CMF MSU

Derivatives Pricing Course

Lecture 6 – Monte Carlo methods. Advanced topics.

Artem Isaev

2016 CMF

Agenda

- Discretization Schemes, Convergence, and Stability
- Variance Reduction Techniques
- Strong path-dependency
- C++ implementation

Monte Carlo methods

Convergence and stability

- For the simple Black-Scholes model, SDE state variables could be expressed analytically in terms of independent increments of a Brownian motion, making path generation straightforward.

- In practice however we are often working with SDEs that do not permit closed-form solution.

Monte Carlo methods

Convergence and stability

- Consider a vector-valued SDE:

$$dX(t) = \mu(t, X(t))dt + \sigma\big(t, X(t)\big)dW(t)$$

- Consider an equidistant time grid $\{0, \Delta, 2\Delta, \dots, m\Delta\}$.
  Set $\hat{X}_i \triangleq \hat{X}(i\Delta)$. We say that the underlying approximation is *weakly consistent* if there exists a function $c(\Delta)$ with

$$\lim_{\Delta \downarrow 0} c(\Delta) = 0$$

such that $E\left(\left|E\big(\Delta^{-1}(\hat{X}_{i+1} - \hat{X}_i)\big) - \mu\big(i\Delta, \hat{X}_i\big)\right|^2\right) \leq c(\Delta)$, and

$$E\left(\left|E\left(\Delta^{-1}\big(\hat{X}_{i+1} - \hat{X}_i\big)\big(\hat{X}_{i+1} - \hat{X}_i\big)^T\right) - \sigma\big(i\Delta, \hat{X}_i\big)\sigma\big(i\Delta, \hat{X}_i\big)^T\right|^2\right) \leq c(\Delta),$$

for all $i = 0, \dots, m-1$

Monte Carlo methods

Convergence and stability

- A concept related to consistency is the notion of *weak convergence*.

- We say that an approximate solution converges weakly at time $T = m\Delta$ with respect to a class C of test functions $g: \mathbb{R}^p \to \mathbb{R}$ if
$$\lim_{\Delta \downarrow 0} \left| E\big(g(X(T))\big) - E\left(g\big(\hat{X}(T)\big)\right) \right| = 0$$
for all $g \in C$.

- ${C_P}^l$ - functions with polynomially bounded derivatives of order $0, 1, \ldots, l$. We say that a scheme converges with *weak order* β if, for all $g \in {C_P}^{2(\beta+1)}$:
$$\left| E\big(g(X(T))\big) - E\left(g\big(\hat{X}(T)\big)\right) \right| \le c\Delta^\beta$$

for all $\Delta \in (0, \Delta_0)$, where Δ_0 and c are constants.

Monte Carlo methods

Examples – Euler-Maryama

- The most straightforward. Starting from $\hat{X}_0 = X(0)$,
$$\hat{X}_{i+1} = \hat{X}_i + \mu\left(i\Delta, \hat{X}_i\right)\Delta + \sigma\left(i\Delta, \hat{X}_i\right)(W(i\Delta + \Delta) - W(i\Delta)),$$
$$i = 0,1, \dots, m - 1$$

- The most widely used and easy to implement.
- The scheme is weakly consistent.
- Given that the Euler scheme is fully explicit, our experience from finite difference methods suggests that the scheme may have stability problems – there are thus restrictions on how big a time step Δ can be used.

Monte Carlo methods

Examples – Log-Euler

- One potential problem with the pure Euler scheme is the fact that all increments are locally Gaussian, thereby implying a non-zero probability of $\hat{X}$ crossing zero and becoming negative.

- Many SDEs, however, are known to produce only non-negative solutions. For instance:

$$dX(t) = \sqrt{X(t)}dW(t), X(0) > 0$$

- Using the log-transform to the initial SDE:

$$\hat{X}_{i+1} = \hat{X}_i \exp\left(\left(\frac{\mu(t,\hat{X}_i)}{\hat{X}_i} - \frac{1}{2}\frac{\sigma(t,\hat{X}_i)^2}{\hat{X}_i{}^2}\right)\Delta + \frac{\sigma(t,\hat{X}_i)}{\hat{X}_i} Z_i\sqrt{\Delta}\right)$$

Monte Carlo methods

Bias vs. Error

- When we use an m-step discretization scheme in an n-path Monte Carlo run, we are exposed to two types of errors on the expectation we are trying to estimate:

  i. the statistical Monte Carlo error e_s

  ii. a bias e_b, originating from the discretization scheme

- Raising n will reduce the standard error, but will not affect the bias which can only be reduced by increasing the number of steps *m*.

