Finite Difference Approximation of Black Scholes for European Options

Derek A. Williams

April 2, 2010

Abstract

We derive Black Scholes Equation using the Hedging Argument, present it's analytical solution for European Options, solve it numerically using Crank Nicolson with the Thomas Algorithm, then compare the numerical results against the analytic solution both qualitatively and quantitatively.

Contents

1 Introduction

Ever since it's inception in the hallmark paper, *The Pricing of Options and Corporate Liabilities*, the Black Scholes Equation for modelling the dynamics of derivative prices has become widely use in the area of Quantitative Finance. Aside from simply modelling the value of derivatives such as European, Bermudian, and American call and put options, this model is flexible enough that it can be generalized for other exotic securities including: bond options, interest rate caps, floors and swap options or swapations.

Despite it's inherent flexibility, analytic solutions to the Black Scholes Equation exist only under very limited circumstances i.e. European Call or Put Options. There are, however, many methods for obtaining results to Black Scholes under diverse circumstances using numerical techniques. While there exists two important techniques for obtaining numerical solutions to Black Scholes Equations for derivative securities valuation, Monte Carlo Method (MCM) and Finite Difference Method (FDM), the focus on this paper will be on the latter.

The purpose of this paper is not to break new ground in the area of Quantitative Finance but rather to present a comparison of the results of a ubiquitous numerical model against the analytic results of Black Scholes so as to gain a better understanding of the model and perhaps to begin the development of a computational platform with which to generate data for testing models of more complex financial instruments.

Although numerically solving the Black Scholes Equations for European Call and Put Options using this method seems rather redundant given that a closed form analytic solutions exist for both of these. The real interest lies in the fact that once the numerical model is well understood and functioning it can be refined to obtain numerical solutions for other financial instruments.

Consider American options, since an analytic solution for these options doesn't currently exist, numerical methods are the only alternative for pricing these options. It is often the case that different financial derivatives can be modelled by small modifications to the parameters in Black Scholes Equation or by changes to the boundary and initial conditions alone. Such refinements are easily added once a working numerical model is produced.

2 Analytic Solution

For the sake of brevity, the derivation of Black Scholes' Partial Differential Equation (PDE) using the Hedging Argument is presented along with analytic solutions for the European Call and Put Options.

2.1 Black Scholes Equation

The Black Scholes equation can be derived using the Hedging Argument in which an investor who has written a Call option with value $C(S,t)$ on the underlying asset S, purchases Δ shares of S in order to mitigate a potentially unlimited loss of wealth in the event that the value of the asset increases without bound. The investor's portfolio can be represented as follows:

$$\Pi = -C(S,t) + \Delta S$$

By considering small changes in the value of the portfolio:

$$d\Pi = -dC(S,t) + \Delta dS$$

If the value of the underlying asset follows a stochastic process with uncertainty component following a Geometric Brownian Motion such that

$$dS = \mu S dt + \sigma S dW$$

Employing Ito's Lemma it can be shown that

$$dC = \frac{\partial C}{\partial t}dt + \frac{\partial C}{\partial S}dS + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 C}{\partial S^2}dt$$

Hence for small changes in Π we can write

$$d\Pi = -(\frac{\partial C}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 C}{\partial S^2})dt + (\Delta - \frac{\partial C}{\partial S})dS$$

We can eliminate the uncertainty in the above equation by choosing $\Delta = \frac{\partial C}{\partial S}$. Thus

$$d\Pi = -(\frac{\partial C}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 C}{\partial S^2})dt$$

As such, one would expect to achieve at best a risk-free return: $d\Pi = r\Pi dt$ from such a portfolio hence,

$$d\Pi = r\Pi dt = r(-C(S,t) + \Delta S)dt = -(\frac{\partial C}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 C}{\partial S^2})dt$$

This gives the famous Black Scholes Equation:

$$\frac{\partial C}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 C}{\partial S^2} + rS\frac{\partial C}{\partial S} = rC \tag{1}$$

