by adding the areas of rectangles drawn through the ordinate corresponding to the midpoint of the intervals. The fifth segment contains the code for a function which compute the four integrals.

```
function y=mcexp(p,N)
% usage y=mcexp (p,n)
if p>1
y = mean(exp(-sum(rand(p,N))));
    else
y= mean(exp(-rand(N,1)));
end


function y=rectarl(p,n)
%usage y=rectarl(p,n)
x=(0:n-1)'/n;
k=ones(size(x));
y=0;
if p==1
    y=mean(exp(-x));
elseif p==2
    y=mean(mean(exp(-(x*k'+k*x'))));
elseif p==3
for j=0:n-1
    y=y+mean(mean(exp(-(x*k'+k*x'+j/n))));
end
y=y/n;
```

```
else
y=naninf;
end


function y=rectarr(p,n)
%usage y=rectarr(p,n)
x=(1:n)'/n;
k=ones(size(x));
y=0;
if p==1
   y=mean(exp(-x));
elseif p==2
   y=mean(mean(exp(-(x*k'+k*x'))));
elseif p==3
   for j=1:n
      y=y+mean(mean(exp(-(x*k'+k*x'+j/n))));
   end
   y=y/n;
else
   y=naninf;
end


function y=rectarc(p,n)
%usage y=rectarc(p,n)
x=((0:n-1)'+0.5)/n;
k=ones(size(x));
y=0;
if p==1
   y=mean(exp(-x));
elseif p==2
   y=mean(mean(exp(-(x*k'+k*x'))));
elseif p==3
   for j=0:n-1
      y=y+mean(mean(exp(-(x*k'+k*x'+(j+0.5)/n))));
   end
   y=y/n;
else
   y=naninf;
end


function [y1,y2,y3,y4,y]=mmcexp(n)
%usage [y1,y2,y3,y4,y]mcexp(n)
for p=1:3
   y1=mcexp(p,n^p);
   y2=rectarl(p,n);
   y3=rectarr(p,n);
   y4=rectarc(p,n);
   y=(1-exp(-1))^p;
   fprintf('For dimension %2.0f, the MC, left, centre',p);
   fprintf('and right integrals are ');
   fprintf('%7.6f %7.5f %7.5f %7.5f\n',y1,y2,y4,y3);
   fprintf('The integral is % 7.5f\n',y);
   fprintf('The errors are %7.5f %7.5f ',abs(y-y1),abs(y-y2));
   fprintf('%7.5f and %7.5f\n\n',abs(y-y4),abs(y-y3));
end
```

Simulation Output:

```
» [y1,y2,y3,y4,y]=mmcexp(10);
For dimension  1, the MC, left, center and right integrals are 0.591330 0.66425
0.63186 0.60104
The integral is 0.63212
The errors are 0.04079 0.03213 0.00026 and 0.03108

For dimension  2, the MC, left, center and right integrals are 0.407794 0.44123
0.39924 0.36125
The integral is 0.39958
The errors are 0.00822 0.04166 0.00033 and 0.03833

For dimension  3, the MC, left, center and right integrals are 0.247943 0.29309
0.25226 0.21713
The integral is 0.25258
The errors are 0.00464 0.04051 0.00032 and 0.03545

» [y1,y2,y3,y4,y]=mmcexp(20);
For dimension  1, the MC, left, center and right integrals are 0.601168 0.64806
0.63205 0.61645
The integral is 0.63212
The errors are 0.03095 0.01593 0.00007 and 0.01567

For dimension  2, the MC, left, center and right integrals are 0.401639 0.41998
0.39949 0.38001
The integral is 0.39958
The errors are 0.00206 0.02040 0.00008 and 0.01957

For dimension  3 For, the MC, left, center and right integrals are 0.251181 0.27217
0.25250 0.23426
The integral is 0.25258
The errors are 0.00140 0.01959 0.00008 and 0.01832
» [0.00768 0.00001 and 0.00748
```

Summary of the simulation results are presented in the following table.

