

Background

The Black-Scholes Model is a partial differential equation (PDE) developed by Fisher Black and Myron Scholes to evaluate the underlying price of European options. An option is an agreement where someone can reserve to buy (call) or sell (put) a stock at specific time. Unlike American options, European options can only be exercised at the maturity date.

I chose to work on this PDE because I had experience with the equation while working as a pricing analyst in the natural gas industry, but never really understood the equation. The Risk Management team that worked alongside the Pricing team would mitigate risk (hedge) by buying call options on future consumption of natural gas by customers, whose contracts we would price.

The research paper ***Examination of Impact from Different Boundary Conditions on the 2D Black-Scholes Model: Evaluating Pricing of European Call Options*** by Tomas Sundvall and David Trång[ST14] informed this project.

Black-Scholes Model

The Black-Scholes model is defined by the partial differential equation...

$$F_t = -rxF_x - ryF_y - \frac{1}{2}x^2\sigma^2(1,1)F_{xx} - \frac{1}{2}y^2\sigma^2(2,2)F_{yy} - xy\sigma^2(1,2)F_{xy} + rF$$

With x and y representing the hypothetical price of two assets with some volatility correlation to each other.

The payoff function at the maturity time is:

$$F(T, x, y) = \Phi(x, y) = \left(\frac{x + y}{2} - K \right)^+$$

The parameters in the equation are:

- r : Risk-free Investment (often US bonds)
- σ : Volatility Correlation Matrix

$$\sigma = \begin{pmatrix} \sigma_{xx} & \sigma_{yx} \\ \sigma_{xy} & \sigma_{yy} \end{pmatrix}, \quad \sigma_{yx} = \sigma_{xy}, \quad \sigma_{xx} = \sigma_{yy}$$

- T : Final Maturity Time
- K : Strike Price (Call Option Premium)

Parameters

As used in the paper by Sundvall and Trångor the project, I will be using the parameters:

- $r = 0.1$
- $\sigma(1,1) = 0.3$
- $\sigma(1,2) = 0.05$

Being undefined in the paper, I will also be using the parameters:

- $T = 1$
- $K = 1$

The following code displays the implementation of the parameters

```

1 %% Parameters
2
3 strike = 1;           % Strike Price
4 T = 1;               % Simulation time or Final Maturity Time
5
6 a = 0;               % Minimum Value of Option for Asset X (must be
   zero)
7 b = round(10*strike); % Maximum Value of Option for Asset X per
   recommendation of reference paper (between 8*K and 12*K)
8 c = 0;               % Minimum Value of Option for Asset Y (must be
   zero)
9 d = b;               % Maximum Value of Option for Asset X
10
11 m = 8* round(10*strike); % Personal Preference: Gives Enough Divisions for
   a More Accurate Result
12 n = m;               % Number of cells along the y-axis
13
14 dx = (b-a)/m;        % Step length along the x-axis
15 dy = (d-c)/n;        % Step length along the y-axis
16
17
18 dt = 0.001;          % Personal Preference: Much less than Von Neumann
   stability criterion for explicit scheme dx^2/(4) (about 0.0039)
19
20 omega11 = 0.3;        % Omega_xx = Omega_yy of the volatility
   correlation matrix
21 omega12 = 0.05;       % Omega_xy = Omega_yx of the volatility
   correlation matrix
22 r = 0.1;              % Risk free interest rate

```

Initial Values

Initial Values

Since the value of the option is discounted to the present, the initial condition is the payoff function:

$$F(T, x, y) = \Phi(x, y) = \left(\frac{x + y}{2} - K \right)^+$$

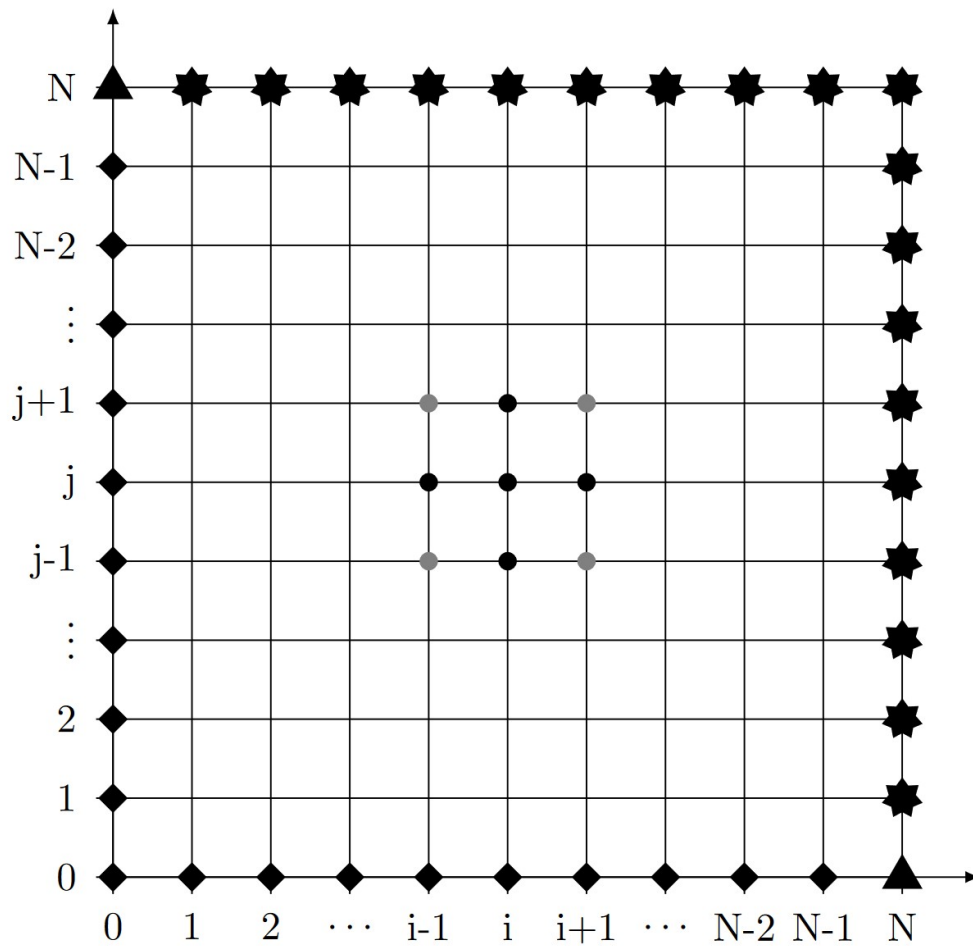
This initial value is displayed in the code below:

```
1 %% Initial Values for time = T
2
3 ICV = max(((X+Y)/2)-strike , 0);
```

Initial Values

Dirichlet Boundary Conditions

By the nature of options having a minimum value of zero (lower domain) but no maximum price, the boundary conditions are divided into close-field and far-field boundary conditions. Shown below, are the close-field (as diamonds) and far-field (as stars) boundary conditions. The triangular points where the close-field boundary conditions and meet the far-field boundary conditions can be either of the two, however, in this project, they are defined as far-field boundary conditions. The far-field boundary condition will be updated with each time step with the close-field boundary conditions being incorporated into the inverted matrix, A .



Close-Field Boundary Conditions

Due to $y = 0$ on the x-axis and $x = 0$ on the y-axis, the x and y axes simplify as:

$$F_t = -rxF_x - \frac{1}{2}x^2\sigma^2(1,1)F_{xx} + rF$$

$$F_t = -ryF_y - \frac{1}{2}y^2\sigma^2(2,2)F_{yy} + rF$$

The value at the origin $F(t, 0, 0) = 0$.