- Assume that the discretization scheme has weak order β. Informally:

$$e_b = c_b \Delta^{\beta}$$

Monte Carlo methods

Bias vs. Error

- Also, we know that the variance of e_s:

$$\mathrm{Var}(e_s) = \frac{c_s}{n}$$

- Consider a problem of minimizing MSE:

$${c_b}^2 \Delta^{2\beta} + \frac{c_s}{n}$$

- Omitting several assumptions – when we increase or decrease our computing budget, it is thus reasonable to allocate resources in such a way that we keep the factor $n^{1/2} m^{-\beta}$ constant.

Monte Carlo methods

Variance Reduction Techniques

- The convergence of the Monte Carlo method is quite slow, of order $O\left(n^{-1/2}\right)$.

- While there is little that can be done to improve the order itself, the constant multiplying $n^{-1/2}$ can be affected by a careful choice of the Monte Carlo estimator.

- Recall that the goal of the Monte Carlo method is to estimate some quantity μ as the sample mean of n i.i.d. random variables $Y_1, \dots, Y_n$ where each Y_i has expectation $E(Y_i) = \mu$ and variance $\mathrm{Var}(Y_i) = \sigma^2$

Monte Carlo methods

Variance Reduction Techniques

- Suppose we have available two sets of i.i.d. sequences $Y_{1,i}$ and $Y_{2,i}$ where $E\left(Y_{1,i}\right) = E\left(Y_{2,i}\right) = \mu$, but $\mathrm{Var}\left(Y_{1,i}\right) = {\sigma_1}^2$ and $\mathrm{Var}\left(Y_{2,i}\right) = {\sigma_2}^2$.

- Also suppose that the time it takes on a computer to generate individual samples $Y_{1,i}$ and $Y_{2,i}$ is τ_1 and τ_2. Fixed computing time budget - τ.

- Thus we need to compare

$$ {\sigma_1}^2\tau_1 \text{ vs } {\sigma_2}^2\tau_2 $$

- For instance, a high-variance estimator may, in fact, be preferable to a low-variance estimator, provided that the former takes less time to compute that the latter.

Monte Carlo methods

Antithetic variates

- Assume that we are interested in estimating the expected value of a random variable $Y = G(Z), Z -q$-dimensional vector of independent Gaussian random variables.

- Rather than using the regular sample average estimator for $E(Y)$, consider instead using

$$\bar{Y}_n{}^a = n^{-1} \sum_{i=1}^{n} \frac{G(Z_i) + G(-Z_i)}{2}$$

- In other words, in addition to set $Z_1, \dots, Z_n$ of Gaussian samples, we also effectively include the set $-Z_1, \dots, -Z_n$ in the Monte Carlo trial. We still must have

$$E\left(\bar{Y}_n{}^a\right) = E(Y)$$

Monte Carlo methods

Antithetic variates

- Also, as $G(Z)$ and $G(-Z)$ have identical variance

$$\mathrm{Var}\left(\bar{Y}_n{}^{a}\right) = n^{-1}\left[\frac{1}{2}\mathrm{Var}(Y) + \frac{1}{2}\mathrm{Cov}\big(G(Z), G(-Z)\big)\right] = \frac{\mathrm{Var}(Y)}{n}\frac{(1+\rho)}{2}$$

- Recalling that the regular sample average has variance

$$\mathrm{Var}(\bar{Y}_n) = \frac{\mathrm{Var}(Y)}{n}$$

- While use of antithetic variates can always be expected to lower the standard error, it is not necessarily more efficient.

Monte Carlo methods
Control variates

- While we may need to use Monte Carlo simulation to estimate the unknown mean of a random variable Y, there may be random variables “close” to Y with means that can be computed analytically.
- Formally, vector of control variates, ideally with a strong negative or positive correlation with Y

$$Y^c = \left(Y_1{}^c, \dots, Y_q{}^c\right)^T$$

- The mean is known to be

$$E(Y^c) = \mu^c = \left(\mu_1{}^c, \dots, \mu_q{}^c\right)^T$$

- Arbitrary constant vector

$$\beta = \left(\beta_1, \dots, \beta_q\right)^T$$

Monte Carlo methods
Control variates

- And consider forming the linear combination
$$X = Y - \beta^T (Y^c - \mu^c)$$
- Clearly
$$E(X) = E(Y) - \beta^T (E(Y^c) - \mu^c) = E(Y)$$
- Σ_{Y^c} - $q \times q$ covariance matrix of the vector Y^c
- Σ_{Y,Y^c}- q-dimensional vector of covariances between Y and Y^c
- Variance of X
$$\mathrm{Var}(X) = \mathrm{Var}(Y) - 2\beta^T \Sigma_{Y,Y^c} + \beta^T \Sigma_{Y^c} \beta$$

