COMPUTATIONAL FINANCE

Lecture 5: Black-Scholes Hedge Simulation

Philip H. Dybvig
Washington University
Saint Louis, Missouri

Black-Scholes Hedge Simulation

Before putting a hedge into place, it is a good idea to evaluate it first using Monte Carlo simulation (with pseudo-random numbers) and/or back-testing (using historical data). In this assignment, we are performing such a simulation for hedging using Black-Scholes. If you do the "Extra for Experts," your simulation will include transaction costs and a stock price process with stochastic volatility (for which Black-Scholes is not necessarily a good fit, even with frequent trading and without transaction costs).

The Header File gaussian.h

```
//
// gaussian.h
//
// declarations for normal computations
//
struct distdens {
  double dist;
  double dens;};
distdens Gaussian(double quantity,double mean=0.0,double variance=1.0);
```

The Implementation File gausstest.cc

```
//
//  Gaussian distribution test file
//
#include <iostream.h>
#include "gaussian.h"
main() {
  double quantity, mean, variance;
  distdens y;
  while(1) {
    cin >> quantity >> mean >> variance;
    y = Gaussian(quantity, mean, variance);
    cout << y.dist << " " << y.dens << endl;}}
```

The Implementation File gaussian.cc

```
//
// gaussian.cc
//
// C++ program for normal computations
// based on the algorithm contributed by Alan Miller
// contained in the Applied Statistics Algorithm as66
//    see http://lib.stat.cmu.edu/apstat/ for more information
//
#include <math.h>
#include "gaussian.h"
distdens Gaussian(double quantity,double mean,double variance) {
  distdens toreturn;
  double q,sigma,t;
  sigma = sqrt(variance);
  q = (quantity - mean)/sigma;
  toreturn.dens = exp(-q*q/2.0)*.3989422804014327/sigma;
  if(q>0.0) {
```

```
  t = 1.0/(1.0+0.2316419*q);
  toreturn.dist = 1.0-exp(-q*q/2.0)*.3989422804014327*
    t * (0.319381530 +
    t * (-0.356563782 +
    t * (1.781477937 +
    t * (-1.821255978 +
    t * 1.330274429))));}
else {
  t = 1.0/(1.0-0.2316419*q);
  toreturn.dist = exp(-q*q/2.0)*.3989422804014327*
    t * (0.319381530 +
    t * (-0.356563782 +
    t * (1.781477937 +
    t * (-1.821255978 +
    t * 1.330274429))));}
return(toreturn);}
```

The Test File bstest.cc

```
//
// bstest.cc Black-Scholes test program
//
#include <iostream.h>
#include "bs.h"
main() {
  double S,X,r,sigma,ttm;
  valhedge callinfo;
  bs bspr;
  while(1) {
    cin >> S >> X >> r >> sigma >> ttm;
    bspr.newpars(ttm,r,sigma);
    callinfo = bspr.call(S,X);
    cout << callinfo.value << " " << callinfo.delta << endl;}}
```

The Header File bs.h

```
//
// bs.h Black-Scholes header file
//
#include "gaussian.h"
struct valhedge {
  double value;
  double delta;};
class bs {
public:
  bs(double ttm=.25,double r=.05,double sigma=.3);
  void newpars(double ttm,double r,double sigma);
  valhedge call(double S0,double X);

private:
  valhedge ezcall(double stock,double bond,double vol);
  double disc,sigsqttm;};
```

The Implementation File bs.cc

```
//
// bs.cc Black-Scholes Pricing Engine
//
#include <math.h>
#include "bs.h"

bs::bs(double ttm,double r,double sigma) {
  sigsqttm = sigma * sigma * ttm;
  disc = exp(-r * ttm);}

void bs::newpars(double ttm,double r,double sigma) {
  sigsqttm = sigma * sigma * ttm;
  disc = exp(-r * ttm);}

valhedge bs::ezcall(double stock,double bond,double vol) {
  distdens Nn1,Nn2;
  valhedge toreturn;
```

```
  Nn1 = Gaussian(log(stock/bond)/vol+vol/2.0);
  Nn2 = Gaussian(log(stock/bond)/vol-vol/2.0);
  toreturn.value = stock * Nn1.dist - bond * Nn2.dist;
  toreturn.delta = Nn1.dist;
  return(toreturn);}

valhedge bs::call(double S0,double X) {
  return(ezcall(S0,X*disc,sqrt(sigsqttm)));}
```