Far-Field Boundary Conditions

Extrapolating the 1D case of the limit as the payoff function goes to infinity into 2D, the paper defined the far-field boundary conditions as:

$$F(t, x_{max}, y) = \frac{x_{max} + y}{2} - Ke^{-r(T-t)}$$

$$F(t, x, y_{max}) = \frac{x + y_{max}}{2} - Ke^{-r(T-t)}$$

The paper notes, as a "rule of thumb", to limit the upper domain by setting the x_{max} and y_{max} to 4K to 6K times the number of spatial dimensions (two in this case). In this case, I have decided to use 10K as the upper limit for both x and y.

Discretization in Space

Following the same method as the referred paper, this section will cover the derivative matrices using central differences of second order accuracy:

$$\left(\frac{\partial F}{\partial x}\right)_{i,j} \approx \frac{U_{i+1,j} - U_{i-1,j}}{2\Delta x}, \quad \left(\frac{\partial^2 F}{\partial x^2}\right)_{i,j} \approx \frac{U_{i+1,j} + 2U_{i,j} + U_{i-1,j}}{\Delta x^2},$$

$$\left(\frac{\partial F}{\partial y}\right)_{i,j} \approx \frac{U_{i,j+1} - U_{i,j-1}}{2\Delta y}, \quad \left(\frac{\partial^2 F}{\partial y^2}\right)_{i,j} \approx \frac{U_{i,j+1} + 2U_{i,j} + U_{i,j-1}}{\Delta y^2},$$

$$\left(\frac{\partial^2 F}{\partial x \partial y}\right)_{i,j} \approx \frac{U_{i+1,j+1} - U_{i+1,j-1} - U_{i-1,j+1} + U_{i-1,j-1}}{4\Delta x \Delta y}$$

The Black-Scholes equation can be reconstructed with these approximations as...

$$\begin{aligned} \left(\frac{\partial F}{\partial t}\right)_{i,j} \approx & -rx \frac{U_{i+1,j} - U_{i-1,j}}{2\Delta x} - ry \frac{U_{i+1,j} + 2U_{i,j} + U_{i-1,j}}{\Delta x^2} \\ & - \frac{\sigma^2(1,1)x^2}{2} \frac{U_{i+1,j} + 2U_{i,j} + U_{i-1,j}}{\Delta x^2} - \frac{\sigma^2(2,2)y^2}{2} \frac{U_{i,j+1} + 2U_{i,j} + U_{i,j-1}}{\Delta y^2} \\ & - \sigma^2(1,2)xy \frac{U_{i+1,j+1} - U_{i+1,j-1} - U_{i-1,j+1} + U_{i-1,j-1}}{4\Delta x \Delta y} + rU_{i,j} \end{aligned}$$

One point to notes is that since the goal is to create an implicit euler scheme, in order to make the invertable matrix, A , I will be making $\left(\frac{\partial}{\partial t}\right)_{i,j}$ and later applying F .

Implementing $A = \left(\frac{\partial}{\partial t}\right)_{i,j}$ Not Including Boundary Conditions

The MATLAB code below displays the implementation of $A = \left(\frac{\partial}{\partial t}\right)_{i,j}$. All rows corresponding to the near and far boundary conditions are left blank in this process.

```

1 %% Setting Up Matrix for Fx
2
3 Fx = Fx_Matrix(m,n,dx,dy); % 2nd Order 2D Scheme for
   First Derivative with Respect to X
4 Fy = Fy_Matrix(m,n,dx,dy); % 2nd Order 2D Scheme for
   First Derivative with Respect to Y
5
6 Fxx = Fxx_Matrix(m,n,dx,dy); % 2nd Order 2D Scheme for
   Second Derivative with Respect to X
7 Fyy = Fyy_Matrix(m,n,dx,dy); % 2nd Order 2D Scheme for
   Second Derivative with Respect to Y
8
9 Fxy = Fxy_Matrix(m,n,dx,dy); % 2nd Order 2D Scheme for
   Mixed Derivative with Respect to X and Y
10
11 sub_matrix = diag(diag(comp_matrix("x", m, n))); % Matrix to Subtract from
   speye so All Boundary Conditions in A are Zero
12
13 %% Using Black-Scholes PDE to Create A (Excluding Boundary Conditions)

```

```

14
15 A = (-r*Xmatrix*Fx) - (r*Ymatrix*Fy) - ((1/2)*omega11^2*Xmatrix*Xmatrix * Fxx)
    - ((1/2)*omega11^2*Ymatrix*Ymatrix * Fyy) - (omega12^2*Xmatrix*Ymatrix*Fxy)
    + (r*(speye((m+1)*(n+1))-sub_matrix));

```

Implementation of $\left(\frac{\partial}{\partial t}\right)_{i,j}$

```

1  function matrixdx = Fx_Matrix(m,n,dx,dy)
2      one = ones(m+1,1);
3      sparse_m = sparse(m+1,1);
4      A = spdiags([-1*one sparse_m one],-1:1,m+1,m+1);
5      A(1,:) = sparse_m';
6      A(end,:) = sparse_m';
7
8      sparse_y = speye(n+1,n+1);
9      sparse_n = sparse(n+1,1);
10     sparse_y(:,1) = sparse_n;
11     sparse_y(:,end) = sparse_n;
12
13     matrixdx = kron(sparse_y, A)/(2*dx);
14 end
15
16 function matrixdy = Fy_Matrix(m,n,dx,dy)
17     one = ones(n-1,1);
18     sparse_n = sparse(n+1,1);
19     one = sparse([0;one;0]);
20     one_list = repmat(one,m-1,1);
21     one_list1 = [sparse_n; one_list; sparse_n];
22     one_list2 = [sparse_n; one_list; sparse_n];
23
24     A = spdiags([-1*one_list1 repmat(sparse_n, n+1, 2*(m+1)-1) one_list2],-(m
+1):(m+1),(m+1)*(n+1),(m+1)*(n+1));
25
26     matrixdy = -1*((A)/(2*dy))';
27 end
28
29 function matrixdxx = Fxx_Matrix(m,n,dx,dy)
30     one = ones(m+1,1);
31     sparse_m = sparse(m+1,1);
32     A = spdiags([one -2*one one],-1:1,m+1,m+1);
33     A(1,:) = sparse_m';
34     A(end,:) = sparse_m';
35
36     sparse_y = speye(n+1,n+1);
37     sparse_n = sparse(n+1,1);
38     sparse_y(:,1) = sparse_n;
39     sparse_y(:,end) = sparse_n;
40
41     matrixdxx = kron(sparse_y, A)/(dx^2);
42 end
43
44 function matrixdyy = Fyy_Matrix(m,n,dx,dy)
45     one = ones(n-1,1);
46     sparse_n = sparse(n+1,1);
47     one = sparse([0;one;0]);
48     one_list = repmat(one,m-1,1);

