# MTH 600- Assignment 2

Name: Sophia Rybnik

Date: March 2024

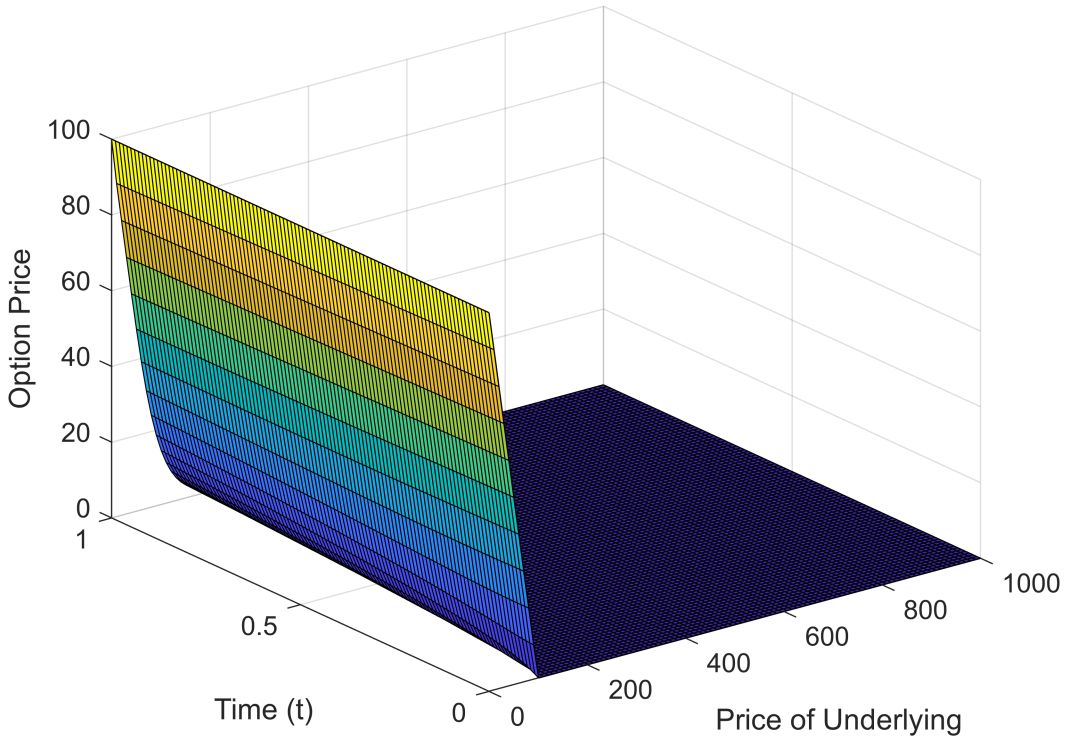
Student Number: 501015789

## Question #1

a) The model uses upstream weighting to ensure a positive coefficient discretization. LU factorization was employed to reduce unnecessary computation. The model alternates between a fully implicit and Crank-Nicolson method, depending on the value of $\theta$ provided.

