Application of Monte Carlo Method Based on Matlab:
Calculation of Definite Integrals and Simulation of Heston's Model

Yannan Gao¹ and Xin Zhao²

¹School of economics, Shandong Women’s University, Daxue Road No. 2399, Changqing District, Jinan, Shandong, China. Email: sddxgyn@qq.com.
²School of management, Shandong Women’s University, Daxue Road No. 2399, Changqing District, Jinan, Shandong, China. Email: 965509529@qq.com

Abstract. This paper discusses Monte Carlo method in three aspects: pi-approximation, an algorithm to calculate definite integral and simulation to generate financial time series. The first two calculations are based on geometric probability: to calculate the probability that the random points fall within the certain area. The third one is to transfer a stochastic differential equation into a difference equation and realize these equations on matlab to derive a time series which has the properties of the corresponding stochastic differential equation. By analyzing these time series, one can make further analysis on these data, e.g. density function. The paper shows the applicability of Monte Carlo method. The method gives practitioners accessible means of solving complicated models and is easy to operate on computers.

One uses Monte Carlo method to get statistical conclusions by applying simulation techniques to carry on numerous experiments on computers. The method can simplify some complicated mathematical models and generate data, which can be the basis of further analyses. It’s widely used in natural sciences, statistics and finance. An example is that a finance guy can simulate return time series of financial assets and do further research on that data. Mathematicians can solve difficult equations or make numerical calculations through this simple but tricky method. This paper tries to illustrate the charm of this method and provides some codes based on the software matlab, which could be a good reference for the readers to get captivated by this interesting method.

1. Calculating Pi Using Monte Carlo Method
Calculating pi by using simulation method is a computer realization of the so-called random experiment in statistics. There are many ideas supporting this realization. A common one is that one can calculate the ratio of the areas between a square with the side length 1 and its inscribed circle. Referring to the geometric probability, one throws many beans, (the size of which is so small that it can be ignored) into a square sided 1. The probability that the beans fall within the inscribed circle is equal to the ratio of areas between the inscribed circle and the square. We denote this ratio as k. It’s easy to derive that $k = \frac{\pi}{4}$.

The more concrete idea of the algorithm is to generate two random numbers x and y which follow uniform distribution at the interval (-1, 1). Each time when matlab generates a pair of numbers x and y, we form a coordinate which corresponds to a point within the square. The equation of inscribed circle is $x^2+y^2=1$. The points in the circle must satisfy the condition $x^2+y^2<1$. The codes are presented as in appendix B.

$N$ denotes the number of times of throwing beans. $m$ is used to keep record of the number of beans
falling within the inscribed circle. 4m/n is the estimated value of pi. Theoretically, the bigger n is, the more accurate result we get. One thing to notice is that the same n will release different results as random numbers are newly generated every time matlab calculates a result. One can write a code including a function using the command function in matlab so that one can call the program to calculate pi. Another thing to mention is that one can take average of different results generated by same n. From Table 1 we can find that the calculating result is different. And the bigger n is, the more accurate result we can get.

| Times of experiment | n=100000 | n=200000 |
|---------------------|---------|----------|
| Results (five decimal places kept) | 3.13616 | 3.14312 |
| 3.14260 | 3.13726 |
| 3.14252 | 3.14760 |
| 3.14432 | 3.13864 |
| 3.14016 | 3.14352 |
| 3.14852 | 3.14310 |
| 3.14200 | 3.14282 |
| 3.15068 | 3.13924 |
| Mean | 3.14300 | 3.14125 |
| Absolute error | 0.0014 | 0.0003 |

2. Calculating Definite Integral Using Monte Carlo Simulation Method

The idea of the calculation of definite integral is very similar. According to the geometric meaning of definite integral, definite integral is the area of the geometric figure that integrand curve and horizontal axis form at a definite integral. Therefore, we can transfer the ratio into the definite integral according to the geometric probability theory.