```

```

49     one_list1 = [sparse_n; one_list; sparse_n];
50     one_list2 = [sparse_n; one_list; sparse_n];
51     diag = [sparse_n; one_list; sparse_n];
52
53     A = spdiags([one_list1 repmat(sparse_n, n+1, m) -2*diag repmat(sparse_n, n
+1, m) one_list2], -(m+1):(m+1), (m+1)*(n+1), (m+1)*(n+1));
54
55     matrixddy = ((A)/(dy^2))';
56 end
57
58 function matrixdxy = Fxy_Matrix(m,n,dx,dy)
59     one = ones(n-1,1);
60     sparse_n = sparse(n+1,1);
61     one1 = sparse([0;one;0]);
62     one2 = sparse([0;one;0]);
63     one_list = repmat(one1,m-1,1);
64     one_list2 = repmat(one2,m-1,1);
65     one_list1 = [sparse_n; one_list; sparse_n];
66     one_list2 = [sparse_n; one_list2; sparse_n];
67     one_list3 = [sparse_n; one_list2; sparse_n];
68     one_list4 = [sparse_n; one_list; sparse_n];
69
70     sparse_list = repmat(sparse_n,m+1,1);
71
72     diags1 = [one_list1 sparse_list -1*one_list2];
73
74     diags2 = [one_list1 sparse_list -1*one_list2];
75
76
77     A1 = spdiags(diags1, -(m+1)-1:-(m+1)+1, (m+1)*(n+1), (m+1)*(n+1));
78
79     A2 = spdiags(diags2, (m+1)-1:(m+1)+1, (m+1)*(n+1), (m+1)*(n+1));
80
81     A = A1+A2;
82
83     matrixdxy = ((A)/(4*dy*dy))';
84 end
85
86 function A = comp_matrix(yee, m, n)
87     bc_matrix = sparse(m+1,n+1);
88     bc_matrix(1,:) = 1;
89     bc_matrix(end,:) = 1;
90     bc_matrix(:,1) = 1;
91     bc_matrix(:,end) = 1;
92
93     bc_list = reshape(bc_matrix, (m+1)*(n+1),1);
94     bc_matrix = repmat(bc_list, 1, (m+1)*(n+1));
95
96     if yee=="x"
97         A = bc_matrix;
98     else
99         A = bc_matrix';
100     end
101 end

```


Including Close-Field Boundary Conditions into $A = \left(\frac{\partial}{\partial t}\right)_{i,j}$

As mentioned previously, the equations dictating the boundary conditions on the x-axis and y-axis, in order, are...

$$F_t = -rx F_x - \frac{1}{2}x^2 \sigma^2(1,1) F_{xx} + rF$$

$$F_t = -ry F_y - \frac{1}{2}y^2 \sigma^2(2,2) F_{yy} + rF$$

The MATLAB code below displays the incorporation of the close-field boundary conditions...

```

1 %% Incorporating Close-Field Boundary Conditions into A
2
3 Fx_1D = Derivative_1D_Matrix(m,dx);           % 2nd Order 1D Scheme for
    First Derivative with Respect to X
4 Fy_1D = Derivative_1D_Matrix(n,dy);           % 2nd Order 1D Scheme for
    First Derivative with Respect to Y
5 Fxx_1D = Double_Derivative_1D_Matrix(m,dx);   % 2nd Order 1D Scheme for
    Second Derivative with Respect to X
6 Fyy_1D = Double_Derivative_1D_Matrix(n,dy);   % 2nd Order 1D Scheme for
    Second Derivative with Respect to Y
7
8 Xmatrix_1D = diag(xgrid');
9 Ymatrix_1D = diag(ygrid');
10
11 I_1D = speye(m+1,n+1);                       % Origin and Far-Field
    Boundary Conditions Are Later Addressed
12 I_1D(end,end) = 0;
13 I_1D(1,1) = 0;
14
15 xaxis = ((-r * Xmatrix_1D*Fx_1D) - (1/2 * omega11^2 * Xmatrix_1D*Xmatrix_1D *
    Fxx_1D) + r*I_1D);
16 yaxis = ((-r * Ymatrix_1D*Fy_1D) - (1/2 * omega11^2 * Ymatrix_1D*Ymatrix_1D *
    Fyy_1D) + r*I_1D);
17
18
19 A(1:m+1, 1:n+1) = sparse(xaxis);             % Inserting Close-Field
    Boundary Condition for X-Axis into A
20
21 row_insert = [1:m+1:(m+1)*(n+1)];            % Resizing Y to be
    Inserted Into A Matrix
22 yaxis_matrix1 = sparse((m+1)*(n+1),m+1);
23 yaxis_matrix1(row_insert,:) = yaxis;
24 col_insert = [1:n+1:(n+1)*(m+1)];
25 yaxis_matrix2 = sparse((m+1)*(n+1),(m+1)*(n+1));
26 yaxis_matrix2(:, col_insert) = yaxis_matrix1;
27
28 A = A+yaxis_matrix2;                         % Inserting Close-Field
    Boundary Condition for Y-Axis into A

```

Incorporation of the Close-Field Boundary Conditions

```

1 function matrixdx = Derivative_1D_Matrix(m,dx)

```

```

2   one = ones(m+1,1);
3   sparse_m = sparse(m+1,1);
4   A = spdiags([-1*one sparse_m one], -1:1, m+1, m+1);
5   A(1,:) = sparse_m';
6   A(end,:) = sparse_m';
7
8   matrixdx = (A)/(2*dx);
9 end
10
11 function matrixdxx = Double_Derivative_1D_Matrix(m,dx)
12     one = ones(m+1,1);
13     sparse_m = sparse(m+1,1);
14     A = spdiags([one -2*one one], -1:1, m+1, m+1);
15     A(1,:) = sparse_m';
16     A(end,:) = sparse_m';
17
18     matrixdxx = (A)/(dx^2);
19 end

```

Functions Used for Incorporation of the Close-Field Boundary Conditions

Adjusting $A = \left(\frac{\partial}{\partial t}\right)_{i,j}$ to Account for Far-Field Boundary Conditions

As the value of the far-field boundary conditions are predefined, ones will be inserted into the diagonals of the A matrix corresponding to the positions of the far-field boundary.

The following MATLAB code shows such implementation...

```

1 %% Updating A to Account for Far-Field Dirichlet Boundary Conditions
2
3 dirichlet_far = zeros((m+1),(n+1));
4 dirichlet_far(end,:) = ones(length(xgrid),1);
5 dirichlet_far(:,end) = ones(length(ygrid),1);
6
7 dirichlet_far = diag(reshape(dirichlet_far, 1, (m+1)*(n+1)));
8
9 A = sparse(A+dirichlet_far); % Values Corresponding to Far-
    Field Boundary in A Are One on Diagonal
10 A(1,1) = 1; % Origin is Always Zero

```

Updating A for Far-Field Boundary Conditions

Discretization in Time

Given that the Black-Scholes model approximates the first derivative with respect to time (F_t), the true payoff function F will need to be approximated through time as well.

Rewriting in the form of $(Ax + b)$, where A is the constructed matrix approximating $(\frac{\partial}{\partial t})_{i,j}^n$ including boundary conditions, U^n is a vector the value of the F at time, n , and b^n is a vector of the dirichlet boundary values at time n , gives...

$$\left(\frac{\partial U}{\partial t}\right)^n = AU^n - b^n$$

Second-Order Implicit Euler Scheme

Since the value is discounted back to the present, the equation above can be rewritten in terms of U^{n-1} as...

$$\left(\frac{\partial U}{\partial t}\right)^{n-1} \approx \frac{U^n - U^{n-1}}{\Delta t} + \frac{\Delta t}{2} \frac{U^{n-1} - 2U^n + U^{n+1}}{\Delta t^2} = AU^{n-1} - b^{n-1}$$

Solving for U^{n-1} ...

$$\frac{U^n}{\Delta t} + \frac{\Delta t}{2} \frac{U^{n-1} - 2U^n + U^{n+1}}{\Delta t^2} + b^{n-1} = AU^{n-1} + \frac{U^{n-1}}{\Delta t} - \frac{U^{n-1}}{2\Delta t}$$

$$\frac{U^n}{\Delta t} + \frac{\Delta t}{2} \frac{U^{n-1} - 2U^n + U^{n+1}}{\Delta t^2} + b^{n-1} = \left(A + \frac{1}{2\Delta t}\right) (U^{n-1})$$

$$U^{n-1} = \left(A + \frac{1}{2\Delta t}\right)^{-1} \left(\frac{U^n}{\Delta t} + \frac{\Delta t}{2} \frac{U^{n-1} - 2U^n + U^{n+1}}{\Delta t^2} + b^{n-1}\right)$$

However, because the first time-step is not defined, it will be approximated by the first-order implicit scheme...

$$\frac{U^n - U^{n-1}}{\Delta t} = AU^{n-1} - b^{n-1}$$

$$U^{n-1} = \left(A + \frac{1}{\Delta t}\right)^{-1} \left(\frac{U^n}{\Delta t} + b^{n-1}\right)$$

This will result in the scheme being only first-order accurate, which will be analyzed in later sections.