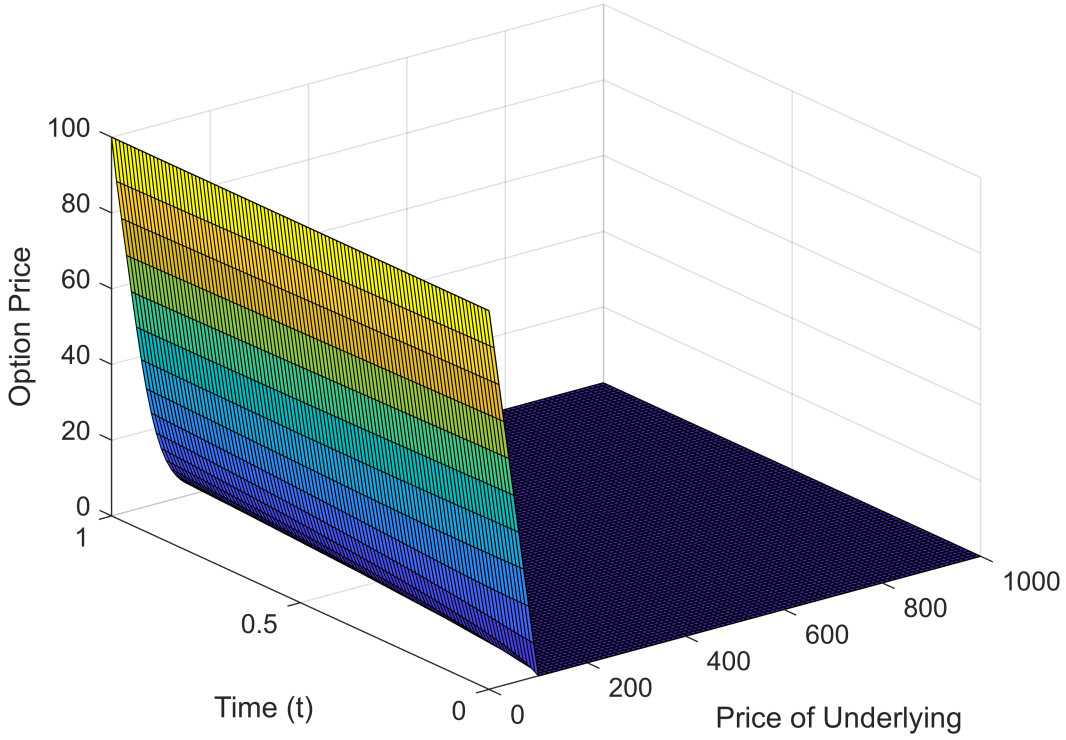
b) Use function $blackscholes\_put\_model$ to price a put option, using minimum and maximum values for the underlying (a stock) as \$0 and \$1000, respectively. The option is priced under the assumption that the price of the underlying today is at the strike price. We will use a $100 \times 100$ finite grid.

```
clear;
clc;
flagScreenOutput = 1;
S0 = 100;
r = 0.02;
sig = 0.4;
K = 100;
T= 1;
Nt = 100;
S = linspace(0,1000,100);
type = 'put';
theta = 1;

implicit_put = blackscholes_put_model(S0, K, r, sig, T, Nt, S, theta, type,
flagScreenOutput)
```




```
implicit_put = 14.6766
```

```
CN_put = blackscholes_put_model(S0, K, r, sig, T, Nt, S, 0.5, 'put',
flagScreenOutput)
```




```
CN_put = 14.6978
```

```
% check using MATLAB built in Black Scholes model
[exact_call, exact_put]=blsprice(S0,K,r,T,sig);
exact_put
```

```
exact_put = 14.7243
```

Compare with exact solution using built-in Matlab function:

```
error_implicit = abs((implicit_put-exact_put)/exact_put)
```

```
error_implicit = 0.0032
```

```
error_cn = abs((CN_put-exact_put)/exact_put)
```

```
error_cn = 0.0018
```

As shown above, the implicit model prices the given put option today at $14.68. Similarly, the Crank-Nicolson Model prices the put option today at $14.70. Since the exact solution gives a price of $14.72, the relative error of the fully implicit and Crank-Nicolson methods are 0.32% and 0.18%, respectively.