2.1.1 Final and Boundary Conditions

Equation (1) alone doesn't contain enough information to determine derivative prices; we require additional conditions as well. At maturity we expect that the value of a Call Option, which gives it's holder the right to purchase the underlying for K, is equal to the difference in the strike price K of the Option and the current value of the underlying, S. We also expect that the value of the option should be at least zero as the holder is under no obligation. It can be easily verified that the following conditions must be satisfied

$$C(S,T) = \max(S-K,0) \tag{2a}$$

$$C(0,T) = 0 \tag{2b}$$

A final condition is derived from considering the situation in which S increases very greatly.

$$\lim_{S\to\infty} \frac{\partial^2 C}{\partial S^2} = 0$$

or equivalently

$$\lim_{S\to\infty} \frac{\partial C}{\partial S} = N(\infty) = 1 \tag{2c}$$

2.2 Black Scholes Formulas

Black Scholes Equation, (1), can be solved together with final condition (2a) along with initial conditions (2b) and (2c) for the European Call Option by transforming it into a diffusion equation with the following substitutions

$$\tau = T - t \qquad u = Ce^{r\tau} \qquad x = \ln(\frac{S}{K}) + (r - \frac{\sigma^2}{s})\tau$$

We won't go into the full details here but under these variables Black Scholes Equation is reduced to the diffusion equation of the form

$$\frac{\partial u}{\partial \tau} = \frac{\sigma^2}{2}\frac{\partial^2 u}{\partial x^2}$$

and is readily solved using a Fourier Transform or Greens Functions.

2.2.1 Vanilla Call Option

As discussed in the preceding section solving equation (1) together with final condition (2a) and initial conditions (2b) and (2c). The solution to this equation for the European Call Option is

$$C(S,t) = SN(d_1) - Ke^{-r(T-t)}N(d_2) \tag{3}$$

where $N(z)$ is the Cumulative Normal Probability Distribution thus

$$N(z) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{z} e^{-\frac{\beta^2}{2}} d\beta \tag{4a}$$

$$d_1 = \frac{\ln(\frac{S}{K}) + (r - \frac{1}{2}\sigma^2)(T-t)}{\sigma\sqrt{T-t}} \tag{4b}$$

and

$$d_2 = d_1 - \sigma\sqrt{T-t} \tag{4c}$$

2.2.2 Vanilla Put Option

Using the Put-Call Parity relation

$$P(t) + S(t) = C(t) + Ke^{-r(T-t)}$$

together with equation (3) we can derive the value of the European Put Option:

$$P(S,t) = Ke^{-r(T-t)}N(-d_2) - SN(-d_1) \tag{5}$$

3 Finite Differences

The method of Finite Difference approximation is based upon the discretization of time and space parameters of a PDE to produce difference equations. The difference equations relate the values of the approximation to values at different time and space coordinates. Once a set of difference equations is derived for a particular PDE these equations are then solved iteratively with respect to time and space for each point on a lattice. Initial conditions are required to begin the iterations while boundary conditions are required at each time step.

Finite Difference schemes for PDE differ mainly by the particular discretizations chosen that make up these schemes. Each scheme has inherent advantages and disadvantages in terms of order of accuracy, consistency, rate of convergence, dissipation, stability, and error. The three most common schemes used for solving Black Scholes Equation are: the Explicit, the Implicit, and Crank Nicolson.

Explicit finite difference schemes are the simplest types of finite difference schemes that can be obtained for solving a PDE. These are schemes for which the difference equations specify the value of the approximation at some future time as a function of some set of neighbouring values at previous times. In general, consider schemes with difference equations of the following form:

$$v_m^{n+1} = \sum_{k=-\alpha}^{\beta} c_{m+k} v_{m+k}^{n} \tag{6}$$

where c_{m+k} can be interpreted as the weight or contribution of the value at $m+k$. Explicit schemes are the easiest schemes to implement however they are not the most accurate due to their simplicity.

