COMPUTATIONAL FINANCE

Lecture 4: Stochastic Volatility Option Pricing Model
Simulation Estimation of Option Prices

Philip H. Dybvig
Washington University
Saint Louis, Missouri

The Fundamental Theorem of Probability

The Fundamental Theorem of Probability says that if we draw a random variable independently again and again from a probability distribution, the sample mean of the random variable will approach the population mean. For example, consider a fair (50-50) coin and the random variable that is 1 given heads and 0 given tails. The Fundamental Theorem of Probability tells us that in a large number of independent coin flips, the mean of this random variable will tend to 1/2. This says that in a large sample, the proportion of heads tends to 1/2.

Expectations Can Be Estimated by Simulation

It has been said that the Fundamental Theorem of Probability makes statistical analysis possible. It also makes simulation possible: the sample mean of a random variable we simulate approximates the population mean as the simulation gets large.

Option Prices are Expectations: Single Period

Recall that we can write option prices in a one-period model as expectations. Absent dividends (which are valued separately in the same way)

$$V_0 = E^*[\frac{1}{r}V_1] \tag{1}$$

This assumes no dividends. E^* indicates expectations under the risk-neutral probabilities that equalize expected returns across assets, and r is one plus the riskless rate. Intuitively, investing in any asset is a fair gamble after discounting (by $1/r$) adjusts for time preference and changing from actual probabilities to risk-neutral probabilities adjusts for risk preferences.

Option Prices are Expectations: Multiple Periods

$$
\begin{aligned}
V_0 &= E_0^*[\frac{1}{r_0}V_1] \\
&= E_0^*[\frac{1}{r_0}E_1^*[\frac{1}{r_1}V_2]] \\
&= E_0^*[\frac{1}{r_0 r_1}V_2] \\
&= ... = E_0^*[\frac{1}{r_0 r_1 ... r_{T-1}}V_T]
\end{aligned} \tag{2}
$$

Option Prices are Expectations: Observations

1. If the the interest rate is nonrandom, we can take the discounting out of the expectation, and we simply discount the expected cash flows in the risk-neutral probabilities. In a simulation, we simulate the underlying processes using the risk-neutral probability distributions, compute the cash flows, and then take net present values.

2. If the interest rate moves randomly, we need to discount using the realized spot rates, and we take the expectation *after* discounting. *Do not* discount discount expected terminal cash flows at a multiperiod riskless rate.

3. For multiple cash flows, we can treat each maturity separately and add the results. Or, we can accumulate cash flows and reinvest them in the riskless asset which will give the same answer.

A Menagerie of Random Variables

Let double precision variables z_i (for example, each generated as `((double) rand()/(double) RAND_MAX)`) be independent pseudo-random numbers uniform on [0,1]. Then we can generate pseudo-random numbers with other distributions as follows:

1. $x_1 = a + (b - a)z_1$: uniform on the interval $[a, b]$
2. $x_2 = (z_1 < p?u : d)$, where $0 < p < 1$: Bernoulli random variable that is u with probability p and d with probability $1 - p$
3. $x_3 = (z_1 < p_1?u : (z_1 < p_1 + p_2?m : d))$, where $0 < p_1 < p_1 + p_2 < 1$: three-valued random variable giving u with probability p_1, m with probability p_2, and d with probability $p_3 = 1 - p_1 - p_2$.
4. $x_4 = \sqrt{-2.0 * \log(z_1)} * \cos(2 * \Pi * z_2)$ and $x_5 = \sqrt{-2.0 * \log(z_1)} * \sin(2 * \Pi * z2)$: two independent normal random variables each having mean zero and variance one. (The constant $\Pi \approx 3.1415926535$.)
5. $x_6 = z_1 + z_2 + z_3 + z_4 + z_5 + z_6 - z_7 - z_8 - z_9 - z_{10} - z_{11} - z_{12}$: a different variable approximately normal with mean zero and variance 1.