P/n	10	20	30	40	50
1	0.04079	0.03095	0.04275	0.04863	0.00825
2	0.00822	0.00206	0.00145	0.00217	0.00205
3	0.00464	0.00140	0.00002	0.00025	0.00022

Note that, although the errors using the traditional numerical techniques are improving with n, the Monte Carlo techniques yield the above errors, which do not uniformly decrease with n due to random sampling.

5. CONCLUSION:

In this paper, We have shown how stochastic simulation can be carried out in a simple and effective way using MATLAB. Two examples (Queue and Integration) are considered for this purpose. With further work , it should be possible to simulate complex simulation. The MC integration technique considered here can be effectively used with multiple integration with complicated integrands which will be useful for practical applications such as volume and surface area enumerations of different shapes . The codes presented here do not use the full power of graphic, animation procedures of MATLAB. We think it should be possible use many other built in functions of MATLAB to perform animated complex simulation with graphics.

REFERENCES

1. LAW ,A.M and W.D.KELTON (1991), Simulation Modeling and Analysis, ed, McgrawHill, New York, NY.
2. PART-Enander.E, SJOBERG. A, MELIN.B and ISAKSON. (1996) The MATLAB hand-book, Addison Wesley Longaman
3. BANKS. J, CARSON. J. S and NELSON. B. (1996) Discrete -Event system simulation, Pretice-Hall of India

Appendix 1:

Some basic theoretical results of M/M/1 queue:

If time between arrival is assumed as exponentially distributed with parameter λ and time between service time is assumed as exponentially distributed with parameter μ, the following result can be proved.

Probability that the service is busy $= \dfrac{\lambda}{\mu}$

Average Number of Customers in the Queue $= \dfrac{\lambda}{\mu}\dfrac{\lambda}{\mu-\lambda}$

Average Number of Customers in the System $= \dfrac{\lambda}{\mu-\lambda}$

Expected waiting time of a customer in the queue $= \dfrac{\lambda}{\mu} \cdot \dfrac{1}{\mu-\lambda}$

Expected time a customer spend in the system $= \dfrac{1}{\mu-\lambda}$

Above techniques may be generalized to more than one dimension. Suppose we wish to approximate the integral.

$$\int_{a_1}^{b_1}\int_{a_2}^{b_2} g(x_1, x_2)dx_1dx_2$$

Without loss of generality, let's suppose that $a_1 = a_2 = 0$ and $b_1 = b_2 = 1$ (We may transform x_1 and x_2 so that the limits will be 0 and 1)

Divide the x_1 and x_2 unit intervals into n equal segments. Then the sum of the 'left volume is

$$\frac{1}{n^2}\sum_{i=0}^{n-1}\sum_{j=0}^{n-1} g\left(\tfrac{i}{n}, \tfrac{j}{n}\right)$$

$$\frac{1}{n^2}\sum_{i=0}^{n-1}\sum_{j=0}^{n-1} g\left(\tfrac{i}{n}, \tfrac{j}{n}\right)$$

The error will be of the same order as the difference between the two approximations. The difference is

$$\frac{1}{n^2}\left[\sum_{i=1}^{n} g\left(\frac{i}{n},1\right)+\sum_{j=1}^{n-1} g\left(1,\frac{j}{n}\right)-\sum_{i=0}^{n-1} g\left(\frac{i}{n},0\right)-\sum_{j=1}^{n-1} g\left(\frac{j}{n},0\right)\right]$$
$$=\frac{1}{n}\left[\int_0^1 g(x,1)+\int_0^1 g(1,x)dx-\int_0^1 g(x,0)dx-\int_0^1 g(0,x)dx+O\left(\frac{1}{n}\right)\right]$$
$$=O\left(\frac{1}{n}\right)$$

However, the number of points at which the function has been computed is $N = n^2$, so that the order of the error is $\frac{1}{\sqrt{N}}$. In general, for p dimensional integrals, the order of the errors is $\frac{1}{\sqrt[p]{N}}$.