The following code shows the implementation of the time discretization...

```
1 %% 1st Order Time Scheme to Calculate U After First Time Step
2
3 U = reshape(ICV, (m+1)*(n+1), 1);
4
5 U_minus = U;
```

```

6 BC_minus = BC;
7
8 U = inv((speye(size(A))+(dt*A)))*U_minus+(dt*BC_minus);
9
10 U = U-BC;
11
12 BC = reshape(BC,(m+1),(n+1));
13 upperx = ((xgrid+d)/2)-(strike*exp(-r*(T))); % Updating Far-Field Boundary
        Conditions for X
14 uppery = ((b+ygrid)/2)-(strike*exp(-r*(T))); % Updating Far-Field Boundary
        Conditions for Y
15
16 BC(end,:) = upperx;
17 BC(:,end) = uppery;
18 BC = reshape(BC,(m+1)*(n+1), 1);
19
20 U = reshape(U,(m+1),(n+1));
21 U(end,:) = 0;
22 U(:,end) = 0;
23 U = reshape(U,(m+1)*(n+1), 1);
24
25 U = U + BC; % Making Sure Far-Field
        Boundaries Have Correct Value in Case of Rounding Error
26
27 %% Calculate Inverse of Matrix Needed for 2nd Order Implicit Time Scheme
28
29 A_second_order = inv(A+((1/(2*dt))*speye(size(A))));
30
31 %% Time Integration Loop
32 % Note: Value is Being Discounted back to the Present from Exercize Date
33
34 count = 1;
35 len = length(dt : dt : T)-1;
36
37 for t = dt : dt : T-dt
38     fprintf("%f ",count)
39     fprintf("%f \n",len)
40
41     count = count + 1;
42
43     top = (((U-BC))/dt)+((dt/2)*((-2*(U-BC) + (U_minus-BC_minus))/(dt*dt))) +
        BC; % Updating Implicit Scheme Vector
44
45     top = reshape(top,(m+1),(n+1)); % Making Sure Far-Field
        Boundaries Have Correct Value in Case of Rounding Error
46     top(end,:) = 0;
47     top(:,end) = 0;
48     top = reshape(top,(m+1)*(n+1), 1);
49     top = top + BC;
50
51     U_plus = A_second_order*top; % 2nd Order Implicit
        Scheme for Next Time Step
52
53     BC_minus = BC;
54

```

```

55 BC = reshape(BC,(m+1),(n+1));
56 upperx = ((xgrid+d)/2)-(strike*exp(-r*(T-t))); % Updating Far-Field
Boundary Conditions for X
57 uppery = ((b+ygrid)/2)-(strike*exp(-r*(T-t))); % Updating Far-Field
Boundary Conditions for Y
58
59 BC(end,:) = upperx;
60 BC(:,end) = uppery;
61 BC = reshape(BC,(m+1)*(n+1), 1);
62
63 U_plus = reshape(U_plus,(m+1),(n+1));
64 U_plus(end,:) = 0;
65 U_plus(:,end) = 0;
66 U_plus = reshape(U_plus,(m+1)*(n+1), 1);
67
68 U_plus = U_plus + BC;
69
70 U_minus = U; % Updating U_minus and U
for Next Time Step
71 U = U_plus;
72
73 end

```

Time Discretization

Results

As done in the referred paper, this section will focus on the area encompassing $[0 \leq x \leq \frac{5K}{3}]$ and $[0 \leq y \leq \frac{5K}{3}]$ for the approximated F at time, $t = 0$.

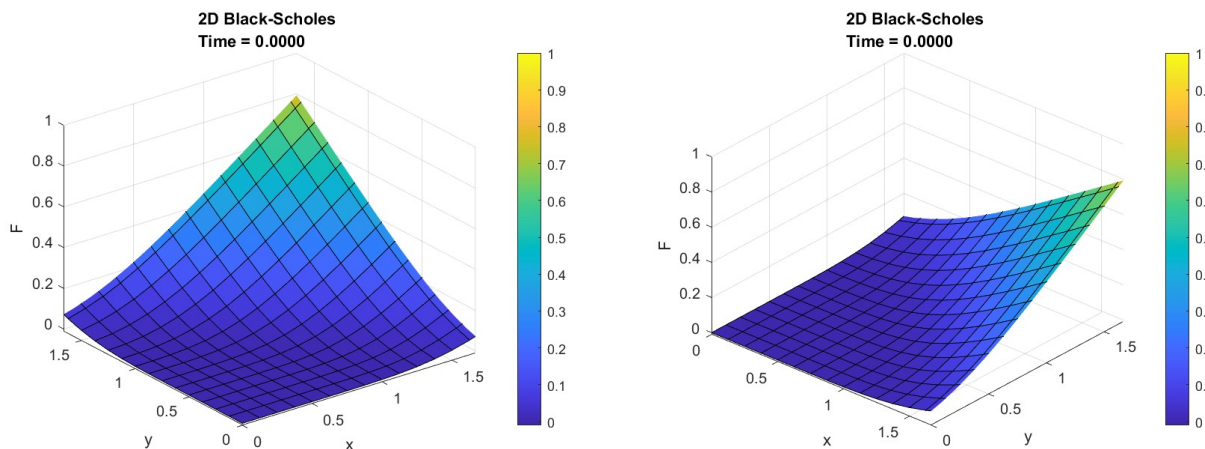


Figure 1: Approximated F around $[0 \leq x \leq \frac{5K}{3}]$ and $[0 \leq y \leq \frac{5K}{3}]$

The figures show a curve similar to the 1D case of the Black-Scholes model, however extrapolated to two dimensions.