6. $x_7 = -k * \log(z_1)$: exponential distribution with mean k
7. $x_8 = m + s * x_4$: a normal random variable with mean m and standard deviation s and therefore variance s^2. (Obviously, we can use x_5 or x_6 in place of x_4.)
8. $x_9 = m + s * (z_1 - 0.5) + \sqrt{12.0}$: uniform random variable with mean of m and standard deviation s.

The Header File simu2.h

```
//
// simu2.h  Stochastic Volatility Option Pricing Model
//

class svprice {
  public:
    svprice(double ttm=0.25,int nper=12,double r=.05,double initstd=.15,
      double meanstd=.2,double k=6.0,double sigstd=.5);
    double eurcall(double stock,double strike,long int nsimu=1000);
  private:
    int npers;
    double tinc, r1per, stockP, sigma0, sigma, meansigma, sigmasigma, kappa,
      c0, c1, c2, c3, c4, c5;
    double stocktotret();};
```

The Test File simu2test.cc

```
//
// simu2test.cc Stochastic Volatility Option Pricing Test File
//
#include <iostream.h>
#include <math.h>
#include "simu2.h"

main() {
  long int i,nsimus;
  svprice sim2;
  for(nsimus=10;nsimus<=100000l;nsimus *= 10)
    cout << nsimus << " " <<
      floor(sim2.eurcall((double) 100.0,(double) 100.0,nsimus)*100.0+0.5)/100.0
      << endl;
  cout << endl;
  for(i=0;i<20;i++)
    cout << "100000 " <<
```

```
    floor(sim2.eurcall((double) 100.0,(double) 100.0,(long int) 1000001)*100.0+0.5)/100.0
    << endl;
return(0);}
```

The Implementation File simu2.cc: Part 1

```
//
// simu2.cc  Stochastic Volatility Option Pricing Model
//
#include <math.h>
#include <stdlib.h>
#include <iostream.h>
#include "simu2.h"
#define unifrand() ((double) rand()/((double) RAND_MAX))
#define MAX(a,b) (((a) > (b)) ? (a) : (b))
```

The Implementation File simu2.cc: Part 2

```
svprice::svprice(double ttm,int nper,double r,double initstd, double
    meanstd, double k, double sigstd) {
  npers = nper;
  tinc = ttm/(double) nper;
  r1per = 1.0 + r*tinc;
  sigma0 = initstd;
  meansigma = meanstd;
  sigmasigma = sigstd;
  kappa = k;
  c0 = kappa * tinc * meansigma;
  c1 = 1.0 - kappa * tinc;
  c2 = 1.0 - sigmasigma * sqrt(tinc)*0.5*sqrt((double) 12);
  c3 = sigmasigma * sqrt(tinc) * sqrt((double) 12);
  c4 = sqrt(tinc)*sqrt((double) 12);}
```

The Implementation File simu2.cc: Part 3

```
double svprice::eurcall(double stock,double strike,long int nsimu) {
  long int i,n;
  double x;
  x=0.0;
  for(n=0;n<nsimu;n++) {
    stockP = stock;
    sigma = sigma0;
    for(i=0;i<npers;i++) {
      stockP *= stocktotret();
      }
    x += MAX(stockP-strike,0);}
  return(x/((double) nsimu * pow(r1per,(double) npers)));}

double svprice::stocktotret() {
double stockret;
//
```

```
//  The following straightforward computations are algebraically the same as
//  the ones used below but are much slower because many more calculations are
//  performed in each pass through the loop.
//
//  stockret = r1per + sigma * sqrt(tinc) * (unifrand()-0.5) * sqrt((double) 12);
//  sigma = (kappa*tinc * meansigma + (1.0 - kappa * tinc) * sigma) *
//    (1 + sigmasigma * sqrt(tinc) * (unifrand()-0.5) * sqrt((double) 12));
//  return(stockret);}
//
  stockret = r1per + sigma * c4 * (unifrand()-0.5);
  sigma = (c0 + c1 * sigma) * (c2 + c3 * unifrand());
  return(stockret);}
```