Lecture 2

Monte Carlo methods

We now look at a numerical scheme that uses the probabilistic solution - Monte Carlo techniques. The main idea behind the Monte Carlo technique is that you simulate paths that could be taken by the underlying asset (under the risk-neutral probability) and then use these to estimate an expected option price at expiry, which can be discounted back to today. Monte Carlo techniques are very useful for options on more than one underlying asset. Sadly, the convergence of Monte Carlo methods is slow and it is hard to determine the error terms. The convergence to the correct option value will be at a rate of $n^{-\frac{1}{2}}$ where n is the number of sample paths. As it is a forward induction technique, which makes it particularly suitable for valuing path dependent options such as lookback and Asian options. It is very unsuitable for valuing American style options for exactly the same reasons, although we shall see methods for overcoming this.

Example 2.1 (Simulations)**.** Plot some example stock price paths on a graph.

Solution 2.1.

The computational effort increases linearly as you add underlying assets, thus to price an option with d underlying assets (or sources of uncertainty such as stochastic volatility or stochastic interest rates) then an n sample paths Monte Carlo method requires approximately nd calculations. Practitioners love these methods!!

2.1 Background - Large numbers

If we have a sequence of independent, identically distributed random variables Y^i then we have that

$$\lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{N} Y^i = E[Y]$$

which is the law of large numbers. In other words the expectation is exactly like taking a long run average (as we'd expect), so to evaluate the expectation to any desired accuracy we can simply take more and more draws of Y^i.

With the Monte Carlo technique what we are trying to do is to evaluate the value of $E[f(Y_T)]$ which is the expectation of a function of a random variable Y_T.

2.2 Application to Options

If we consider S_t as the value of a share price at time t, then the option value at expiry, $t = T$ we can think of as $V(S_T, T)$ and from the fundamental theorem of finance we know that

$$V(S_t, t) = E_t^Q[e^{-\int_t^T r(s)ds} V(S_T, T)]$$

or

$$e^{-r(T-t)} E_t^Q[V(S_T, T)]$$

if r is constant, where Q is the risk-neutral measure and E_t denotes taking the expectation at time t. Thus if we can estimate the expectation on the right hand side then we can simply discount this value at the risk-free rate to obtain the option price today. In fact, with Monte-Carlo methods it is also fairly straightforward to factor in stochastic interest rates as well.

Example 2.2 (How to generate risk-neutral paths)**.** Outline how to generate random paths of a Geometric Brownian Motion.

Solution 2.2.

To estimate the expected option value at time T, $V(S_T)$ then we take random draws from the $N(0,1)$ distribution which enables us to calculate S_T and then calculate $V(S_T)$. To get an approximation of the expectation we then average $V(S_T)$. Thus if the ith draw from the normal distribution gives $V(S_T^i)$ then by the law of large numbers:

$$\frac{1}{n}\sum_{i=1}^{n} V(S_T^i) \to E_t^Q[V(S_T)] \quad \text{as} \quad n \to \infty$$

Example 2.3 (Calculating errors)**.** What is the error if we calculate an option value with a single path?

Solution 2.3.

2.2.1 Central limit theorem and error

Definition 2.1 (Central limit theorem). If $V(S_i^T)$ is a sequence of independent and identically distributed random variables with mean $E_t^Q[V(S_T)]$ and variance η^2, then we can say that

$$\sqrt{n}\left(\frac{1}{n}\sum_{i=1}^{n} V(S_T^i) - E_t^Q[V(S_T)]\right) \xrightarrow{d} N(0, \eta^2).$$

Example 2.4 (Central limit theorem)**.** What does this mean we can say about our calculation errors?

Solution 2.4.

2.2.2 The Monte-Carlo method for European options

That gives the basics of the Monte Carlo method, it is very simple to implement for many different types of options. For a European call option the payoff at maturity $V(S_T)$ is given by

$$V(S_T) = \max(S_T - X, 0)$$

and so, to value the option one simulates N possible values or paths for S_T by making n independent draws from $N(0,1)$ then to use these possible values, call them ϕ^i we have for $1 \leq i \leq n$

$$\begin{aligned} S_T^i &= S_t \exp[(r - \tfrac{1}{2}\sigma^2)(T-t) + \sigma\phi^i\sqrt{T-t}] \\ V(S_T^i) &= \max(S_T^i - X, 0) \\ V(S_t, t) &= e^{-r(T-t)}\frac{1}{n}\sum_{i=1}^{n} V(S_t^i) \end{aligned}$$

Example 2.5 (Pseudo Code)**.** Write down a simple Monte Carlo option algorithm in pseudo code.