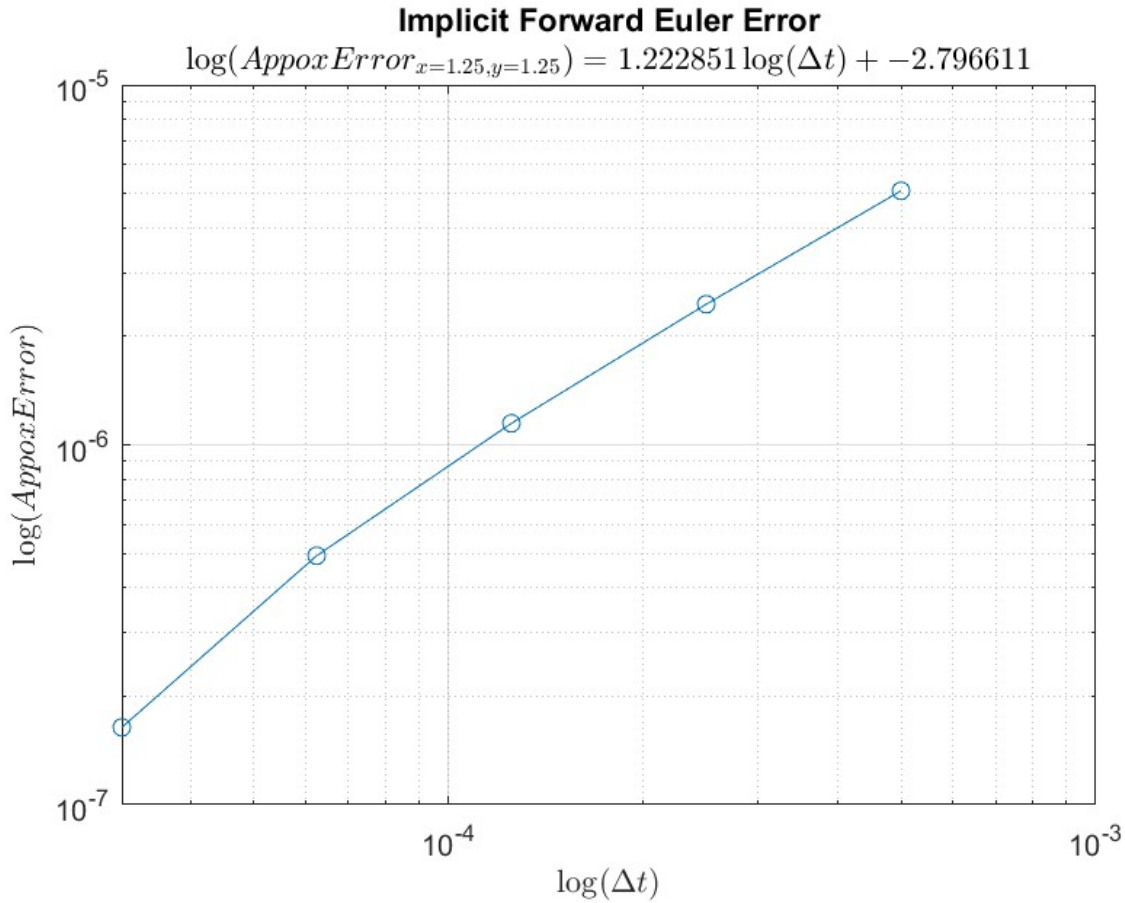
Error Analysis

For the error analysis, based off of the initial $\Delta t = 10^{-3} = 0.001$, I chose the differing Δt list as...

$$\Delta t = \left[\frac{10^{-3}}{2}, \frac{10^{-3}}{4}, \frac{10^{-3}}{8}, \frac{10^{-3}}{16}, \frac{10^{-3}}{32} \right]$$

I chose the time step $\Delta t = \frac{10^{-3}}{64}$ as the most accurate time-step of comparison because the order of accuracy between the time step $\Delta t = \frac{10^{-3}}{32}$ and $\Delta t = \frac{10^{-3}}{64}$ (shown below) was not majorly affected. I also chose the point, $F_{x=1.25, y=1.25}$, instead of the midpoint because since maximum value of x and y can technically be infinite, a value near the strike price, $K = 1$, would be more relevant.

The following log-log plot displays the order of accuracy for the system...



Even though the time discretization was second-order accurate, with the estimation of the initial U^n being only first-order accurate, the first-order accuracy flowed into the system, leaving it only first-order accurate (≈ 1.22).

Personal Reflection

In the future, I would update the system to be second-order accurate by creating a ghost time-step before the time, $t = T$. Due to time constraints and this problem being already rather complex, I did not implement this idea.

I greatly appreciate Jared Brzenski for his guidance for this Black-Scholes model project, as well as his insights for my COMP-670 project for the same model using mimetic operators.

References

- [ST14] Thomas Sundvall and David Trång. “Examination of Impact from Different Boundary Conditions on the 2D Black-Scholes Model: Evaluating Pricing of European Call Options”. In: (Aug. 2014).

Appendix: MATLAB Code

```

1 %% 2D Black-Scholes PDE
2 % Zachary Humphries
3 % COMP 521
4 % Fall 2022
5
6 clear
7 close all
8
9 %% Parameters
10
11 strike = 1;           % Strike Price
12 T = 1;               % Simulation time or Final Maturity Time
13
14 a = 0;               % Minimum Value of Option for Asset X (must be
    zero)
15 b = round(10*strike); % Maximum Value of Option for Asset X per
    recommendation of reference paper (between 8*K and 12*K)
16 c = 0;               % Minimum Value of Option for Asset Y (must be
    zero)
17 d = b;               % Maximum Value of Option for Asset X
18
19 m = 4* round(10*strike); % Personal Preference: Gives Enough Divisions for
    a More Accurate Result
20 n = m;               % Number of cells along the y-axis
21
22 dx = (b-a)/m;        % Step length along the x-axis
23 dy = (d-c)/n;        % Step length along the y-axis
24
25
26 dt = 0.001;          % Personal Preference: Much less than Von Neumann
    stability criterion for explicit scheme  $dx^2/(4)$  (about 0.0039)
27
28 omega11 = 0.3;        % Omega_xx = Omega_yy of the volatility
    correlation matrix
29 omega12 = 0.05;        % Omega_xy = Omega_yx of the volatility
    correlation matrix
30 r = 0.1;              % Risk free interest rate
31
32 %% Setting Up Matrices of F, X, and Y
33
34 xgrid = [a : dx : b];
35 ygrid = [c : dy : d];
36
37 [X, Y] = meshgrid(xgrid, ygrid);
38
39 Xmatrix = diag(reshape(X, (m+1)*(n+1), 1)); % Diagonal Matrix of X
    Mesh for Calculating A
40 Ymatrix = diag(reshape(Y, (m+1)*(n+1), 1)); % Diagonal Matrix of Y
    Mesh for Calculating A
41
42 %% Setting Up Matrix for Fx
43
44 Fx = Fx_Matrix(m,n,dx,dy); % 2nd Order 2D Scheme for

```

```

    First Derivative with Respect to X
45  Fy = Fy_Matrix(m,n,dx,dy);           % 2nd Order 2D Scheme for
    First Derivative with Respect to Y
46
47  Fxx = Fxx_Matrix(m,n,dx,dy);         % 2nd Order 2D Scheme for
    Second Derivative with Respect to X
48  Fyy = Fyy_Matrix(m,n,dx,dy);         % 2nd Order 2D Scheme for
    Second Derivative with Respect to Y
49
50  Fxy = Fxy_Matrix(m,n,dx,dy);         % 2nd Order 2D Scheme for
    Mixed Derivative with Respect to X and Y
51
52  sub_matrix = diag(diag(comp_matrix("x", m, n))); % Matrix to Subtract from
    speye so All Boundary Conditions in A are Zero
53
54 %% Using Black-Scholes PDE to Create A (Excluding Boundary Conditions)
55
56  A = (-r*Xmatrix*Fx) - (r*Ymatrix*Fy) - ((1/2)*omega11^2*Xmatrix*Xmatrix * Fxx)
    - ((1/2)*omega11^2*Ymatrix*Ymatrix * Fyy) - (omega12^2*Xmatrix*Ymatrix*Fxy)
    + (r*(speye((m+1)*(n+1))-sub_matrix));
57
58 %% Encorporating Close-Field Boundary Conditions into A
59
60  Fx_1D = Derivative_1D_Matrix(m,dx);   % 2nd Order 1D Scheme for
    First Derivative with Respect to X
61  Fy_1D = Derivative_1D_Matrix(n,dy);   % 2nd Order 1D Scheme for
    First Derivative with Respect to Y
62  Fxx_1D = Double_Derivative_1D_Matrix(m,dx); % 2nd Order 1D Scheme for
    Second Derivative with Respect to X
63  Fyy_1D = Double_Derivative_1D_Matrix(n,dy); % 2nd Order 1D Scheme for
    Second Derivative with Respect to Y
64
65  Xmatrix_1D = diag(xgrid');
66  Ymatrix_1D = diag(ygrid');
67
68  I_1D = speye(m+1,n+1);                % Origin and Far-Field
    Boundary Conditions Are Later Addressed
69  I_1D(end,end) = 0;
70  I_1D(1,1) = 0;
71
72  xaxis = ((-r * Xmatrix_1D*Fx_1D) - (1/2 * omega11^2 * Xmatrix_1D*Xmatrix_1D *
    Fxx_1D) + r*I_1D);
73  yaxis = ((-r * Ymatrix_1D*Fy_1D) - (1/2 * omega11^2 * Ymatrix_1D*Ymatrix_1D *
    Fyy_1D) + r*I_1D);
74
75
76  A(1:m+1, 1:n+1) = sparse(xaxis);      % Inserting Close-Field
    Boundary Condition for X-Axis into A
77
78  row_insert = [1:m+1:(m+1)*(n+1)];     % Resizing Y to be
    Inserted Into A Matrix
79  yaxis_matrix1 = sparse((m+1)*(n+1),m+1);
80  yaxis_matrix1(row_insert,:) = yaxis;
81  col_insert = [1:n+1:(n+1)*(m+1)];
82  yaxis_matrix2 = sparse((m+1)*(n+1),(m+1)*(n+1));