- <u>Lemma 6.1.</u> The variance of X is minimized at
$$\beta^* = {\Sigma_{Y^c}}^{-1} \Sigma_{Y,Y^c}$$

with minimum value $(1 - R^2)\mathrm{Var}(Y)$

Monte Carlo methods

Importance Sampling

- For a given measure P, consider estimating
$$\mu = E^{P}(Y)$$
- Let $\hat{P}$ be a measure equivalent to P
$$\mu = E^{\hat{P}}(Y/R)$$

  where R is the Radon-Nikodym derivative
$$R = \frac{d\hat{P}}{dP}$$

- It is possible that variance of Y/R under measure $\hat{P}$ is lower than the variance of Y under P.

Monte Carlo methods

Importance Sampling – Density Formulation

- Specifically, let us assume that Y can be represented as $g(X)$, where $g: \mathbb{R}^p \to \mathbb{R}$ is a well-behaved function and X is p – dimensional with probability density $f: \mathbb{R}^p \to \mathbb{R}$

$$\mu = E^P\big(g(X)\big) = \int g(x) f(x) dx$$

to which corresponds an estimator

$$\bar{\mu}_n = \frac{1}{n} \sum_{i=1}^{n} g(X_i)$$

- Let h be another density, such that $h(x) > 0$ where $f(x) > 0$

$$\mu = \int g(x) \frac{f(x)}{h(x)} h(x) dx$$

Monte Carlo methods

Importance Sampling – Density Formulation

- So-called *likelihood ratio* $l(x) = f(x)/h(x)$. New estimator

$$\overline{\mu}_n{}^h = \frac{1}{n}\sum_{i=1}^{n} g(X_i)\frac{f(X_i)}{h(X_i)}$$

- Investigate the variances

$$\mathrm{Var}\left(\overline{\mu}_n{}^h\right) = \frac{1}{n}\left[E^{\hat{P}}\left(g(X)^2\frac{f(X)^2}{h(X)^2}\right) - \mu^2\right]$$

$$= \frac{1}{n}\left[E^{P}\left(g(X)^2\frac{f(X)}{h(X)}\right) - \mu^2\right]$$

and

$$\mathrm{Var}\left(\overline{\mu}_n\right) = \frac{1}{n}[E^P(g(X)^2) - \mu^2]$$

Monte Carlo methods

Importance Sampling – Density Formulation

- A good choice of likelihood density will sample in proportion to f and g.

- That is, values of X where both density $f(X)$ and the payout $g(X)$ are high should be assigned a high value of $h(X)$ (high "importance"), and values of X where either $f(X)$ or $g(X)$ (or both) are low should be assigned a low value of $h(X)$.

Monte Carlo methods

Arithmetic Asian option

- Consider a dividend-free stock S, with BS dynamics:
$$dS(t)/S(t) = r\,dt + \sigma\,dW(t)$$
- Let there be given an increasing set of observations times $\{t_1, t_2, \dots, t_m\}$, with $t_m = T$, and define the stock average:
$$A(T) = \frac{1}{m}\sum_{i=1}^{m} S(t_i)$$
- An *Asian* call option with strike K is defined by the terminal payout
$$g(T) = (A(T) - K)^+$$

we wish to price this option by Monte Carlo simulation.

Monte Carlo methods

Arithmetic Asian option

- In this model stock price can be written explicitly:

$$S(t) = S(0)e^{rt-\frac{1}{2}\sigma^2 t+\sigma W(t)}, t > 0$$

whereby, with $\Delta_i \triangleq t_i - t_{i-1}$ and $t_0 = 0$

$$S(t_i) = S(t_{i-1})\exp\left(\left[r - \frac{1}{2}\sigma^2\right]\Delta_i + \sigma[W(t_i) - W(t_{i-1})]\right)$$

- Recall the properties of a Brownian motion:

$$S(t_i) = S(t_{i-1})\exp\left(\left(r - \frac{1}{2}\sigma^2\right)\Delta_i\right)\exp\left(\sigma\sqrt{\Delta_i}Z_i\right)$$