c) The rate of convergence is defined as $lim_{k\to\infty} = \frac{|x_{k+1} - L|}{|x_k - L|} = \mu$. Since the rate of convergence exists, the numercial solution converges linearly to the exact solution.

```
theta = 0.5;
[exact_c, exact_p] = blsprice(S0,K,r,T,sig); % exact solutions
flagScreenOutput = 0;
```

```
S= [0:0.1*K:0.4*K,...
              0.45*K:0.05*K:0.8*K,...
              0.82*K:0.02*K:0.9*K,...
              0.91*K:0.01*K:1.1*K,...
              1.12*K:0.02*K:1.2*K,...
              1.25*K:.05*K:1.6*K,...
              1.7*K:0.1*K:2*K,...
              2.2*K, 2.4*K, 2.8*K,...
              3.6*K, 5*K, 7.5*K, 10*K];

Iteration = [1,2,3];
Nt = 100;
Numerical_Solution(1) = blackscholes_put_model(S0, K, r, sig, T, Nt, S, theta,
type, flagScreenOutput);
Residual(1) = abs(Numerical_Solution(1)-exact_p);
Convergence_Rate(1)=0;
Nt = 200;
S1 = halve(S);
Numerical_Solution(2) = blackscholes_put_model(S0, K, r, sig, T, Nt, S1, theta,
type, flagScreenOutput);
Residual(2) = abs(Numerical_Solution(2)-exact_p);
Convergence_Rate(2)=abs(Numerical_Solution(2)-exact_p)/abs(Numerical_Solution(1)-
exact_p);

Nt = 400;
S2 = halve(S1);
Numerical_Solution(3) = blackscholes_put_model(S0, K, r, sig, T, Nt, S2, theta,
type, flagScreenOutput);
Residual(3) = abs(Numerical_Solution(3)-exact_p);
Convergence_Rate(3)=abs(Numerical_Solution(3)-exact_p)/abs(Numerical_Solution(2)-
exact_p);
(Numerical_Solution(1)-Numerical_Solution(2))/(Numerical_Solution(2)-
Numerical_Solution(3));
Iteration=Iteration';
Numerical_Solution = Numerical_Solution';
Residual=Residual';
Convergence_Rate=Convergence_Rate';

% Print a convergence table
table(Iteration, Numerical_Solution, Residual, Convergence_Rate)
```

ans = 3×4 table

| | Iteration | Numerical_Solution | Residual | Convergence_Rate |
|---|---|---|---|---|
| 1 | 1 | 14.7033 | 0.0210 | 0 |
| 2 | 2 | 14.7190 | 0.0053 | 0.2504 |
| 3 | 3 | 14.7230 | 0.0013 | 0.2501 |

```
fprintf('Exact solution: %.4f', exact_p)
```

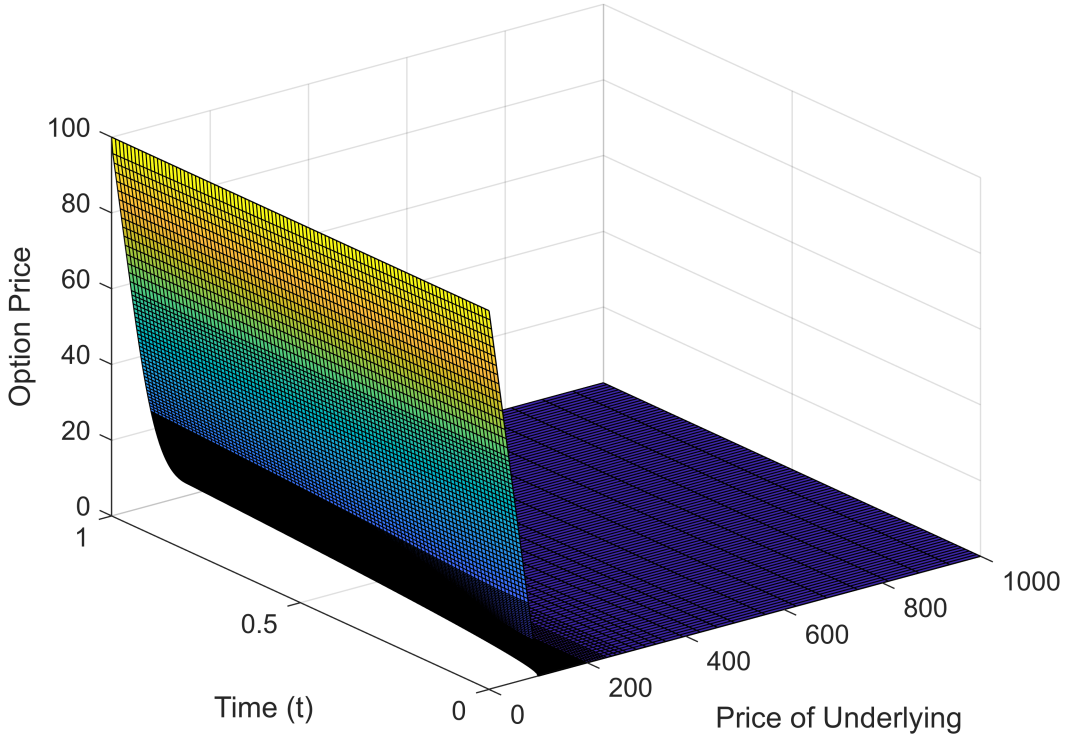
```
Exact solution: 14.7243
```

```
fprintf('(V(h)-V(h/2))/(V(h/2)-V(h/4))= %.2f', (Numerical_Solution(1)-
Numerical_Solution(2))/(Numerical_Solution(2)-Numerical_Solution(3)));
```

```
(V(h)-V(h/2))/(V(h/2)-V(h/4))= 3.99
```