Solution 2.5.

2.2.3 Valuing Asian options

An Asian option is an option whose payoff is a function of the average price of the underlying asset over the lifetime of the option. The share price is observed at M points in time (every day, every trading day, every week etc.) and the average is calculated by using either geometric or arithmetic averaging.

Consider an Asian option whose payoff is

$$V(T) = \max(A - S, 0)$$

where

$$A = \frac{1}{K}\sum_{k=1}^{K} S(t_k)$$

and $S(t_k)$ are the share prices at the M sampling times t_1, ..., t_K. We need to modify our Monte-Carlo method slightly to deal with this.

From the solution to the SDE we know that

$$S_{t_k} = S_t \exp[(r - \tfrac{1}{2}\sigma^2)(t_k - t) + \sigma(W(t_k) - W(t))]$$

$$S_{t_{k+1}} = S_{t_k} \exp[(r - \tfrac{1}{2}\sigma^2)(t_{k+1} - t_k) + \sigma(W(t_{k+1}) - W(t_k))]$$

and so the procedure is as follows:

1. Simulate each of the K increments dW_k by drawing ϕ_k from the Normal distribution and use
$$S^n_{t_k} = S^n_{t_{k-1}} \exp[(r - \tfrac{1}{2}\sigma^2)(t_k - t_{k-1}) + \sigma\sqrt{t_k - t_{k-1}}\phi_k]$$
to estimate the underlying asset values at each time.

2. Calculate the value of A^i and the payoff $\max(A^i - X, 0)$

Then repeat this procedure n times to get the option value:
$$V(t) = e^{-r(T-t)}\frac{1}{n}\sum_{i=1}^{n}\max(A^i - X, 0)$$

Note that here, we need to break up the path of S_t into K time steps and then use the n created paths to take the average.

Example 2.6 (Path Dependent Options)**.** Outline the procedure for valuing a Path Dependent option.

Solution 2.6.

2.3 Generating (Pseudo-)Random Numbers

Many statistical packages have (normal) random number generators which are available to use. By default the standard c++ libraries now have such a package. However if we want to do more complex things, such as multithreading, GPU calculations or low discrepancy sequences, we still need to know how to generate normally distributed random numbers from simpler distributions.

Without a library to implement it, we generate Normal random numbers by:

1. generate a sequence of integers;
2. transform to random numbers that are uniformly distributed on $[0, 1]$;
3. transform to obtain normally distributed random numbers.

The transformation from step 1 to 2 is easily done (divide the integers by the maximum value in the sequence), however step 2 to 3 is a little bit more tricky and warrants some discussion.

The most straightforward way is to take an inverse function. Let F^{-1} denote inverse of Normal distribution function, if x is random number uniformly distributed on $[0,1]$, then $y = F^{-1}(x)$ is a standard variable from the normal distribution.

If you have function F^{-1} then this is easy. For example, in Excel:-

- Has function RAND() which generates a random number uniformly distributed on $[0,1]$ (steps 1 to 2 is encoded already in excel, in c language *rand* returns an integer)
- Has function F^{-1}, called NORSINV ()
- Thus, function call NORSINV(RAND()) returns a realisation of a standard normal random variable

However, F^{-1} is not known in closed form, so this approach is not always fast.

The other issue is that these are only "pseudo" random numbers in that the computer program typically has an algorithm for calculating the 'random numbers', as John von Neumann said in 1951 "*Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin*".

There are many alternatives (The "Recipe" books of Press, Teuklovsky, Vetterling and Flannery are a good source, and have a good discussion of the problem), such as the Mid-Square method, Congruential generation or the popular Mersenne twister. In the lab classes we will use an implementation of the Mersenne Twister algorithm contained in the c++ standard *random* library.

Example 2.7 (Is it really random?)**.** Draw a unit square and place 10 crosses inside, assuming that the x and y coordinates are drawn from uniform distributions. What does this look like if a computer does it?

Solution 2.7.

