Implicit Finite Difference Method: Black-Scholes Option Pricing

Tatsuro Senga

Outline

The Black-Scholes Equation

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + (r - d)S\frac{\partial V}{\partial S} - rV = 0$$

where:

- $V(S, t)$ is the option price
- S is the stock price
- t is time
- σ is volatility
- r is risk-free rate
- d is dividend yield

Boundary Conditions

For a call option:

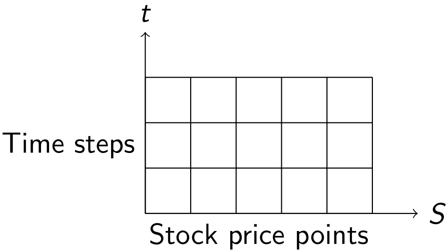
$$\begin{aligned} V(S,T) &= \max(S-K,0) \quad \text{(Terminal condition)} \\ V(0,t) &= 0 \\ V(S_{\text{max}},t) &= S_{\text{max}} - K \end{aligned}$$

For a put option:

$$\begin{aligned} V(S,T) &= \max(K-S,0) \quad \text{(Terminal condition)} \\ V(0,t) &= K \\ V(S_{\text{max}},t) &= 0 \end{aligned}$$

Grid Setup

- Time discretization: $t_i = i\Delta t$, $i = 0, \dots, N$
- Stock price discretization: $S_j = S_{\min} + j\Delta S$, $j = 0, \dots, M$
- $\Delta t = T/N$
- $\Delta S = (S_{\max} - S_{\min})/M$



Finite Difference Approximations

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + (r-d)S\frac{\partial V}{\partial S} - rV = 0$$

We approximate this using:

$$\begin{aligned}
\frac{\partial V}{\partial t} &\approx \frac{V_j^{i+1} - V_j^i}{\Delta t} \\
\frac{\partial V}{\partial S} &\approx \frac{V_{j+1}^i - V_{j-1}^i}{2\Delta S} \\
\frac{\partial^2 V}{\partial S^2} &\approx \frac{V_{j+1}^i - 2V_j^i + V_{j-1}^i}{(\Delta S)^2}
\end{aligned}$$

where V_j^i represents the option value at (S_j, t_i)

Implicit Scheme Derivation (1)

Substituting finite differences into Black-Scholes PDE:

$$\frac{V_j^{i+1} - V_j^i}{\Delta t} + \frac{1}{2}\sigma^2 S_j^2 \frac{V_{j+1}^i - 2V_j^i + V_{j-1}^i}{(\Delta S)^2}$$
$$+ (r - d)S_j \frac{V_{j+1}^i - V_{j-1}^i}{2\Delta S} - rV_j^i = 0$$

where $S_j = S_{\min} + j\Delta S$

Implicit Scheme Derivation (2)

Rearranging terms:

$$V_j^i - V_j^{i+1} + \frac{1}{2}\sigma^2 j^2 \Delta t (V_{j+1}^i - 2V_j^i + V_{j-1}^i)$$
$$+ (r-d)j\frac{\Delta t}{2}(V_{j+1}^i - V_{j-1}^i) - r\Delta t V_j^i = 0$$

Define coefficients:

$$a_j = \frac{1}{2}(r-d)j\Delta t - \frac{1}{2}\sigma^2 j^2 \Delta t$$
$$b_j = 1 + \sigma^2 j^2 \Delta t + r\Delta t$$
$$c_j = -\frac{1}{2}(r-d)j\Delta t - \frac{1}{2}\sigma^2 j^2 \Delta t$$

Implicit Scheme Derivation (3)

Final form:

$$a_j V_{j-1}^i + b_j V_j^i + c_j V_{j+1}^i = V_j^{i+1}$$

For all interior points ($j = 1, \ldots, M-1$):

- Tridiagonal system at each time step
- Solved backwards in time ($i = N-1, \ldots, 0$)
- Boundary conditions used at $j = 0$ and $j = M$

Implicit Scheme

- After substituting finite differences:

$$a_j V_i^{j-1} + b_j V_i^j + c_j V_i^{j+1} = V_{i+1}^j$$

where:

$$a_j = \frac{1}{2}(r-d)j\Delta t - \frac{1}{2}\sigma^2 j^2 \Delta t$$
$$b_j = 1 + \sigma^2 j^2 \Delta t + r\Delta t$$
$$c_j = -\frac{1}{2}(r-d)j\Delta t - \frac{1}{2}\sigma^2 j^2 \Delta t$$