d)

```
Nt = 100;
S = linspace(50,150,100);
type = 'put';
flagScreenOutput = 1;
cn_put = blackscholes_put_model(S0, K, r, sig, T, Nt, S2, theta, type,
flagScreenOutput);
```




## Question #2

a) Solve the PDE as follows:

$$u_t = \frac{u_i^{n+1} - u_i^n}{\Delta t}$$

$$u_{xx} = \frac{1}{2(\Delta x)^2}\left((u_{i-1}^n - 2u_i^n + u_{i+1}^n) + (u_{i-1}^{n+1} - 2u_i^{n+1} + u_{i+1}^{n+1})\right)$$

sub into $u_t = u_{xx}$

$$\Rightarrow u_i^{n+1} = u_i^n + \frac{\Delta t}{2(\Delta x)^2}\left((u_{i-1}^n - 2u_i^n + u_{i+1}^n) + (u_{i-1}^{n+1} - 2u_i^{n+1} + u_{i+1}^{n+1})\right)$$

let $\alpha = \frac{\Delta t}{(\Delta x)^2}$, multiply by 2, and rearrange to group like terms.

$$\Rightarrow -\alpha u_{i-1}^{n+1} + 2(\alpha+1)u_i^{n+1} - \alpha u_{i+1}^{n+1} = \alpha u_{i-1}^n + 2(1-\alpha)u_i^n + \alpha u_{i+1}^n$$

In matrix form, applying boundary conditions $u_0^{n+1} = u_1^{n+1} = 0$ :

$$\begin{bmatrix} 1 & 0 & 0 & 0 & \dots & 0 & 0 \\ -\alpha & 2(\alpha+1) & -\alpha & 0 & \dots & 0 & 0 \\ 0 & -\alpha & 2(\alpha+1) & -\alpha & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & 0 & -\alpha & 2(\alpha+1) & -\alpha \\ 0 & 0 & \dots & \dots & \dots & 0 & 1 \end{bmatrix} \begin{bmatrix} u_0^{n+1} \\ u_1^{n+1} \\ \\ \vdots \\ u_M^{n+1} \end{bmatrix} = \begin{bmatrix} 0 \\ \alpha u_0^n + 2(1-\alpha)u_1^n + \alpha u_2^n \\ \alpha u_1^n + 2(1-\alpha)u_2^n + \alpha u_3^n \\ \vdots \\ \alpha u_{M-2}^n + 2(1-\alpha)u_{M-1}^n + \alpha u_M^n \\ 0 \end{bmatrix}$$

