Computational Finance with C++

Lecture 5: Monte Carlo Methods

Panos Parpas
Imperial College London
p.parpas@imperial.ac.uk

Outline

- Path Dependent Options
- Option Valuation with Monte Carlo
- Error estimation
- Estimating Greeks
- Variance Reduction
- Multidimensional options: Path Dependent basket options
- Reading:
  - ✦Chapter 5 Capinski+Zastawniak,Numerical Methods in finance with C++

Market Setup

- Market with two assets
    - Risk Free Asset Dynamics: $A(t) = e^{rt}$
    - Risky Asset Dynamics: $S(t) = S(0)e^{(r-\sigma^2/2)t+\sigma W_Q(t)}$

where r is the risk free rate
σ is volatility
$W_Q(t)$ is a Wiener process.

Path Dependent Options

- Let T be the expiry date
- Let

$$t_k = \frac{k}{m} T \text{ for } k = 1, \ldots, m$$

A Path dependent option has the payoff:

$$H(T) = h(S(t_1), \ldots, S(t_m))$$

where $h : \mathbb{R}^m \to \mathbb{R}$ is a given deterministic function.

Example: Arithmetic Asian Call Option

$$h(z_1, \ldots, z_m) = \max\left(\frac{1}{m}\sum_{k=1}^{m} z_i - K, 0\right)$$

Path Dependent Options

- Let H(0) denote the price of the path dependent option
- Price determined under the risk neutral measure Q

$$H(0) = e^{-rT}\mathbb{E}_Q(H(T))$$

- Apart from some easy cases (e.g. European options) the expectation can only be estimated using Numerical Methods
- **Monte Carlo Methods** by far the most widely used

Path Simulation

- We know that

  W_Q has independent increments
  $W_Q(t) - W_Q(s)$ has a Normal distribution $N(0, t-s)$

- Using the facts above:

$$S(t_k) = S(t_{k-1}) e^{\left(r - \frac{1}{2}\sigma^2\right)(t_k - t_{k-1}) + \sigma\sqrt{t_k - t_{k-1}} Z_k}$$

where $Z_1, \ldots, Z_m$ are independent identically distributed random variables $N(0,1)$

Path Simulation

- Let
  $\hat{Z}_1, \ldots \hat{Z}_k$ be a sequence independent samples from $Z_1, \ldots Z_k.$
- Generate a sequence:

$$\left(\hat{S}(t_1), \ldots, \hat{S}(t_m)\right)$$

Where

$$\hat{S}(t_1) = S(0)e^{\left(r-\frac{1}{2}\sigma^2\right)t_1+\sigma\sqrt{t_1}\hat{Z}_1}$$

$$\hat{S}(t_k) = \hat{S}(t_{k-1})e^{\left(r-\frac{1}{2}\sigma^2\right)(t_k-t_{k-1})+\sigma\sqrt{t_k-t_{k-1}}\hat{Z}_{t_k}}$$

is called a **sample path**

Convergence

Let $(\hat{S}^i(t_1), \ldots S^i(t_m))$, for $i \in \mathbb{N}$ be a sequence of independent sample paths

By the law of large numbers:

$$\mathbb{E}_Q(h(S(t_1), \ldots, S(t_m)) = \lim_{N \to \infty} \frac{1}{N} \sum_{i=1}^{N} h(\hat{S}^i(t_1), \ldots, \hat{S}^i(t_m))$$

For sufficiently large N we can find an approximation for H(0)

$$H(0) \approx \hat{H}_N(0) = e^{-rT} \frac{1}{N} \sum_{i=1}^{N} h(\hat{S}^i(t_1), \ldots, \hat{S}^i(t_m))$$

Generating Normal Random Variables with C++

- In C++ we have `rand()` generates integer uniform in 0 to `RAND_MAX`
- Use Box-Muller theorem to generate N(0,1)