The concrete idea to calculate the definite integral \( I = \int_{a}^{b} f(x) \, dx \) is to generate random numbers following uniform distribution within the integral \((a,b)\). And then make sure of the maximum of the integrand \( f_{\text{max}} \) at the interval. We can easily find that \( x=a, x=b, x\)-axis and \( y=f_{\text{max}} \) forms a rectangle with its length \( b-a \) and its width \( f_{\text{max}} \). And the continuous curve \( f(x), x=a, x=b \) along with the horizontal axis form the geometric figure \( I \), the area of which is what we want to derive as the result of the definite integral. Now we start the experiment, throw the beans randomly to the rectangle. Suppose the ratio of number of beans into \( I \) and number thrown in total is \( r \), then we get

\[
r = I / [(b-a)f_{\text{max}}] \quad (1)
\]

\( I \) denotes the result of the definite integral

\[
I = r[(b-a)f_{\text{max}}] \quad (2)
\]

If the integrand is below the \( x \)-axis, one can transfer the integrand to make it positive. Easy to prove

\[
\int_{a}^{b} f(x) \, dx = \int_{a}^{b} [f(x) + g] \, dx - g(b - a) \quad (3)
\]

One can make flexible transforms to our definite integrals to facilitate the calculations.

Let’s take \( \int_{0}^{1} \sin(x) \, dx \) as an example using the above idea.

First calculate this integral using precise method

\[
\int_{0}^{1} \sin(x) \, dx = 1 - \cos(1) \approx 0.4596976941 \ldots \quad (4)
\]

According to the properties of function \( \sin(x) \), it’s a monotonically increasing function with its
maximum existing at x=1, namely sin(1) ≈ 0.84147. The area of rectangle is sin(1), random number x is generated within the interval (0,1). y should be generated within the interval (0,sin(1)). The points within the figure I should follow the condition

\[ y - \sin(x) \leq 0 \]  \hspace{1cm} (5)

It’s very easy to write a code based on the analysis above, see appendix A.

The result comes with Table 2. The figure is shown in Figure 1.

Table 2. Calculating result of the definite integral.

| Times of experiments | n=10000 | n=100000 |
|----------------------|---------|-----------|
| Calculating result(15 decimal places kept) | 0.466090778485094 | 0.460343531658856 |
| 0.464239542318517 | 0.459771331389187 |
| 0.465585895894209 | 0.460091090363414 |
| 0.458685833818784 | 0.460898902508829 |
| 0.453300419516014 | 0.460335116949008 |
| 0.459359010606631 | 0.458820469176354 |
| 0.454394331796264 | 0.460335116949008 |
| 0.477955519370885 | 0.460015357974781 |
| 0.459274863508150 | 0.459451572414960 |
| 0.459359010606631 | 0.459697694131860 |
| Mean | 0.461824520592118 | 0.459982540606374 |
| Real value using precise method | 0.459697694131860 | 0.459697694131860 |
| Absolute error | 0.002126826460258 | 0.000284846474514 |
| Relative error | 0.4626576% | 0.0619639% |

Figure 1. Calculation of definite integral \( \int_0^1 \sin(x) \, dx \)

3. Monte Carlo Simulation of Financial Time Series: an Example of Heston’S Model

Monte Carlo simulation method can be used as a tool to simulate financial time series. Then one can work with these financial data for further researches. We consider a return model introducing stochastic volatility, namely the Heston’s model put forward in 1993. Heston’s model includes two patterns: the price follows a usual BS stochastic differential equation. The volatility (standard deviation) follows a stochastic volatility process, Ornstein-Uhlenbeck process.

The price of underlying asset follows

\[ dS_t = \mu S_t \, dt + S_t \sqrt{V_t} \, dW_t(t) \]  \hspace{1cm} (6)
W denotes Wiener process.

\( \sqrt{V_t} \) follows an Ornstein-Uhlenbeck process, simply using Ito’s lemma we can get the process \( V_t \) following

\[
dV_t = \kappa(\varphi - V_t)dt + \sigma\sqrt{V_t}dW_2(t)
\] (7)

The relative variables or parameters include:
- \( S_t \) - price of underlying assets at time \( t \)
- \( \kappa \) - mean-reverting speed parameter
- \( \rho \) - correlation of two Wiener processes
- \( \phi \) - long-run variance
- \( \sigma \) - volatility of volatility
- \( V \) - present variance of underlying
- \( r \) - risk-less rate
- \( T \) - maturity
- \( K \) - Strike price
- \( \lambda \) - Volatility risk premium

The idea to simulate is to simply discretize the above two stochastic differential equations

\[
S_t = S_{t-1} + \mu S_{t-1}\Delta t + S_{t-1}\sqrt{V_t}\sqrt{\Delta t}W_1(t)
\] (8)

\[
V_t = V_{t-1} + \kappa[\varphi - V_{t-1}]\Delta t + \sigma\sqrt{V_{t-1}}\sqrt{\Delta t}W_2(t)
\] (9)