Solve the following linear system:

$$A\vec{U}^{n+1} = \vec{b}^n$$

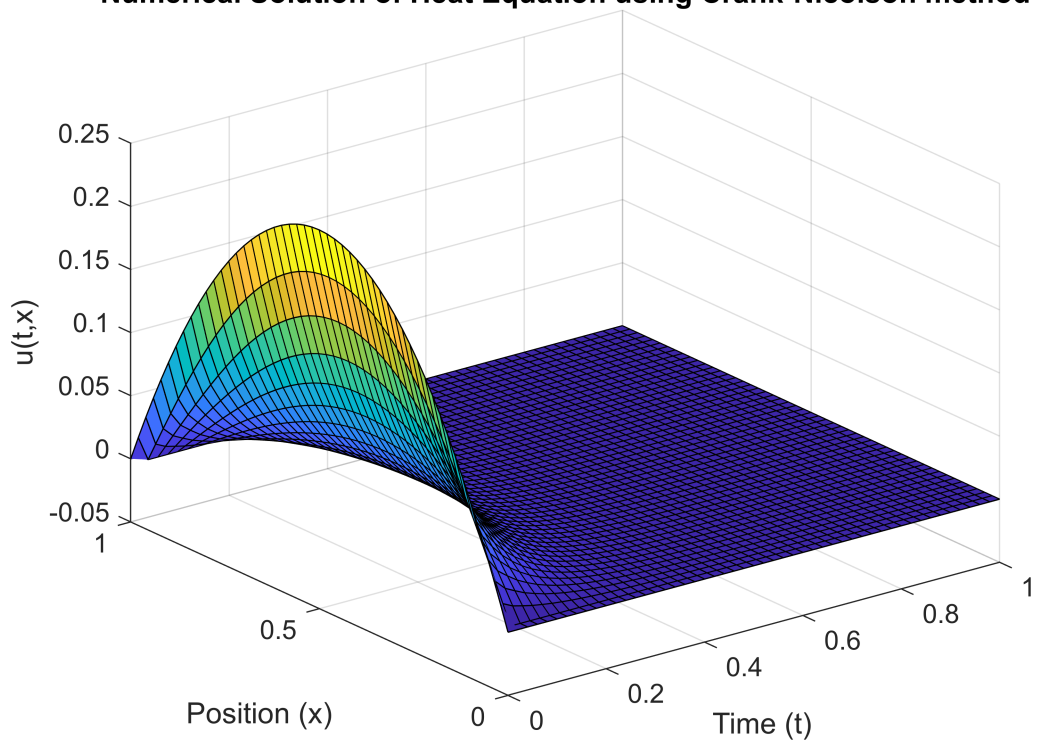
To solve the PDE, the algorithm is as follows:

1. Initialize a matrix of solutions of size $((M+1)\times(N+1))$.
2. Apply boundary conditions and initial conditions.
3. Iterate through each time step starting at $t = 1$. Solve $A\vec{U}^{n+1} = \vec{b}^n$, where A is the matrix of coefficients constructed in the derivation. LU factorization can be applied here to increase speed since A is a tridiagonal matrix.