Theorem: If U_1, U_2 are independent random variables with uniform distribution in $[0,1]$ then the random variable

$$Z = \sqrt{-2\ln(U_1)}\cos(2\pi U_2)$$

has distribution $N(0,1)$

Generating Normal Random Variables with C++

1. Generate two integer numbers and rescale them to be in [0,1]

$$\hat{U}_i = \frac{k_i + 1}{RAND\_MAX + 1}, \ i = 1, 2.$$

2. Compute a sample from N(0,1):

$$\hat{Z} = \sqrt{-2\ln(\hat{U}_1)}\cos(2\pi\hat{U}_2)$$

3. Repeat the procedure until the whole path is generated.

Implementation: `BSModel01.h`, `BSModel01.cpp`

- Use of `vector` from STL class to store sample path
- Note use of `typedef` to make code easier to read
- Parameters stored `public` for ease, in a real implementation should be `private` and use set/get functions to change
- Random numbers cannot be generated by a deterministic method!
- C++ uses a seed to achieve this
- For debugging it is easier to use the same seed
- Note that we pass the sample path by reference to speed up computation

Implementation: `PathDepOption01.h`, `PathDepOption01.cpp`

- PriceByMC() main pricing function, note use of running average
- Payoff() is a virtual function
- ArithmAsianCall is a concrete implementation for pricing an arithmetic Asian option
- Note how SamplePath is passed by reference

Exercise 5.1: Add a class to the code described above to compute the price of European call and put options.
You should verify that your code produces the same price as the Black and Scholes formula

Pricing Error Estimation

- The more sample paths we generate the more accurate the estimate.
- How does the error depend on the number of samples?
- **Standard Error:** the standard deviation of the sample mean, divided by the number of samples:

$$\hat{\sigma}_N = \sqrt{\frac{1}{N}}\sigma_e$$

where σ_e is an unbiased estimator of the sample standard deviation :

$$\sigma_e^2 = \frac{1}{N-1}\sum_{i=1}^{N}(e^{(-rT)}(\hat{H}^i(T) - \hat{H}_N(0))^2$$

Pricing Error Estimation

Equation for standard error, can be rewritten as:

$$\hat{\sigma}_N = \frac{e^{-rT}}{\sqrt{N-1}} \sqrt{\frac{1}{N}\sum_{i=1}^{N} \hat{H}^i(T)^2 - \left(\frac{1}{N}\sum_{i=1}^{N} \hat{H}^i_N(T)\right)^2}$$

If N is large, the law of large numbers can be used to replace sample averages with exact values:

$$e^{rT}\sqrt{N-1}\hat{\sigma}_N \approx \sqrt{\mathbb{E}H(T)^2 - (\mathbb{E}H(T))^2} = \sqrt{\mathrm{Var}(H(T))}$$

To reduce the error by one decimal point we need to make about 100 times more simulations:

$$\hat{\sigma}_{100N} \approx \frac{1}{10}\hat{\sigma}_N$$

Implementation: `PathDepOptions02.h` `PathDepOptions02.cpp`

- Modified code to estimate errors
- `Price` and `PriceError` become members of the class
- Note that `H` and `Hsq` can be computed simultaneously

Greek Parameters

Denote the option price at time 0 as: $H(0) = u(S(0))$

Delta parameter is defined as: $\delta = \frac{du}{dz} u(S(0))$

Can use finite difference approximation

$$\frac{du}{dz} u(S(0)) \approx \frac{u((1+\epsilon)S(0)) - u(S(0))}{\epsilon S(0)}$$

Use Monte Carlo to estimate perturbed price

$$\begin{aligned} u((1+\epsilon)S(0)) &= e^{-rT} \mathbb{E}_Q h((1+\epsilon)(S(t_1), \ldots, S(t_m))) \\ &\approx e^{-rT} \frac{1}{N} \sum_{i=1}^{N} h((1+\epsilon)(\hat{S}^i(t_1), \ldots, \hat{S}^i(t_m))) \\ &:= \hat{H}_{\epsilon,N}(0) \end{aligned}$$

Greek Parameters

Final Delta approximation is given by:

$$\delta \approx \hat{\delta} = \frac{\hat{H}_{\epsilon,N}(0) - \hat{H}_N(0)}{\epsilon S(0)}$$

Other greek parameters are estimated in the same way

Implementation: `PathDepOption03.h`, `PathDepOption03.cpp`

- Add new members for delta and epsilon
- Rescale sample path

Exercise 5.2: Use the fact that:

$$\frac{d^2u}{dz^2}(S(0)) = \frac{u((1+\epsilon)S(0)) - 2u(S(0)) + u((1-\epsilon)S(0))}{(\epsilon S(0))^2}$$

to expand the code for delta hedging from the previous slide to compute the gamma parameter.