An implicit scheme differs from an explicit scheme in that the values of the approximation at future times also depends on neighbouring values at those future times. For example, consider schemes with the following form:

$$av_{m+1}^{n+1} + bv_m^{n+1} + cv_{m-1}^{n+1} = \sum_{k=-\beta}^{\alpha} c_{m+k} v_{m+k}^{n} \tag{7}$$

These schemes are often more accurate and exhibit better stability and convergence characteristics then their Explicit counterparts.

The Crank Nicolson scheme is an excellent scheme for solving the Black Scholes Equation. For diffusion equations (like many forms of Black Scholes) this implicit scheme is characterized by unconditional numeric stability, a rapid rate of convergence, second order accuracy, and relative simplicity.

3.1 Derivation of Difference Equations for Black Scholes

The Crank Nicolson Finite Difference scheme for Black Scholes Equation is based upon discretization of time and price of the underlying to produce difference equations for the PDE. The difference equations are then solved iteratively with respect to time and space for each point on a lattice for all of time up to maturity of the option. Initial, or in the case of Black Scholes, final conditions are required to begin the iterations along with conditions to be satisfied at the boundaries.

To obtain the Crank Nicolson finite difference approximation to equation

(1) we discretize the parameters of time and space (t and S) such that

$$\begin{aligned} t &= n\Delta t & n &= 1, 2, \ldots, N \\ S &= m\Delta S & m &= 1, 2, \ldots, M \end{aligned}$$

We use a second order difference for the time partial derivative. For each of the S partial derivatives we take the average of two differences, one in $t = (n+1)\Delta t$ and the other in $t = n\Delta t$. We use central differences for the first partial and second order differences for the second partial derivatives with respect to S. These differences, obtained by taking Taylor series approximations to $V(S, t)$ around $S + \Delta S$ and $t + \Delta t$, are as follows:

$$\begin{aligned} \frac{\partial v}{\partial t} &\approx \frac{v_m^{n+1} - v_m^n}{\Delta t} + \mathcal{O}(\Delta t^2) \\ \frac{\partial v}{\partial S} &\approx \frac{1}{2}\left(\frac{v_{m+1}^{n+1} - v_{m-1}^{n+1}}{2\Delta S} + \frac{v_{m+1}^{n} - v_{m-1}^{n}}{2\Delta S}\right) + \mathcal{O}(\Delta t^2) + \mathcal{O}(\Delta S^2) \\ \frac{\partial^2 v}{\partial S^2} &\approx \frac{1}{2}\left(\frac{v_{m+1}^{n+1} - 2v_m^{n+1} + v_{m-1}^{n+1}}{\Delta S^2} + \frac{v_{m+1}^{n} - 2v_m^{n} + v_{m-1}^{n}}{\Delta S^2}\right) \\ &\quad + \mathcal{O}(\Delta t^2) + \mathcal{O}(\Delta S^2) \end{aligned}$$

After converting equation (1) into an initial value problem using the time reversal substitution $\hat{n} = N - n$ and collecting like terms we arrive at the following set of difference equations for Black Scholes' PDE:

$$\begin{aligned} &\left(-\frac{\sigma^2 m^2}{2} - \frac{rm}{2}\right) v_{m+1}^{\hat{n}+1} + \left(\frac{2}{\Delta t} + \sigma^2 m^2 + r\right) v_m^{\hat{n}+1} + \left(-\frac{\sigma^2 m^2}{2} + \frac{rm}{2}\right) v_{m-1}^{\hat{n}+1} \\ &= \left(\frac{\sigma^2 m^2}{2} + \frac{rm}{2}\right) v_{m+1}^{\hat{n}} + \left(\frac{2}{\Delta t} - \sigma^2 m^2 - r\right) v_m^{\hat{n}} + \left(\frac{\sigma^2 m^2}{2} - \frac{rm}{2}\right) v_{m-1}^{\hat{n}} \end{aligned}$$

where

$$\hat{n} = 1, 2, \ldots, N \qquad \text{and} \qquad m = 1, 2, \ldots, M$$

Noting the structure of the above equation we see that it is a tridiagonal matrix. We can write this more compactly as