- To produce a single sample draw of $g(T)$, we thus
  1. Draw independent standard Gaussian samples Z_i.
  2. Starting from $S(0)$, generate $S(t_i)$ from the iteration.
  3. Compute $g(T)$

Monte Carlo methods

Arithmetic Asian option

- Repeating the procedure n times, we can generate random samples $g_1, g_2, \dots, g_n$ of $g(T)$.
- Therefore our estimate of an option value:

$$\overline{V}(0) = e^{-rT} \frac{1}{n} \sum_{j=1}^{n} g_j$$

- Computational cost of the pricing algorithm - $O(mn)$.
- Now consider a p-dimensional basket of stocks $S_1, S_2, \dots, S_p$ each following geometric Brownian motion.
- The Brownian motions W_k and W_j are assumed to be correlated with constant correlation coefficient $\rho_{j,k}, j \neq k$.
- Define a unit-weighted basket price as

$$B(t) = \sum_{k=1}^{p} S_k(t)$$

Monte Carlo methods

Basket Asian option

- And set the terminal Asian option payout to be

$$g(T) = \left(\frac{1}{m}\sum_{i=1}^{m} B(t_i) - K\right)^+$$

- We draw sample paths

$$S_k(t_i) = S_k(t_{i-1})\exp\left(\left(r - \frac{1}{2}\sigma_k{}^2\right)\Delta_i\right)\exp\left(\sigma_k\sqrt{\Delta_i}Z_{k,i}\right)$$

- where $Z_{k,i}$ are Gaussian samples, independently drawn at each time step but correlated across k's.
- Let C be the Cholesky decomposition of the correlation matrix, in which case we can generate the correlated sample vectors $Z_i = \left(Z_{1,i}, Z_{2,i}, \dots, Z_{p,i}\right)^T$ as $Z_i = CX_i$ for vector X_i of independent Gaussian samples.

Monte Carlo methods

Basket Asian option

- The last steps are the same as for an Asian option on a single stock.

- The total computational effort of an n-sample Monte Carlo scheme is $O(nmp)$, with the convergence order $O(n^{-\frac{1}{2}})$ and dependent only on n.

C++ code

Random(.h) (1/2)

```
#ifndef RANDOM_H
#define RANDOM_H

//Abstract class - define the interface, not all member functions are pure virtual
class RandomGenerator{
public:
            //Simulate array of standard normal variables
            virtual void getStdNormalInverse(std::vector<double>& arrayToFillGauss);
            //Simulate array of uniform variables
            virtual void getUniform(std::vector<double>& arrayToFillUni) = 0;
            int getSize() { return Size; }
            void setSize(int newSize) { Size = newSize;}
            virtual RandomGenerator* clone() const = 0;
            RandomGenerator(unsigned long Size_): Size(Size_){}
private:
            //Dimension of the array
            int Size;
};


class BasicRandomGenerator : public RandomGenerator{
public:
            virtual void getUniform(std::vector<double>& arrayToFillUni);
            BasicRandomGenerator(unsigned long Seed_, int Size_ = 1) : RandomGenerator(Size_), Seed(Seed_){}
            virtual RandomGenerator* clone() const;
            void resetSeed();
private:
            //Seed of the random generator
            unsigned long Seed;
};
```

C++ code

Random(.cpp) (1/2)

```
//Using random engine from boost
#include <boost/math/distributions/inverse_gaussian.hpp>
#include <boost/random.hpp>
void RandomGenerator::getStdNormalInverse(std::vector<double>& arrayToFillGauss)
{
            //Gaussian inverse function from boost
            boost::math::normal distr(0.0, 1.0);
            std::vector<double> arrayToFillUni;
            arrayToFillUni.resize(arrayToFillGauss.size());

            getUniform(arrayToFillUni);
            for (int i = 0; i < arrayToFillGauss.size(); i++){
                        arrayToFillGauss[i] = quantile(distr, arrayToFillUni[i]);
            }
            Size = arrayToFillGauss.size();
}

void BasicRandomGenerator::getUniform(std::vector<double>& arrayToFillUni)
{
//What happens if they are not static?
//Uniform engine (Mersenne twister) from boost
static boost::mt19937 generator(Seed);
static boost::uniform_real<> range(0, 1);
static boost::variate_generator<boost::mt19937&, boost::uniform_real<> > uniform(generator, range);
            for (int i = 0; i < arrayToFillUni.size(); i++){
                        arrayToFillUni[i] = uniform();
            }
}
```