Exercise 5.3:Expand the code from the previous slide to compute other Greek parameters such as vega, theta and rho.

Variance Reduction

Control Variate method: $\mathbb{E}(X) = \mathbb{E}(X - Y) + y$

Where $y = \mathbb{E}(Y)$, and Y is a random variable that is 'close' to X

- **Control Variates** offer potential variance reduction without increasing the number of simulations

Exercise 5.4 Let Z be a random variable with normal distribution $N(0, \frac{1}{4})$. Let $X = \cos(Z)$ and $Y = 1 - \frac{1}{2}Z^2$. Using the fact that

$$\mathbb{E}(Y) = 1 - \frac{1}{2}\mathbb{E}(Z^2) = \frac{7}{8}$$

write a program which computes $\mathbb{E}(X)$, using Y as a control variate.

The idea behind the choice of Y as a control variate follows from the fact that by Taylor's expansion of $\cos(z)$, for small z,

$$\cos(z) \approx 1 - \frac{1}{2}z^2.$$

Impelementation of Control Variates: `PathDepOption04.h/cpp`

$G(T)$ is the control variate
Assume we can compute $\mathbb{E}_Q(G(T))$ analytically
Modify code to compute $e^{-rT}\mathbb{E}_Q(H(T) - G(T)) + G(0)$

- Add `PriceByBSFormula` and use in `PriceByVarRedMC`
- Note use of `this` keyword in `PariceByVarRedMC`
- Example: Pricing an arithmetic Asian Call option with Geometric Asian Call.
- Payoff for Geometric Asian Call:

$$g(z_1, \ldots, z_m) = h^{gac}(z_1, \ldots, z_m) = \max\left(\sqrt{\Pi_{k=1}^{m} z_k} - K, 0\right)$$

Example: Pricing an arithmetic Asian Call option with Geometric Asian Call.

- Payoff for Geometric Asian Call:

$$g(z_1, \ldots, z_m) = h^{gac}(z_1, \ldots, z_m) = \max\left(\sqrt{\Pi_{k=1}^{m} z_k} - K, 0\right)$$

- Analytical formula for Geometric Asian Call:

$$G(T) = \max(ae^{\left(r-\frac{1}{2}b^2\right)T+b\sqrt{T}Z} - K, 0)$$

Where Z is N(0,1) and a,b are constants defined in text/code

- Implementation in `GmtrAsianCall.h/cpp Main22.cpp`

Control Variates to compute Greeks

- Example with delta calculation: $\delta_H = \delta_{H-G} + \delta_G$
- Where

$$\delta_H = \frac{du_H}{dz}(S(0)) \quad \delta_G = \frac{du_G}{dz}(S(0))$$

$$\delta_{H-G} = \frac{d(u_H - u_G)}{dz}(S(0))$$

Exercise 5.5 Expand the code in previous slides to compute δ of an arithmetic Asian call. Use the fact that the δ of the geometric Asian call is $N(d_+^{a,b})\frac{a}{S(0)}$, where

$$d_+^{a,b} = \frac{\ln(\frac{a}{K}) + (r + b^2/2)T}{b\sqrt{T}}$$

Exercise 5.6 Price a barrier call option with payoff function

$$h^{bc}(z_1, \ldots, z_m) = \mathbf{1}_{\{\max\limits_{k=1,\ldots,m} z_k \leq L\}} \max(z_m - K, 0)$$

Using a call option with payoff function

$$g(z_1, \ldots, z_m) = h^c(z_1, \ldots, z_m) = \max(z_m - K, 0)$$

as a control variate. Also compute δ using the method described in the lecture.