$$\alpha v_{m+1}^{\hat{n}+1} + \beta v_m^{\hat{n}+1} + \gamma v_{m-1}^{\hat{n}+1} = a v_{m+1}^{\hat{n}} + b v_m^{\hat{n}} + c v_{m-1}^{\hat{n}} \tag{8}$$

where

$$\alpha = -\frac{\sigma^2 m^2}{2} - \frac{rm}{2} \qquad \beta = \frac{2}{\Delta t} + \sigma^2 m^2 + r \qquad \gamma = -\frac{\sigma^2 m^2}{2} + \frac{rm}{2} \tag{9a}$$

and

$$a = \frac{\sigma^2 m^2}{2} + \frac{rm}{2} \qquad b = \frac{2}{\Delta t} - \sigma^2 m^2 - r \qquad c = \frac{\sigma^2 m^2}{2} - \frac{rm}{2} \tag{9b}$$

3.2 Discrete Conditions

Applying the time reversal substitution $\hat{n} = N - n$ to the final condition (2a) gives the initial condition

$$v_m^0 = \max(m\Delta S - K) \qquad m = 1, 2, \ldots, M \tag{10a}$$

Boundary conditions are derived by differencing equations (2b) and (2c) to second order using the finite difference approximation given above. For the left boundary condition at $m = M$ we use the second order difference approximation to approximate the partial derivative with respect to S. This results in the following

$$v_0^{\hat{n}} = 0 \tag{10b}$$

and

$$v_{M+1}^{\hat{n}+1} - v_{M-1}^{\hat{n}+1} + v_{M+1}^{\hat{n}} - v_{M-1}^{\hat{n}} = 1 \tag{10c}$$

4 Implementation

4.1 Finite Difference Calculations

The solution to the difference equations (8), (9a), and (9b) with initial and boundary conditions (10a), (10b), and (10c) was obtained using the Thomas Algorithm to solve a tridiagonal matrix at each time step. See *Section 7* for the complete code listing.

4.2 Error Calculations

A numerical approximation was used for calculating the cumulative normal distributions for the exact solution (See *Section 7*). This method was only verified qualitatively but needs to be more closely considered as it introduces errors in the calculation of the exact solution.

Errors were calculated using the following method:

$$\text{Error(n)}^2 = \Delta S \sum_{m=1}^{M} |C(n\Delta t, m\Delta S) - v_m^n|^2 \qquad n = 1, 2, \ldots, N$$

where $C(n\Delta t, m\Delta S)$ is the exact solution given in equation (3).

5 Results

The surface resulting from the finite difference calculations for the European Call Option is shown in *Figure 1* along with the parameters used.

Figure 2 shows the error in the numerical solution as a function of time. Since the error decreases with time step the figure either indicates convergence to the solution, an error in the finite difference approximation, or an error in the calculation of the exact solution.

6 Future Refinements

Further consideration is required to verify the accuracy of the solutions. Once accuracy has been verified the algorithms can be modified to produce numerical solutions for the European Put Option. American Options would be next on the list followed by Chinese and Bermudian Options. Fortunately, these methods can be used for all of these with slight modifications to the Difference Equations and Boundary Conditions.

7 Code Listing

```
void thomas (const int n)
{
  int m;
  double dd, denom, numer, val;
  double alphas[3], betas[3];
  int left = configs[CF_LEFT].ival;
  int right = configs[CF_RIGHT].ival;

  pdata[1] = 0;

  /* left boundary */
  qdata[1] = 0;

  for (m = 1; m < Xm-1; m++) {
    blackParams(n, m, alphas, betas);

    dd =  betas[0] * data[n-1][m+1];
    dd += betas[1] * data[n-1][m];
    dd += betas[2] * data[n-1][m-1];
```