```

```

83 yaxis_matrix2(:, col_insert) = yaxis_matrix1;
84
85 A = A+yaxis_matrix2; % Inserting Close-Field
    Boundary Condition for Y-Axis into A
86
87
88 %% Updating A to Account for Far-Field Dirichlet Boundary Conditions
89
90 dirichlet_far = zeros((m+1),(n+1));
91 dirichlet_far(end,:) = ones(length(xgrid),1);
92 dirichlet_far(:,end) = ones(length(ygrid),1);
93
94 dirichlet_far = diag(reshape(dirichlet_far, 1, (m+1)*(n+1)));
95
96 A = sparse(A+dirichlet_far); % Values Corresponding to Far-
    Field Boundary in A Are One on Diagonal
97 A(1,1) = 1; % Origin is Always Zero
98
99 %% Creating Far-Field Dirichlet Boundary Condition Values
100
101 BC = zeros(m+1, n+1);
102
103 upper_y = ((b+ygrid)/2)-(strike*exp(-r*(0))); % Updating Boundary Conditions
104 upper_x = ((xgrid+d)/2)-(strike*exp(-r*(0))); % Updating Boundary Conditions
105 BC(end,:) = upper_x;
106 BC(:,end) = upper_y;
107
108 BC = reshape(BC, (m+1)*(n+1), 1);
109
110 %% Initial Values for time = T
111
112 ICV = max(((X+Y)/2)-strike, 0);
113
114 %% 1st Order Time Scheme to Calculate U After First Time Step
115
116 U = reshape(ICV, (m+1)*(n+1), 1);
117
118 U_minus = U;
119 BC_minus = BC;
120
121 U = inv((speye(size(A))+(dt*A)))*U_minus+(dt*BC_minus);
122
123 U = U-BC;
124
125 BC = reshape(BC, (m+1),(n+1));
126 upper_x = ((xgrid+d)/2)-(strike*exp(-r*(T))); % Updating Far-Field Boundary
    Conditions for X
127 upper_y = ((b+ygrid)/2)-(strike*exp(-r*(T))); % Updating Far-Field Boundary
    Conditions for Y
128
129 BC(end,:) = upper_x;
130 BC(:,end) = upper_y;
131 BC = reshape(BC, (m+1)*(n+1), 1);
132
133 U = reshape(U, (m+1),(n+1));

```

```

134 U(end,:) = 0;
135 U(:,end) = 0;
136 U = reshape(U,(m+1)*(n+1), 1);
137
138 U = U + BC; % Making Sure Far-Field
    Boundaries Have Correct Value in Case of Rounding Error
139
140 %% Calculate Inverse of Matrix Needed for 2nd Order Implicit Time Scheme
141
142 A_second_order = inv(A+((1/(2*dt))*speye(size(A))));
143
144 %% Time Integration Loop
145 % Note: Value is Being Discounted back to the Present from Exercise Date
146
147 count = 1;
148 len = length(dt : dt : T)-1;
149
150 for t = dt : dt : T-dt
151     fprintf("%f ",count)
152     fprintf("%f \n",len)
153
154     count = count + 1;
155
156     top = (((U-BC))/dt)+((dt/2)*((-2*(U-BC) + (U_minus-BC_minus))/(dt*dt)) +
    BC); % Updating Implicit Scheme Vector
157
158     top = reshape(top,(m+1),(n+1)); % Making Sure Far-Field
    Boundaries Have Correct Value in Case of Rounding Error
159     top(end,:) = 0;
160     top(:,end) = 0;
161     top = reshape(top,(m+1)*(n+1), 1);
162     top = top + BC;
163
164     U_plus = A_second_order*top; % 2nd Order Implicit
    Scheme for Next Time Step
165
166     BC_minus = BC;
167
168     BC = reshape(BC,(m+1),(n+1));
169     upperx = ((xgrid+d)/2)-(strike*exp(-r*(T-t))); % Updating Far-Field
    Boundary Conditions for X
170     uppery = ((b+ygrid)/2)-(strike*exp(-r*(T-t))); % Updating Far-Field
    Boundary Conditions for Y
171
172     BC(end,:) = upperx;
173     BC(:,end) = uppery;
174     BC = reshape(BC,(m+1)*(n+1), 1);
175
176     U_plus = reshape(U_plus,(m+1),(n+1));
177     U_plus(end,:) = 0;
178     U_plus(:,end) = 0;
179     U_plus = reshape(U_plus,(m+1)*(n+1), 1);
180
181     U_plus = U_plus + BC;
182

```

```

183     U_minus = U; % Updating U_minus and U
184     for Next Time Step
185         U = U_plus;
186     end
187
188 %% Graphing Final U
189
190 U_graph = max(U_plus, 0); % Option Value doesn't go
    below zero
191 surf(X, Y, reshape(U_graph, m+1, n+1))
192 title(['2D Black-Scholes \newlineTime = ' num2str(T-t-dt, '%1.4f')])
193 xlabel('x')
194 ylabel('y')
195 zlabel('F')
196 colorbar
197 caxis([-0.01 strike])
198 axis([0 5*strike/3 0 5*strike/3 -0.01 strike]) % Examining Boundary Up to
    5*strike/3 as Done in Paper
199
200 % caxis([-0.05, (b+d-strike)/2]) % Uncomment for Graph of
    All of U
201 % axis([a b c d -0.05 (b+d-strike)/2])
202 drawnow
203
204 %% Error Testing
205 mult_list = [1/2, 1/4, 1/8, 1/16, 1/32, 1/64];
206 value_list = error_testing(mult_list, strike, T, dx, dy, omega11, omega12, r);
207
208 value_list_adj = abs((value_list(1:(end-1))-value_list(end)));
209
210 dt_list = dt*mult_list(1:(end-1));
211
212 %% Error plotting
213 figure
214 forward_poly = polyfit(log(dt_list), log(value_list_adj), 1);
215 loglog(dt_list, value_list_adj, "o-"); grid on;
216 title("Implicit Forward Euler Error")
217 subtitle_name_forward = strcat("$\log(\text{Appox Error}_{x=1.25,y=1.25}) = ",
    sprintf("%2.6f", forward_poly(1)), "\log(\Delta t) + ", sprintf("%2.6f",
    forward_poly(2)), "$");
218 subtitle(subtitle_name_forward, 'interpreter', 'latex')
219 xlabel("$\log(\Delta t)$", 'interpreter', 'latex')
220 ylabel("$\log(\text{Appox Error})$", 'interpreter', 'latex')
221
222 %% Functions
223
224 function matrixdx = Derivative_1D_Matrix(m,dx)
225     one = ones(m+1,1);
226     sparse_m = sparse(m+1,1);
227     A = spdiags([-1*one sparse_m one], -1:1, m+1, m+1);
228     A(1,:) = sparse_m';
229     A(end,:) = sparse_m';
230
231     matrixdx = (A)/(2*dx);