b)

```
clear;
clc;

% parameters
```

```
L=1; % domain
T=1; % time
M=50; % number of grid points (space)
N=50; % number of grid points (time)
dx = L / M; % space step
dt = T / N; % time step
alpha = dt/(dx^2);

% construct solution grid
u = zeros(M+1,N+1);

% apply initial and boundary conditions
x=linspace(0,L,M+1); % create spacial vector (avoid using loop to set initial
condition)
u(:,1) = -x.^2+x;
u(1,:) = 0;
u(end,:) = 0;


% build out RHS matrix
a(1:M-2)=-alpha;
a(end)=0;
b(2:M-1)=2*(alpha+1);
b(1)=1;
b(end)=1;
c(1:M-2)=-alpha;
c(1)=0;
A=diag(b,0)+diag(a,-1)+diag(c,1);
[L,U]=lu(A); % apply LU factorization since A is tridiagonal

B = zeros(M-1,1);
for i=2:N+1 % iterate through each time step starting from t = 1 (t = 0 already
defined by initial condition)
    for j=3:M-1 % start at position 1 since position 2 already defined by boundary
condition
        % solve for previous time step
        B(j-1)= alpha*u(j-1,i-1)+2*(1-alpha)*u(j,i-1)+alpha*u(j+1,i-1);
    end
    % solve for new time step
    u(2:M, i)= U\(L\B);
end


% plot
[X, T] = meshgrid(x, linspace(0, T, N+1));
surf(T, X, u');
xlabel('Time (t)');
ylabel('Position (x)');
zlabel('u(t,x)');
title('Numerical Solution of Heat Equation using Crank-Nicolson method');
```




```
function price = blackscholes_put_model(S0, K, r, sig, T, Nt, S, theta, type,
flagScreenOutput)
% Prices European call option using implicit and CN finite difference method
% Inputs:
% S0 - the price of the underlying today, at which we want to value the option at
% K - strike price
% S - vector of stock prices
% r - risk free interest rate
% sig - volatility of underlying
% T - expiry time (years)
% Nt - number of grid points (time)
% type - 'put' or 'call'
% theta - parameter to determine if CN or implicit method is used -- 1 = implicit,
0.5 = CN
% flagScreenOutput - set to 1 to output option plot, 0 otherwise
% Output: price - option price at S0
  %  S = unique(S);
    S_max = max(S);
    S_min = min(S);
    NS = length(S);
```

```
    dt = T/(Nt-1); % time step
    t = linspace(0, T, Nt);  % time discretization

    V = zeros(NS, Nt);  % grid initialization

    if strcmp('call',type)
        % expiry time boundary condition (terminal condition)
        payoff = max(S-K, 0);
        V(:, 1)= payoff;
        % minimum and maximum price (boundary conditions)
        % V(1,:) = 0  -- when stock price is zero -- already specified by grid
initialization of all zeros
        V(end,:)=(S_max -K)./(1-r*t);

    else
        % expiry time boundary condition (terminal condition)
        payoff = max(K-S, 0);
        V(:, 1)=payoff;
        % minimum and maximum price (boundary conditions)
        V(1,:) = (K-S_min)./(1-r*t); % when stock price is at minimum, you receive
amount of strike price
       % V(end,:)= 0; %-- as stock price tends to infinity, value of put is zero
since it will not be exercised -- already specified by grid initialization of all
zeros
    end
    M=zeros(NS, NS);
    sig2 = sig*sig;

    for i=2:NS-1 % price
        % Calculate the coefficients
        alpha = (sig2 * S(i)^2)/((S(i)-S(i-1))*(S(i+1)-S(i-1))) - r*S(i)/(S(i+1)-
S(i-1));
        beta = (sig2 * S(i)^2)/((S(i+1)-S(i))*(S(i+1)-S(i-1))) + r*S(i)/(S(i+1)-
S(i-1));

        % upstream weighting -- use forward difference as needed to ensure positive
coefficient discretization
        if alpha< 0
            alpha = alpha + r*S(i)/(S(i+1)-S(i-1));
            beta = (sig2 * S(i)^2)/((S(i)-S(i-1))*(S(i+1)-S(i-1))) + r*S(i)/(S(i+1)-
S(i));

        end
        % construction of the tri-diagonal matrix D
        M(i,i) = -alpha - beta - r;
        M(i, i-1) = alpha;
        M(i, i+1) = beta;
    end
```

```
   [L U] = lu(eye(NS) - theta*dt*M); % using theta to switch between implicit and
CN method
    B = U\(L\((eye(NS) + (1- theta)*dt*M))); % solve linear system
    % iterate through each time to solve for each V_(m+1)
    for n=2:Nt
       V(:, n) = B*V(:, n-1);
    end

    price=interp1(S, V(:,Nt), S0); % calculates and returns the option price at S0
    if flagScreenOutput
        % plot
        [T,X] = meshgrid(t, S);
        surf(X, T, V);
        xlabel('Price of Underlying');
        ylabel('Time (t)');
        zlabel('Option Price');
    end
end
```

```
function S_halved = halve(S)

    l = length(S);
    S_new = zeros(1,l-1);
    j=2;
    for i = 1:l-1
        S_new(i+j-2)=S(i);
        S_new(i+j)=S(i+1);
        S_new(i+j-1)=(S(i+1)+S(i))/2;
        j=j+1;
    end
    S_halved = S_new;
end
```