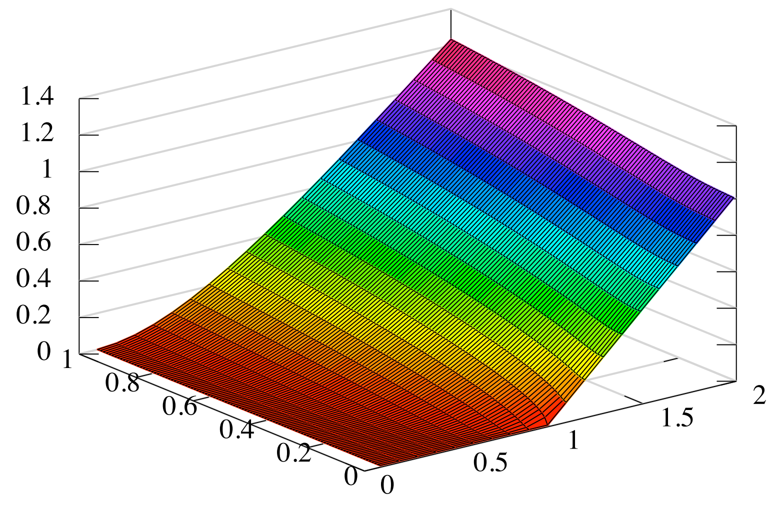

Figure 1: Numerical solution to Black Scholes for European Call Option with $\sigma = 0.5, r = 0.1, K = 1, T = 2$