Exercise 5.7 Price an arithmetic Asian call with payoff

$$H(T) = \max\left(\frac{1}{T}\int_0^T S(t)dt - K, 0\right)$$

using

$$G(T) = \max\left(\exp\left(\frac{1}{T}\int_0^T \ln(S(t))dt\right), 0\right)$$

as a control variate. The price of the control variate can be computed using the fact that,

$$G(T) = \max\left(c_1 e^{(r-c_2^2/2)T + c_2\sqrt{T}Z} - K, 0\right)$$

where Z has standard normal distribution $N(0,1)$ and $c_1, c_2 \in \mathbb{R}$ are constants:

$$c_1 = S(0)e^{-\frac{1}{2}(r+\sigma^2/6)T}, \quad c_2 = \frac{\sigma}{\sqrt{3}}$$

Path-dependent basket options

Suppose we have d underlying assets $S(t) = \begin{bmatrix} S_1(t) \\ \vdots \\ S_d(t) \end{bmatrix}$

The prices have the following dynamics:

$$S_j(t) = S_j(0)\exp\left(\left(r - \frac{\sigma_j^2}{2}\right)t + \sum_{l=1}^{d} c_{jl} W_l(t)\right)$$

where $W_1(t), \ldots, W_m(t)$ are independent Wiener processes. C is $d \times d$ correlation matrix, and

$$\sigma_j = \sqrt{c_{ji}^2 + \ldots + c_{jd}^2}$$

for $j = 1, \ldots, d$.

Path-dependent basket options

Path dependent option payoff: $H(T) = h(S(t_1), \ldots, S(t_m))$
$\text{where } h : \mathbb{R}^d \times \ldots \times \mathbb{R}^d \to \mathbb{R}$

Arithmetic Asian basket call:

$$H(T) = \max\left(\left(\sum_{j=1}^{d}\left(\frac{1}{m}\sum_{k=1}^{m} S_j(t_k)\right) - K\right), 0\right)$$

Data Structures for Basket Options: `Matrix.h/cpp`

- `Matrix.h` defines the operators we need to use
- Note use of ^ since other operators are taken
- Use of `const` to ensure no unintended changes are made

- Implementation in `PathDepOption05.h/cpp` is similar to previous definitions
- Note how easy coding is with the correct definitions
- `Main23.cpp` implements a 3d basket option

Exercise 5.8 Expand the code for path dependent basket options to calculate the error.

Computing Greeks for Basket Options

Let $u(S(0)) = u(S_1(0), \ldots, S_d(0))$ denote the price of the option

The delta with respect to the j^{th} asset is defined as

$$\delta_j = \frac{\partial u}{\partial z_j}(S_1(0), \ldots, S_d(0)) \text{ for } j = 1, \ldots, d.$$

Let ϵ_j denote a vector $\mathbb{R}^d$ of zeros where the j^{th} coordinate is ϵ

Perturbed Price Vector: $(1+\epsilon_j)\hat{S}(t_k) = \begin{bmatrix} \hat{S}_1(t_k) \\ \vdots \\ \hat{S}_{j-1}(t_k) \\ (1+\epsilon_j)\hat{S}_j(t_k) \\ \hat{S}_{j+1}(t_k) \\ \vdots \\ \hat{S}_d(t_k) \end{bmatrix}$

Computing Greeks for Basket Options

Perturbed Price Vector:

$$\hat{H}_{\epsilon_j,N} = e^{-rT}\frac{1}{N}\sum_{i=1}^{N} h((1+\epsilon_j)\hat{S}^i(t_1),\ldots,(1+\epsilon_j)\hat{S}^i(t_m))$$

Approximate delta for j-th asset is given by:

$$\delta_j \approx \hat{\delta}_j = \frac{H_{\epsilon_j,N}(0) - \hat{H}_N(0)}{\epsilon S_j(0)}$$

Exercise 5.9 Expand the code to implement the delta for basket options

Exercise 5.10 Write a class for pricing of a European basket call option with payoff

$$H(T) = \max\left(\left(\sum_{j=1}^{d} S_j(T) - K, 0\right), 0\right)$$

Exercise 5.11 Use

$$G(T) = \max\left(\left(\sum_{j=1}^{d} S_j(T) - K, 0\right), 0\right),\ K_j = \frac{K S_j(0)}{\sum_{l=1}^{d} S_l(0)}$$

as a control variate to reduce the pricing error for the European basket call from Exercise 5.10.