C++ code

PathDependentAsian(.h)

```
#ifndef PATHDEPENDENTASIAN_H
#define PATHDEPENDENTASIAN_H

#include "Bridge.h"
#include <vector>

class PathDependentAsian{
public:
            PathDependentAsian(double Time_, const Bridge& ThePayOff_, std::vector<double> averagingTimes_);
            double optionPayoff(double Spot) const;
            double getTime() const;
            std::vector<double> getAveragingTimes() const;
private:
            double Time;
            Bridge thePayoff;
            std::vector<double> averagingTimes; //at these dates from today underlying values will be fixed
};
#endif
```

C++ code

PathDependentAsian(.cpp)

```
PathDependentAsian::PathDependentAsian(double Time_, const Bridge& ThePayOff_, std::vector<double> averagingTimes_):
Time(Time_), thePayoff(ThePayOff_)
{
	averagingTimes.swap(averagingTimes_);
}

double PathDependentAsian::optionPayoff(double Spot) const{
	return thePayoff(Spot);
}

double PathDependentAsian::getTime() const{
	return Time;
}

std::vector<double> PathDependentAsian::getAveragingTimes() const{
	return averagingTimes;
}
```

C++ code
ExoticMonteCarlo(.h)

```
#include "PathDependentAsian.h"
#include "Random.h"

double* ExoticMC(const PathDependentAsian& theOption,
            RandomGenerator& theGenerator,
            double Spot,
            double Vol,
            double Rate,
            int numberOfPaths);
```

C++ code

ExoticMonteCarlo(.cpp) (1/2)

```
#include "ExoticMonteCarlo.h"

double* ExoticMC(const PathDependentAsian& theOption,
            RandomGenerator& theGenerator,
            double Spot,
            double Vol,
            double Rate,
            int numberOfPaths)
{
            double* result = new double[1];
            Vol = Vol / 100.0; // in %
            Rate = Rate / 100.0; // in %
            //Euler scheme
            //choose time step equal to difference between averaging dates
            int numberOfDates = theOption.getAveragingTimes().size();

            double sum = 0.0;
```

C++ code

ExoticMonteCarlo(.cpp) (2/2)

```
//outer loop - Paths
for (int i = 0; i < numberOfPaths; i++){
   double averagePath = 0.0;
   double lastSpot = Spot;
   std::vector<double> gaussianVector;
   gaussianVector.resize(numberOfDates - 1);
   theGenerator.getStdNormalInverse(gaussianVector);
   //inner loop - sum underlying values on each date
   for (int j = 1; j < numberOfDates; j++){
      double precomputedSpot = lastSpot*exp((Rate - 0.5*Vol*Vol)*(theOption.getAveragingTimes()[j] -
                               theOption.getAveragingTimes()[j - 1]));
      double stdDeviation = Vol * sqrt(theOption.getAveragingTimes()[j] - theOption.getAveragingTimes()[j - 1]);
      double thisSpot = precomputedSpot*exp(stdDeviation*gaussianVector[j - 1]);
      lastSpot = thisSpot;
      averagePath += thisSpot;
   }
   averagePath = averagePath / (numberOfDates - 1);
   double payoffCalc = theOption.optionPayoff(averagePath);
   sum += payoffCalc;
}

result[0] = (sum / numberOfPaths) * exp(-Rate * theOption.getTime());

return result;
}
```

C++ code

BS_Pricer(.h)

```cpp
const int numberOfPaths = 10000000;

//Create option object, define payoff
PayoffCall thePayoff(Strike);


//State averaging times
//Daily - 252 business days

int length = round(Time * 252);

std::vector<double> averagingTimes;
averagingTimes.resize(length);

for (int i = 0; i < length; i++){
            averagingTimes[i] = i * (Time / (length - 1));
}

PathDependentAsian theOption(Time, thePayoff, averagingTimes);

BasicRandomGenerator theGenerator(121);
//Decorated generator
AntiThetic theDecoratedGenerator(&theGenerator);

double* result = ExoticMC(theOption, theDecoratedGenerator, Spot, Vol, Rate, numberOfPaths);
```

Monte Carlo methods

Antithetic variates results - variance

	N=100	N=400	N=900
Simple MC	0.9925	0.5520	0.2644
Antithetic	0.5263	0.2385	0.0983

Homework assignment 5*

- Modify program to price Geometric Weekly Asian Put option on a Basket of two stocks.