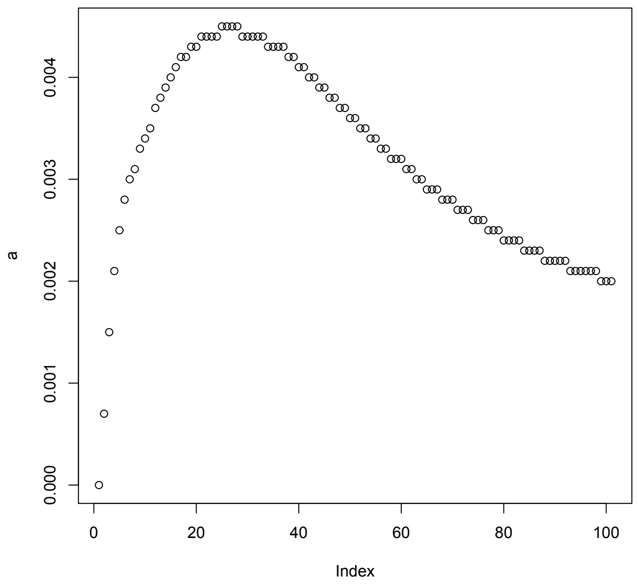

Figure 2: Error vs Time Step for numerical solution to Black Scholes for European Call Option with $\sigma = 0.5, r = 0.1, K = 1, T = 2$

```
    denom = (alphas[0] * pdata[m] + alphas[1]);

    pdata[m+1] = -alphas[2] / denom;
    qdata[m+1] = (dd - alphas[0] * qdata[m]) / denom;
  }

  /* right boundary */
  boundary(n);

  for (m = Xm-2; m >= 0; m--) {
    val = pdata[m+1]*data[n][m+1] + qdata[m+1];
    calcError(n, m, val);
    data[n][m] = val;
  }
}

void blackParams (int t, int mm,
  double alphas[3], double betas[3])
{
  double sigma2 = configs[CF_SIGMA].fval
    *configs[CF_SIGMA].fval;
  double r = configs[CF_R].fval;
  double m = mm;
  double m2 = m*m;

  alphas[0] = -(sigma2*m2/2.0 + r*m/2.0);
  alphas[1] = 2.0/dt + sigma2*m2 - r;
  alphas[2] = (-sigma2*m2/2.0 + r*m/2.0);

  betas[0] = -alphas[0];
  betas[1] = 2.0/dt - sigma2*m2 + r;
  betas[2] = -alphas[2];
}

void boundary (int n)
{
  double numer, denom;
  double b = bFunc(0, 0);    /* since b is constant */
  double lambda = configs[CF_LAMBDA].fval;
  double mu = configs[CF_MU].fval;
  int right = configs[CF_RIGHT].ival;

  switch (configs[CF_BOUNDARY].ival) {
    case BOUND_ONE:
      data[n][Xm-1] = data[n-1][Xm-2];
      break;

    case BOUND_TWO:
      denom = 1 + lambda * (1 - pdata[Xm-1]) - dt;
      numer = data[n-1][Xm-1] + lambda * qdata[Xm-1];
      data[n][Xm-1] = numer/denom;
      break;

    case BOUND_THREE:
      denom = 1 - pdata[Xm-1]*(2 - pdata[Xm-2]);
      numer = qdata[Xm-1]*(2 - pdata[Xm-2])
        - qdata[Xm-2];
      data[n][Xm-1] = numer/denom;
      break;

    /* Neumann Boundary */
    case BOUND_FOUR:
      denom = 1 + b*mu/2*(1 - pdata[Xm-1]);
      numer = data[n-1][Xm-1] * (1 - b*mu/2)
        + b*mu/2* ( data[n-1][Xm-2] + qdata[Xm-1] );
      data[n][Xm-1] = numer/denom;
      break;

    /* Exact Boundary */
    default:
      data[n][Xm-1] = solution(n*dt, right);
      break;
  }
}

void fdm (void)
{
  int m, n, i;
  double v0m, x;
  int x0 = configs[CF_LEFT].ival;
  int y0 = configs[CF_BOTTOM].ival;
  double k = configs[CF_K].fval;

  /* reset maxima and error */
  for (i = 0; i < Tn; i++) {
    maxv[i] = TINY;
    error[i] = 0;
  }
  maxmaxv = TINY;

  blowup = 0;

  /* apply initial condition */
  for (m = 0; m < Xm; m++)
  {
    x = x0 + m*dx;
    v0m = ((x0 + m*dx) - k);
    data[0][m] = (v0m >= 0) ? v0m : 0;
    calcError(0, m, (v0m >= 0) ? v0m : 0);
  }

  /* solve pde */

  for (n = 1; n < Tn; n++)
    thomas(n);

  reshapeWorld();
}

void calcError(const int n, const int m, const double val)
{
  double t = n * dt;
  double x = configs[CF_LEFT].ival + m * dx;

  if (fabs(val) > BLOWUP_VAL && !blowup)
    blowup = n;

  if (val > maxv[n])
    maxv[n] = val;
  if (val > maxmaxv)
    maxmaxv = val;

  error[n] += pow(solution(t, x) - val, 2);
}

double N01_b[5] = {0.31938153, -0.356563782,
  1.781477937, -1.821255978, 1.330274429};

/* approximate cumulative normal distribution */
double N01(double z)
{
  if (z > 6.0)
    return 1.0;
  else if (z < -6.0)
    return 0.0;

  double a = fabs(z);
  double t = 1.0/(1.0 + a * 0.2316419);
  double d = 0.3989423 * exp(-z*z/2.0);
  double n = N01_b[4];

  for (int i = 3; i >= 0; i--)
    n = (n + N01_b[i])*t;
  n = 1.0 - d*n;
  if (z < 0.0)
    n = 1.0 - n;

  return n;
}

double solution (double t, double s)
{
  double res;
  double K = configs[CF_K].fval;
  double sigma = configs[CF_SIGMA].fval;
  double r = configs[CF_R].fval;
  double T = Tn*dt; // configs[CF_MU].fval;
  double Tsqrt;
  double d1;
  double d2;

  t = T - t;

  Tsqrt = sqrt(T - t);

  d1 = (log(s/K) + (r + sigma*sigma/2)*Tsqrt)/(sigma*Tsqrt);
  d2 = d1 - sigma*Tsqrt;

  switch (configs[CF_DERIV].ival) {
    case DERIV_VANILLA_CALL:
      res = s * N01(d1) - K*exp(-r*(T-t)) * N01(d2);
      break;

    case DERIV_VANILLA_PUT:
      res = K*exp(-r*(T-t)) * N01(-d2) - s * N01(-d1);
      break;
  }

  return res;
}
```