```

```

232 end
233
234 function matrixdxx = Double_Derivative_1D_Matrix(m,dx)
235     one = ones(m+1,1);
236     sparse_m = sparse(m+1,1);
237     A = spdiags([one -2*one one], -1:1,m+1,m+1);
238     A(1,:) = sparse_m';
239     A(end,:) = sparse_m';
240
241     matrixdxx = (A)/(dx^2);
242 end
243
244 function matrixdx = Fx_Matrix(m,n,dx,dy)
245     one = ones(m+1,1);
246     sparse_m = sparse(m+1,1);
247     A = spdiags([-1*one sparse_m one], -1:1,m+1,m+1);
248     A(1,:) = sparse_m';
249     A(end,:) = sparse_m';
250
251     sparse_y = speye(n+1,n+1);
252     sparse_n = sparse(n+1,1);
253     sparse_y(:,1) = sparse_n;
254     sparse_y(:,end) = sparse_n;
255
256     matrixdx = kron(sparse_y, A)/(2*dx);
257 end
258
259 function matrixdy = Fy_Matrix(m,n,dx,dy)
260     one = ones(n+1,1);
261     sparse_n = sparse(n+1,1);
262     one = sparse([0;one;0]);
263     one_list = repmat(one,m-1,1);
264     one_list1 = [sparse_n; one_list; sparse_n];
265     one_list2 = [sparse_n; one_list; sparse_n];
266
267     A = spdiags([-1*one_list1 repmat(sparse_n, n+1, 2*(m+1)-1) one_list2], -(m
+1):(m+1), (m+1)*(n+1), (m+1)*(n+1));
268
269     matrixdy = -1*(((A)/(2*dy))');
270 end
271
272 function matrixdxx = Fxx_Matrix(m,n,dx,dy)
273     one = ones(m+1,1);
274     sparse_m = sparse(m+1,1);
275     A = spdiags([one -2*one one], -1:1,m+1,m+1);
276     A(1,:) = sparse_m';
277     A(end,:) = sparse_m';
278
279     sparse_y = speye(n+1,n+1);
280     sparse_n = sparse(n+1,1);
281     sparse_y(:,1) = sparse_n;
282     sparse_y(:,end) = sparse_n;
283
284     matrixdxx = kron(sparse_y, A)/(dx^2);
285 end

```

```

286
287 function matrixdyy = Fyy_Matrix(m,n,dx,dy)
288     one = ones(n-1,1);
289     sparse_n = sparse(n+1,1);
290     one = sparse([0;one;0]);
291     one_list = repmat(one,m-1,1);
292     one_list1 = [sparse_n; one_list; sparse_n];
293     one_list2 = [sparse_n; one_list; sparse_n];
294     diag = [sparse_n; one_list; sparse_n];
295
296     A = spdiags([one_list1 repmat(sparse_n, n+1, m) -2*diag repmat(sparse_n, n
+1, m) one_list2], -(m+1):(m+1), (m+1)*(n+1), (m+1)*(n+1));
297
298     matrixdyy = ((A)/(dy^2))';
299 end
300
301 function matrixdxy = Fxy_Matrix(m,n,dx,dy)
302     one = ones(n-1,1);
303     sparse_n = sparse(n+1,1);
304     one1 = sparse([0;one;0]);
305     one2 = sparse([0;one;0]);
306     one_list = repmat(one1,m-1,1);
307     one_list2 = repmat(one2,m-1,1);
308     one_list1 = [sparse_n; one_list; sparse_n];
309     one_list2 = [sparse_n; one_list2; sparse_n];
310     one_list3 = [sparse_n; one_list2; sparse_n];
311     one_list4 = [sparse_n; one_list; sparse_n];
312
313     sparse_list = repmat(sparse_n,m+1,1);
314
315     diags1 = [one_list1 sparse_list -1*one_list2];
316
317     diags2 = [one_list1 sparse_list -1*one_list2];
318
319
320     A1 = spdiags(diags1, -(m+1)-1:-(m+1)+1, (m+1)*(n+1), (m+1)*(n+1));
321
322     A2 = spdiags(diags2, (m+1)-1:(m+1)+1, (m+1)*(n+1), (m+1)*(n+1));
323
324     A = A1+A2;
325
326     matrixdxy = ((A)/(4*dy*dy))';
327 end
328
329 function A = comp_matrix(yee, m, n)
330     bc_matrix = sparse(m+1,n+1);
331     bc_matrix(1,:) = 1;
332     bc_matrix(end,:) = 1;
333     bc_matrix(:,1) = 1;
334     bc_matrix(:,end) = 1;
335
336     bc_list = reshape(bc_matrix, (m+1)*(n+1),1);
337     bc_matrix = repmat(bc_list, 1, (m+1)*(n+1));
338
339     if yee=="x"

```

```

340     A = bc_matrix;
341     else
342     A = bc_matrix';
343     end
344 end
345
346 function error_list = error_testing(mult_list, strike, T, dx, dy, omega11,
    omega12, r)
347     error_list = zeros(length(mult_list),1);
348     for ii = [1:length(mult_list)]
349         mult = mult_list(ii);
350         %% Spatial discretization
351
352         a = 0; % Minimum Value of Option for Asset X (
must be zero)
353         b = round(10*strike); % Maximum Value of Option for Asset X
354         c = 0; % Minimum Value of Option for Asset Y (
must be zero)
355         d = b; % Maximum Value of Option for Asset X
356
357         m = 8* round(10*strike); % Personal Preference: Gives Enough
Divisions for a More Accurate Result
358         n = m; % Number of cells along the y-axis
359
360         dx = (b-a)/m; % Step length along the x-axis
361         dy = (d-c)/n; % Step length along the y-axis
362
363
364         dt = 0.001 * mult;
365
366         m = 8* round(10*strike); % 2*k+1 = Minimum number of cells to
attain the desired accuracy
367         n = m; % Number of cells along the y-axis
368
369         dx = ((b-a)/m); % Step length along the x-axis
370         dy = ((d-c)/n); % Step length along the y-axis
371
372
373         %% Setting Up Matricies of F, X, and Y
374
375         xgrid = [a : dx : b];
376         ygrid = [c : dy : d];
377
378         [X, Y] = meshgrid(xgrid, ygrid);
379
380         Xmatrix = diag(reshape(X, (m+1)*(n+1), 1)); % Diagonal
Matrix of X Mesh for Calculating A
381         Ymatrix = diag(reshape(Y, (m+1)*(n+1), 1)); % Diagonal
Matrix of Y Mesh for Calculating A
382
383         %% Setting Up Matrix for Fx
384
385         Fx = Fx_Matrix(m,n,dx,dy); % 2nd Order 2D
Scheme for First Derivative with Respect to X
386         Fy = Fy_Matrix(m,n,dx,dy); % 2nd Order 2D

```


Scheme for First Derivative with Respect to Y

```
387 Fxx = Fxx_Matrix(m,n,dx,dy); % 2nd Order 2D
```

Scheme for Second Derivative with Respect to X

```
389 Fyy = Fyy_Matrix(m,n,dx,dy); % 2nd Order 2D
```

Scheme for Second Derivative with Respect to Y

```
390 Fxy = Fxy_Matrix(m,n,dx,dy); % 2nd Order 2D
```