2.4 The Box-Muller method

The simplest technique for generating 'decent' Normally distributed random numbers (according to Wilmott) is the Box-Muller method which takes any uniformly distributed variables (from any source you prefer) and turns them into Normally distributed ones. The method is actually quite simple. Given two uniformly distributed random numbers x_1, x_2, then two Normally distributed random numbers, y_1 and y_2 are given by:

$$y_1 = \cos(2\pi x_2)\sqrt{-2\log(x_1)}, \quad y_2 = \sin(2\pi x_1)\sqrt{-2\log(x_2)}.$$

This method works by transforming random draws from the unit square into a quarter of a unit circle (the cos/sin bit), then stretching into log space (the log bit). You can speed this process up by using a rejection method to get a random number in the quarter circle rather than using the sin and cos functions which are relatively expensive. After taking x_1, x_2, calculate the radius of the point $r = \sqrt{x_1^2 + x_2^2}$ and reject if $r > 1$. The calculation then becomes

$$y_1 = x_1\sqrt{\frac{-2\log(r)}{r}}, \quad y_2 = x_2\sqrt{\frac{-2\log(r)}{r}}.$$

2.5 More General Problems

The two examples we talked about above - BSM setup (geometric Brownian motion) are particularly easy because increments to $\log S$ are normally distributed or alternatively we can write the solution to the SDE out explicitly in terms of the Normal distribution But what do we do for non-Normal stochastic processes?

Previously we introduced the SDE,

$$dS_t = \mu S_t dt + \sigma S_t dX;$$

we first introduced a process

$$S_{t+\delta t} = S_t + \mu S_t \delta t + \sigma S_t \phi \sqrt{\delta t}$$

where δt a small increment of time and ϕ a Normal r.v. The second process is more than a way to explain the first process

- Use Euler scheme to approximate the SDE
- Letting $S_t^{\delta t}$ denote the value of the Euler process at time t using an increment of δt , $S_t^{\delta t} \to S_t$ in the sense of mean square.

2.6 Stochastic volatility

The Euler approximation is useful for processes that are not normal, i.e. something other than GBM. For example, consider a stochastic volatility model. Under Q, we have

$$dS_t = rS_t dt + \sigma_t S_t dW_1,$$

$$d\sigma_t = (a - b\sigma_t)dt + v\rho\sigma_t dW_1 + v\sigma_t\sqrt{1-\rho^2}dW_3.$$

Here we have chosen a model where

- a, b, v, r, ρ are constants,
- σ and S are stochastic processes
- σ_t is interpreted as volatility at time t.

Now let use take $K+1$ equally spaced times t_0, t_1, ..., t_K, where $\delta t = t_k - t_{k-1}$. The Euler approximation to this process is given by

$$S_{t_k} = S_{t_{k-1}} + rS_{t_{k-1}}\delta t + \sigma_{t_{k-1}} S_{t_{k-1}} \phi_{1,k}\sqrt{\delta t}$$

$$\sigma_{t_k} = \sigma_{t_{k-1}} + (a - b\sigma_{t_{k-1}})\delta t + v\rho\sigma_{t_{k-1}}\phi_{1,k}\sqrt{\delta t} + v\sigma_{t_{k-1}}\sqrt{1-\rho^2}\phi_{2,k}\sqrt{\delta t}$$

where the $\phi_{j,k}$ $(j = 1, 2)$ are independent standard normal r.v.'s.

Notice that the only change from before is that we now do simulation using Euler approximation - we must compute values at intermediate timesteps (GBM is special as the SDE can be integrated 'exactly').

2.7 Two underlying assets

Now assume that we wish to price a contract depending on two correlated underlying assets such that

$$dS_1 = \mu_1 S_1 dt + \sigma_1 S_1 dX_1$$

$$dS_2 = \mu_2 S_2 dt + \sigma_2 S_2 dX_2^*$$

where

$$X_2^* = \rho X_1 + \sqrt{1-\rho^2}X_2$$

or changing the notation slightly

$$dS_1 = \mu_1 S_1 dt + \sigma_1 S_1 dX_1$$

$$dS_2 = \mu_2 S_2 dt + \sigma_2 \rho S_2 dX_1 + \sigma_2\sqrt{1-\rho^2}S_2 dX_2$$

then it is very straightforward to value an option whose payoff depends upon both of the asset prices at expiry.

Consider an option where $V(T) = \max(S_1 - X_1, S_2 - X_2, 0)$. So

$$V(t) = e^{-r(T-t)} E_t^Q[V(T)]$$

Under risk-neutrality we have

$$dS_1 = rS_1 dt + \sigma_1 S_1 dX_1$$
$$dS_2 = rS_2 dt + \sigma_2 \rho S_2 dX_1 + \sigma_2 \sqrt{1-\rho^2} S_2 dX_2$$

thus from solving the SDEs

$$S_{1T}^i = S_{1t}^i \exp[(r - \tfrac{1}{2}\sigma_1^2)(T-t) + \sigma_1 \sqrt{T-t}\phi_1^i]$$
$$S_{2T}^i = S_{2t}^i \exp[(r - \tfrac{1}{2}\sigma_2^2)(T-t) + \sigma_2 \rho \sqrt{T-t}\phi_1^i + \sigma_2 \sqrt{1-\rho^2}\sqrt{T-t}\phi_2^i]$$

So to value the equations simulate $2n$ draws of normally distributed random numbers and then determine the possible values of $V(T)$ and then discount the average.