\( W_1(t) \) and \( W_2(t) \) are two random variables following standard normal distribution, the correlation of which is \( \rho \). We can use two more accessible notations to represent these two variables, \( n_1(t) \) and \( n_2(t) \). Using some simple tricks of standard normal, we can derive

\[
W_1(t) = n_1(t)
\] (10)

\[
W_2(t) = \rho n_1(t) + \sqrt{1 - \rho^2}n_2(t)
\] (11)

Appendix C is attached as a reference of the codes performed on matlab. In this program, parameters are pre-set, which can be adjusted to reflect different financial time series. Norm1 and norm2 are generated by randn command, which generates of random numbers following standard normal. We should notice that the random numbers, when calculating pi and definite integrals, following uniform distribution. From the above simulation, we get the following graph. Then we can make analysis based on these time series, e.g. analyze the density function of returns.

In order to get more accurate simulation of density function of returns, we use simulation method to obtain 100000 prices. Then we derive log-returns of the simulated stock prices: \( R_t = \ln(S_t/S_{t-1}) \). Matlab provides us with a very efficient function named ksdensity, through which we can derive a fitting density function. The matlab codes of Figure 3 are shown in appendix D. Appendix D is based on the prices that appendix C already generates. Through simulation we can also see the impact that different parameters have on density functions. For example, a positive \( \rho \) and a negative \( \rho \) could leave different results.
Figure 2. Simulation of financial time series generated by Heston’s SDE

Figure 3. Density function of returns generated by simulated data (100000 data)

4. Conclusion
This paper introduces three examples of Monte Carlo simulation method, which presents good applications in different fields. Actually, Monte Carlo is deemed as a good method for practitioners, especially in the field of financial economics. Many boring and profound mathematical models can be simplified through this method. It’s also a very good numerical tool in the field of mathematics, finance or computational science. Hopefully through this paper there will be more Chinese researchers, especially in the field of economics and finance, becoming interested in this method to be more competent in dealing those complicated big models.

Appendix A  Codes of calculation of pi

```matlab
n=5000; %input of number of throwing beans
m=0;  %to record the number of beans falling within the circle
for i=1:1:m;
    x=2*rand-1;
    y=2*rand-1;
    if x^2+y^2<=1;
        m=m+1;
    end
end
m*4/n %approximation of pi
```
Appendix B  Codes of calculation of definite integral

\begin{verbatim}
n=1000;  %input of number of throwing beans
m=0;    %to record the number of beans falling within the curve
for i=1:1:n;
x=rand;
y=rand*sin(1);
if y-sin(x)<=0;
m=m+1;
end
end
format long
m*sin(1)/n  %approximation of definite integral
\end{verbatim}

Appendix C  Codes of simulation of Heston’s SDE

\begin{verbatim}
r=0;
kappa=2;
fan=0.01;
sigma=0.1;
delt=0.01;
ro=0.5;
V=[0.01 zeros(1,1000)];
S=[10 zeros(1,1000)];
for i=1:1:1000
    norm1=randn(1);
    norm2=randn(1);
    W1=norm1;
    W2=ro*norm1+sqrt(1-ro^2)*norm2;
    V(i+1)=abs(V(i)+kappa*(fan-V(i))*delt+sigma*sqrt(V(i))*sqrt(delt)*W2);
    S(i+1)=S(i)+r*S(i)*delt+S(i)*sqrt(delt)*sqrt(V(i))*W1;
end
p=1:1001;
plot(p,S(p))
\end{verbatim}

Appendix D  Codes of deriving a density function

\begin{verbatim}
A=S;  %generate a new S; S is what we have already got through appendix C
B=S;  %generate another new S
A(1)=[];  %to delete the first element of the vector
B(100001)=[];  %to delete the last element of the vector
R=log(A./B);
ksdensity(R)
\end{verbatim}

5. References
[1] Heston S L 1993 A closed-form solution for options with stochastic volatility with application to bond and currency options Review of Financial Studies. 6(2) pp 327-43
[2] Peng T and Wei L Y A new way to compute PI with Monte Carlo method in R language 2014 Computer Knowledge and Technology 10(17) pp 4038-39