Scheme for Mixed Derivative with Respect to X and Y

```
392 sub_matrix = diag(diag(comp_matrix("x", m, n))); % Matrix to
393 Subtract from speye so All Boundary Conditions in A are Zero
```

% Using Black-Scholes PDE to Create A (Excluding Boundary Conditions)

```
396 A = (-r*Xmatrix*Fx) - (r*Ymatrix*Fy) - ((1/2)*omega11^2*Xmatrix*
397 Xmatrix * Fxx) - ((1/2)*omega11^2*Ymatrix*Ymatrix * Fyy) - (omega12^2*
Xmatrix*Ymatrix*Fxy) + (r*(speye((m+1)*(n+1))-sub_matrix));
```

% Incorporating Close-Field Boundary Conditions into A

```
400 Fx_1D = Derivative_1D_Matrix(m,dx); % 2nd Order 1D
```

Scheme for First Derivative with Respect to X

```
402 Fy_1D = Derivative_1D_Matrix(n,dy); % 2nd Order 1D
```

Scheme for First Derivative with Respect to Y

```
403 Fxx_1D = Double_Derivative_1D_Matrix(m,dx); % 2nd Order 1D
```

Scheme for Second Derivative with Respect to X

```
404 Fyy_1D = Double_Derivative_1D_Matrix(n,dy); % 2nd Order 1D
```

Scheme for Second Derivative with Respect to Y

```
405 Xmatrix_1D = diag(xgrid');
```

```
407 Ymatrix_1D = diag(ygrid');
```

```
409 I_1D = speye(m+1,n+1); % Origin and Far-
Field Boundary Conditions Are Later Addressed
```

```
410 I_1D(end,end) = 0;
```

```
411 I_1D(1,1) = 0;
```

```
413 xaxis = ((-r * Xmatrix_1D*Fx_1D) - (1/2 * omega11^2 * Xmatrix_1D*
Xmatrix_1D * Fxx_1D) + r*I_1D);
```

```
414 yaxis = ((-r * Ymatrix_1D*Fy_1D) - (1/2 * omega11^2 * Ymatrix_1D*
Ymatrix_1D * Fyy_1D) + r*I_1D);
```

```
416 A(1:m+1, 1:n+1) = sparse(xaxis); % Inserting Close-
Field Boundary Condition for X-Axis into A
```

```
418 row_insert = [1:m+1:(m+1)*(n+1)]; % Resizing Y to be
Inserted Into Matrix
```

```
420 yaxis_matrix1 = sparse((m+1)*(n+1),m+1);
```

```
421 yaxis_matrix1(row_insert,:) = yaxis;
```

```
422 col_insert = [1:n+1:(n+1)*(m+1)];
```

```
423 yaxis_matrix2 = sparse((m+1)*(n+1),(m+1)*(n+1));
```

```
424 yaxis_matrix2(:, col_insert) = yaxis_matrix1;
```

```

426     A = A+yaxis_matrix2; % Inserting Close-
    Field Boundary Condition for Y-Axis into A
427
428
429     %% Updating A to Account for Far-Field Dirichlet Boundary Conditions
430
431     dirichlet_far = zeros((m+1),(n+1));
432     dirichlet_far(end,:) = ones(length(xgrid),1);
433     dirichlet_far(:,end) = ones(length(ygrid),1);
434
435     dirichlet_far = diag(reshape(dirichlet_far, 1, (m+1)*(n+1)));
436
437     A = sparse(A+dirichlet_far); % Values Corresponding
    to Far-Field Boundary in A Are One on Diagonal
438     A(1,1) = 1; % Origin is Always
    Zero
439
440     %% Creating Far-Field Dirichlet Boundary Condition Values
441
442     BC = zeros(m+1, n+1);
443
444     upperry = ((b+ygrid)/2)-(strike*exp(-r*(0))); % Updating Boundary
    Conditions
445     upperx = ((xgrid+d)/2)-(strike*exp(-r*(0))); % Updating Boundary
    Conditions
446     BC(end,:) = upperx;
447     BC(:,end) = upperry;
448
449     BC = reshape(BC,(m+1)*(n+1), 1);
450
451     %% Initial Values for time = T
452
453     ICV = max(((X+Y)/2)-strike, 0);
454
455     %% 1st Order Time Scheme to Calculate U After First Time Step
456
457     U = reshape(ICV, (m+1)*(n+1), 1);
458
459     U_minus = U;
460     BC_minus = BC;
461
462     U = inv((speye(size(A))+(dt*A)))*U_minus+(dt*BC_minus);
463
464     U = U-BC;
465
466     BC = reshape(BC,(m+1),(n+1));
467     upperx = ((xgrid+d)/2)-(strike*exp(-r*(T))); % Updating Far-Field
    Boundary Conditions for X
468     upperry = ((b+ygrid)/2)-(strike*exp(-r*(T))); % Updating Far-Field
    Boundary Conditions for Y
469
470     BC(end,:) = upperx;
471     BC(:,end) = upperry;
472     BC = reshape(BC,(m+1)*(n+1), 1);
473

```

```

474     U = reshape(U,(m+1),(n+1));
475     U(end,:) = 0;
476     U(:,end) = 0;
477     U = reshape(U,(m+1)*(n+1), 1);
478
479     U = U + BC; % Making Sure Far-
Field Boundaries Have Correct Value in Case of Rounding Error
480
481     %% Calculate Inverse of Matrix Needed for 2nd Order Implicit Time
Scheme
482
483     A_second_order = inv(A+((1/(2*dt))*speye(size(A))));
484
485     %% Time Integration Loop
486     % Note: Value is Being Discounted back to the Present from Exersize
Date
487
488     count = 1;
489     len = length(dt : dt : T)-1;
490     fprintf("\n %f \n",mult)
491     for t = dt : dt : T-dt
492         fprintf("%f ",count)
493         fprintf("%f \n",len)
494
495         count = count + 1;
496
497         top = (((U-BC))/dt)+((dt/2)*((-2*(U-BC) + (U_minus-BC_minus))/(dt
*dt)))) + BC); % Updating Implicit Scheme Vector
498
499         top = reshape(top,(m+1),(n+1)); % Making Sure Far-
Field Boundaries Have Correct Value in Case of Rounding Error
500         top(end,:) = 0;
501         top(:,end) = 0;
502         top = reshape(top,(m+1)*(n+1), 1);
503         top = top + BC;
504
505         U_plus = A_second_order*top; % 2nd Order
Implicit Scheme for Next Time Step
506
507         BC_minus = BC;
508
509         BC = reshape(BC,(m+1),(n+1));
510         upperx = ((xgrid+d)/2)-(strike*exp(-r*(T-t))); % Updating Far-
Field Boundary Conditions for X
511         uppery = ((b+ygrid)/2)-(strike*exp(-r*(T-t))); % Updating Far-
Field Boundary Conditions for Y
512
513         BC(end,:) = upperx;
514         BC(:,end) = uppery;
515         BC = reshape(BC,(m+1)*(n+1), 1);
516
517         U_plus = reshape(U_plus,(m+1),(n+1));
518         U_plus(end,:) = 0;
519         U_plus(:,end) = 0;
520         U_plus = reshape(U_plus,(m+1)*(n+1), 1);

```

```
521         U_plus = U_plus + BC;
522
523         U_minus = U;                                     % Updating U_minus
524     and U for Next Time Step
525         U = U_plus;
526
527     end
528     U_final = max(reshape(U_plus, m+1, n+1), 0);
529     % Option Value doesn't go below zero
530     x_index = find(xgrid==1.25);
531     y_index = find(ygrid==1.25);
532
533     error_list(ii) = U_final(x_index, y_index);
534 end
535 end
```

COMP_521_Final_Project_Black_Scholes_Zachary_Humphries.m