Thus the option value can be estimated as follows:

$$V(t) = e^{-r(T-t)} E_t^Q[V(T)] = \frac{1}{n} \sum_{i=1}^{n} \max(S_{1t}^i - X_1, S_{2t}^i - X_2, 0)$$

One question, however, is how to get the general form of the set of SDEs when we have more than one underlying asset.

The way in which we typically see correlation matrices (in two underlying assets) are in terms of the covariance matrix of continuously compounded returns, i.e. $\log(S_j(T)/S_j(t))$ and this matrix is typically of the form

$$\begin{pmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2 \end{pmatrix}$$

We can write our two SDEs as follows:

$$\begin{pmatrix} dS_{1t} \\ dS_{2t} \end{pmatrix} = \begin{pmatrix} rS_{1t} \\ rS_{2t} \end{pmatrix} dt + \begin{pmatrix} \sigma_1 S_{1t} & 0 \\ \sigma_2 \rho S_{2t} & \sigma_2 \sqrt{1-\rho^2} S_{2t} \end{pmatrix} \begin{pmatrix} dW_{1t} \\ dW_{2t} \end{pmatrix}$$

It so happens that the matrix, M, here

$$\begin{pmatrix} \sigma_1 & 0 \\ \rho\sigma_2 & \sigma_2\sqrt{1-\rho^2} \end{pmatrix}$$

is the square root of the covariance matrix, Σ

$$\begin{pmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2 \end{pmatrix}$$

in that $\Sigma = MM^T$.

You can check this yourself by noting that

$$\begin{pmatrix} \sigma_1 & 0 \\ \rho\sigma_2 & \sigma_2\sqrt{1-\rho^2} \end{pmatrix} \begin{pmatrix} \sigma_1 & \rho\sigma_2 \\ 0 & \sigma_2\sqrt{1-\rho^2} \end{pmatrix} = \begin{pmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2 \end{pmatrix}$$

2.8 General Multi-Factor Models

In general, given any size of covariance matrix for any number of underlying assets it will be possible to generate correlated normally distributed random variables. For a covariance matrix for two underlyings the correlated random variables, y_1, y_2, say were obtained from

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = M \begin{pmatrix} \phi_1 \\ \phi_2 \end{pmatrix}$$

where M is the square root of the covariance matrix and ϕ are the independent Normally distributed random variables. In general to create d-correlated Normally distributed variables $y_1, ..., y_d$ from d uncorrelated variables $\phi_1, ...\phi_d$ and $MM^T = \Sigma =$ covariance matrix, use

$$\begin{pmatrix} y_1 \\ . \\ . \\ y_d \end{pmatrix} = M \begin{pmatrix} \phi_1 \\ . \\ . \\ \phi_d \end{pmatrix}$$

Cholesky factorisation

In general calculating the matrix M is fairly straightforward, one way of doing so is by using a process called Cholesky factorisation. Cholesky factorisation is a special case of LU decomposition, specialised to symmetric, positive definite matrices. See Paul Wilmott on Quantitative finance pp934-5 for the computer algorithm or alternatively Press et al., numerical recipes in Fortran/C etc.

The calculation of M in a fast and efficient way has been a topic of research over the last 20 or so years and there are many numerical packages available that can solve this problem for you.

2.9 Summary

In general Monte-Carlo methods are simple to program and to understand, so they are ideal for a first approximation to a derivative value. However the convergence is slow, and the value is determined probabilistically which makes extrapolation impossible. These methods naturally fit with forward looking problems making the pricing of path dependent derivatives such as lookback options and Asian options easy to implement. They tend to also be good for derivatives where there are multiple sources of uncertainty, as the computational effort only increases linearly.

We have introduced Monte Carlo methods which are very closely related to the probabilistic solution, in that you use simulation to determine an expected value for the option in the future that can be discounted at the risk-free rate (according to the fundamental theorem) to obtain the value today. To simulate the paths we typically use the solution to the SDE or the Euler approximation, along with a decent generator of Normally distributed random variables. It is easy to apply to path dependent options and to options on more than one underlying, for these you need to know the covariance matrix and be able to 'square root' it by means of Cholesky factorisation to be able to perform the simulations.