Matrix Form

- The system can be written as:

$$A\vec{V}_i = \vec{V}_{i+1}$$

- Where A is tridiagonal:

$$A = \begin{pmatrix} b_1 & c_1 & 0 & \cdots & 0 \\ a_2 & b_2 & c_2 & \cdots & 0 \\ 0 & a_3 & b_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & b_M \end{pmatrix}$$

Solution Algorithm

Algorithm 1 Implicit Finite Difference Method

```
Initialize grid and boundary conditions
Set terminal conditions V_N^j for all j for i = N − 1 to 0 do

        end
        Form tridiagonal matrix A
Solve A V⃗_i = V⃗_{i+1}
Apply free boundary condition

```

Set terminal conditions V_N^j for all j **for** $i = N - 1$ *to* 0 **do**; Solve $A\vec{V}_i = \vec{V}_{i+1}$

Tridiagonal Coefficients

The coefficients are:

$$
\begin{aligned}
a_j &= \frac{\Delta t}{2}[(r-d)j - \sigma^2 j^2] \\
b_j &= 1 + \sigma^2 j^2 \Delta t + r\Delta t \\
c_j &= -\frac{\Delta t}{2}[(r-d)j + \sigma^2 j^2]
\end{aligned}
$$

In MATLAB code:

```
a = @(j) 0.5*(r-d)*j*dt - 0.5*volatility^2*j^2*dt;
b = @(j) 1 + volatility^2*j^2*dt + r*dt;
c = @(j) -0.5*(r-d)*j*dt - 0.5*volatility^2*j^2*dt;
```

Matrix System

At each time step, we solve:

$$\begin{pmatrix} b_1 & c_1 & 0 & \cdots & 0 \\ a_2 & b_2 & c_2 & \cdots & 0 \\ 0 & a_3 & b_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & b_M \end{pmatrix} \begin{pmatrix} V_1^i \\ V_2^i \\ V_3^i \\ \vdots \\ V_M^i \end{pmatrix} = \begin{pmatrix} V_1^{i+1} - a_1 V_0^i \\ V_2^{i+1} \\ V_3^{i+1} \\ \vdots \\ V_M^{i+1} - c_M V_{M+1}^i \end{pmatrix}$$

MATLAB Implementation

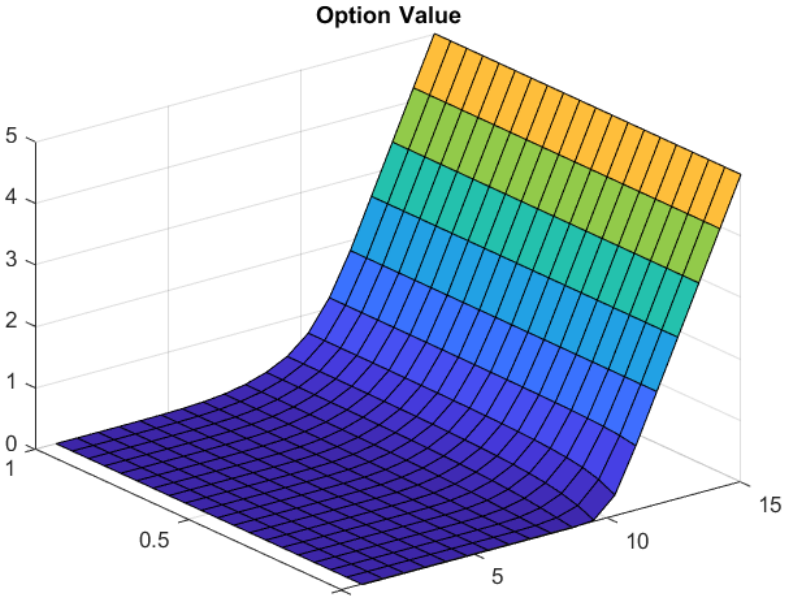
Key steps in the code:

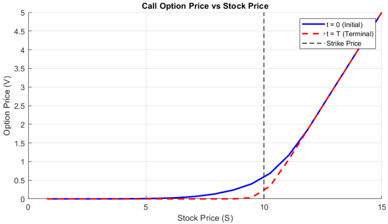
```
% Create tridiagonal matrix
A = diag(a(2:M-1),-1) + diag(b(2:M))
    + diag(c(1:M-2),1);

% Set up right-hand side
v = surf(i+1,2:M)';
v(1) = v(1) - a(1)*surf(i,1);
v(end) = v(end) - c(M+1)*surf(i,M+1);

% Solve system
surf(i,2:M) = A\v;
```

Call Option Price





Option Price Components:

- Blue line: Total option value
  - = Intrinsic value + Time value
- Red dashed: Intrinsic value
  - $= \max(S - K, 0)$
- Time value = Blue − Red

Time Value Analysis:

- Out-of-money ($S < K$):
  - Pure time value
  - Decays with distance from K
- At-the-money ($S = K = 50$):
  - Maximum time value
  - $0.4K\sigma\sqrt{T}$ (rule of thumb)
- In-the-money ($S > K$):
  - Time value decreases
  - Approaches intrinsic value

Parameters: $T = 1$ year, $\sigma = 40\%$, $r = 2\%$, $K = 50$

Grid Selection

Important factors:

- Choice of $S_{\max}$: typically 3-4 times strike price
- Number of time steps (N)
- Number of stock price steps (M)
- Trade-off between accuracy and computational cost

Recommendation:

- Start with $N = M = 100$
- Increase if more accuracy needed
- Check convergence by doubling grid points

Function Definition and Grid Setup

```
function [t_vals,S_vals,surf] = black_scholes_naive_implicit(...
    N,M,Smin,Smax,T,K,volatility,r,d,is_call)
% Initialize the solution surface
surf = zeros(1+N,1+M);  % Size: (time steps + 1)    (price steps + 1)
```

Parameters explanation:

- `N`: Number of time steps
- `M`: Number of stock price steps
- `Smin, Smax`: Stock price range
- `T`: Time to maturity
- `K`: Strike price
- `volatility, r, d`: Market parameters
- `is_call`: Option type flag

Grid Construction

```
% Determine step sizes
dt = T/N;        % Time step size
dS = (Smax-Smin)/M;  % Stock price step size

% Create grid points
t_vals = 0:dt:T;      % Time points array
S_vals = Smin:dS:Smax;  % Stock price array
```

Grid Details:

- Time grid: $[0, T]$ divided into N steps
- Stock price grid: $[S_{min}, S_{max}]$ divided into M steps
- `t_vals`: Vector of length $N + 1$
- `S_vals`: Vector of length $M + 1$

Boundary Conditions

```
% Set boundary conditions
if is_call
    surf(:,1) = 0;                  % At S = Smin
    surf(:,end) = Smax-K;           % At S = Smax
    surf(end,:) = payoff(S_vals,K,is_call);   % At T
else
    surf(:,1) = K;                  % At S = Smin
    surf(:,end) = 0;                % At S = Smax
    surf(end,:) = payoff(S_vals,K,is_call);   % At T
end
```

Explanation:

- `surf(:,1)`: Left boundary ($S = S_{min}$)
- `surf(:,end)`: Right boundary ($S = S_{max}$)
- `surf(end,:)`: Terminal condition at T

Tridiagonal System Setup

```
% Define tridiagonal matrix coefficients
a = @(j) 0.5*(r-d)*j*dt - 0.5*volatility^2*j^2*dt;  % Lower
b = @(j) 1 + volatility^2*j^2*dt + r*dt;            % Main
c = @(j) -0.5*(r-d)*j*dt - 0.5*volatility^2*j^2*dt; % Upper
```

These coefficients come from discretizing the PDE:

- `a(j)`: Coefficient of V_{j-1}^{i} (lower diagonal)
- `b(j)`: Coefficient of V_{j}^{i} (main diagonal)
- `c(j)`: Coefficient of V_{j+1}^{i} (upper diagonal)

Time-Stepping Loop

```
for i = N:-1:1  % Backward in time
    % Construct tridiagonal matrix
    A = diag(a(2:M-1),-1) + diag(b(2:M)) + ...
        diag(c(1:M-2),1);

    % Set up right-hand side vector
    v = surf(i+1,2:M)';
    v(1) = v(1) - a(1)*surf(i,1);           % Left boundary
    v(end) = v(end) - c(M+1)*surf(i,M+1);  % Right boundary

    % Solve linear system
    surf(i,2:M) = A\v;
```

At each time step:

- Build tridiagonal matrix `A`
- Construct RHS vector `v` with boundary adjustments
- Solve system $AV^i = v$ for interior points

Matrix Structure Visualization

Example for $M = 5$:

$$\begin{pmatrix} b_2 & c_2 & 0 & 0 \\ a_3 & b_3 & c_3 & 0 \\ 0 & a_4 & b_4 & c_4 \\ 0 & 0 & a_5 & b_5 \end{pmatrix} \begin{pmatrix} V_2^i \\ V_3^i \\ V_4^i \\ V_5^i \end{pmatrix} = \begin{pmatrix} V_2^{i+1} - a_2 V_1^i \\ V_3^{i+1} \\ V_4^{i+1} \\ V_5^{i+1} - c_5 V_6^i \end{pmatrix}$$

- Matrix is tridiagonal
- System solved efficiently using backslash operator
- First and last rows modified by boundary conditions