Question 2

For the part 1, we implement the constant time stepping function with fixed timesteps and nodes.

Const_stepping.m

```
function [result] = const_timestep(S,N)

alpha_parameter =0.8;     % constant related to alpha
r = .02;                  % risk free rate
T = 1;                    % time to expiry
K = 10;                   % strike price
S0 = 10;                  % initial stock price

delt = T/N;               % time interval

Large = 1e6;              % penalty coefficient
tolerance = 1/Large;      % toleranceeranceeerance term


Vs = max(K-S.^2, S.^2 - K)';

m = length(S);  %number of grids in row

alpha_central = zeros(m,1);
beta_central = zeros(m,1);
alpha_forward = zeros(m,1);
beta_forward  = zeros(m,1);
alpha_backward = zeros(m,1);
beta_backward = zeros(m,1);
alpha = zeros(m,1);
beta = zeros(m,1);


for i = 2: m - 1
    %%central alpha and beta formula
    alpha_central(i) = alpha_parameter^2*S(i)/((S(i) - S(i-1))*(S(i+1) - S(i-
1)))...
        -r*S(i)/(S(i+1) - S(i-1));
    beta_central(i) = alpha_parameter^2*S(i)/((S(i+1) - S(i))*(S(i+1) - S(i-1)))...
        +r*S(i)/(S(i+1) - S(i-1));

    %%forward alpha and beta formula
    alpha_forward(i) = alpha_parameter^2*S(i)/((S(i) - S(i-1))*(S(i+1) - S(i-1)));
    beta_forward(i) = alpha_parameter^2*S(i)/((S(i+1) - S(i))*(S(i+1) - S(i-1)))...
        + r*S(i)/(S(i+1) - S(i));   % same as beta_central

    %%backward alpha and beta formula
    alpha_backward(i) = alpha_parameter^2*S(i) / ((S(i) - S(i-1))*(S(i+1) - S(i-
1))) ...
        - r * S(i) / (S(i+1) - S(i));
    beta_backward(i) = alpha_parameter^2*S(i)/((S(i+1) - S(i))*(S(i+1) - S(i-1)));

end

%% choosing parameter

for i = 2:m-1
   if(alpha_central(i) >=0 && beta_central(i) >=0)
       alpha(i) = alpha_central(i);
       beta(i) = beta_central(i);
   elseif (alpha_forward(i) >=0 && beta_forward(i) >=0)
       alpha(i) = alpha_forward(i);
       beta(i) = beta_forward(i);
   else
       alpha(i) = alpha_backward(i);
```

```
        beta(i) = beta_backward(i);
    end
end




%% CN-Rannacher time stepping

V_3 = zeros(m, N+1);
V_init_3 = max(K - S.^2, S.^2 - K)';

V_3(:,1) = V_init_3;
V_old_3 = V_init_3;
V_new_3 = V_old_3;
V_new3n = [];

for i = 1:2
    M_matrix = [delt.*-alpha, delt.*(alpha + beta + r), delt.*-beta];
    M = spdiags(M_matrix, [-1,0,1], m-1, m);
    M = full([M;zeros(1,m)]);
    I = eye(m);
    t = 1;
    while t > tolerance
        pv = Large *(V_new_3 < Vs);
        PV = diag(pv);
        rhs1 = V_old_3 + PV * Vs; % RHS of the equation

        AP = sparse(spdiags(ones(m,1),0,m,m)+M+PV);
        [L3,U3,P3,Q3] = lu(AP);
        V_new3n = Q3 * ((L3*U3)\(P3* rhs1));% compute (Vn+1)(k+1)
        t = max(abs(V_new3n - V_new_3)./(max(ones(m,1), abs(V_new3n))));
        V_new_3 = V_new3n;
    end
    V_old_3 = V_new_3;
end


for i = 3:N
    M_matrix = [delt.*-alpha/2, delt.*(alpha + beta + r)/2, delt.*-beta/2];

    M = spdiags(M_matrix, [-1,0,1], m-1, m);
    M = full([M;zeros(1,m)]);
    I = eye(m);
    t =1;
    while t > tolerance
        pv = Large *(V_new_3 < Vs);
        PV = diag(pv);
        rhs2 = (I - M)* V_old_3 + PV*Vs; % RHS of the equation
        AP2 = sparse(spdiags(ones(m,1),0,m,m)+M+PV);

        [L4,U4,P4,Q4] = lu(AP2);
        V_new3n = Q4 * ((L4*U4)\(P4* rhs2));% compute (Vn+1)(k+1)
        t = max(abs(V_new3n - V_new_3)./(max(1, abs(V_new3n))));
         % compute relative change
         V_new_3 = V_new3n;
    end
    V_old_3 = V_new_3;
end
X1 = sprintf('The option value for fully implict in N = %d, S= %d,
is: %s',N,length(S),V_new_3(S == S0));
disp(X1)
result = V_new_3(S == S0);
end
```

The following method is used to get the table for different nodes and timesteps

```
K = 10;
N = 25;
S = [0:0.1*K:0.4*K,...  %input S value
    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];
% for node 62 and timestep 25
L = length(S);
V = const_timestep(S,N);

% for node 123 and timestep 50
N1 = 50;
S1 = movmean(S,2);
S1 = [S,S1];
S1 = sort(S1);
S1 = S1(2:end);
L1 = length(S1);
V1 =  const_timestep(S1,N1);

% for node 245 and timestep 100
N2 = 100;
S2 = movmean(S1,2);
S2 = [S1,S2];
S2 = sort(S2);
S2 = S2(2:end);
L2 = length(S2);
V2 =  const_timestep(S2,N2);

% for node 489 and timestep 200
N3 = 200;
S3 = movmean(S2,2);
S3 = [S2,S3];
S3 = sort(S3);
S3 = S3(2:end);
L3 = length(S3);
V3 =  const_timestep(S3,N3);

% for node 977 and timestep 400
N4 = 400;
S4 = movmean(S3,2);
S4 = [S3,S4];
S4 = sort(S4);
S4 = S4(2:end);
L4 = length(S4);
V4 =  const_timestep(S4,N4);

T_1 =table([N;N1;N2;N3;N4],...
           [L;L1;L2;L3;L4],...
           [V;V1;V2;V3;V4],...
           [NaN;V1-V;V2-V1;V3-V2;V4-V3],...
           [NaN;NaN;(V1-V)/(V2-V1);(V2-V1)/(V3-V2);(V3-V2)/(V4-V3)]);
T_1.Properties.VariableNames ={'Timesteps','Node','Value','Change','Ratio'};
```

Result convergence table is shown below:

Timesteps	Node	Value	Change	Ratio
25	62	98.687	NaN	NaN
50	123	98.685	-0.0021605	NaN
100	245	98.684	-0.00059598	3.6252
200	489	98.684	-0.00013814	4.3144
400	977	98.684	-3.4116e-05	4.049

Table1: The option value in different timesteps and Node implemented by const delt method

From the table, we can see that the convergence of constant timestepping is around 4 and means quadratic convergence although there is a bit over fluctuate

```
figure(1);
plot(S_CN_R,V_CN_R);
xlabel('stock price')
ylabel('option value')
title('option value vs stock price in const timestep CN-R')

figure(2);
S_CN_R_2 = S_CN_R(2:end);
plot(S_CN_R_2,delta)
title('delta vs stock price in const timestep CN-R')
xlabel('stock price')
ylabel('delta')
```

Plot of option value vs stock prices:

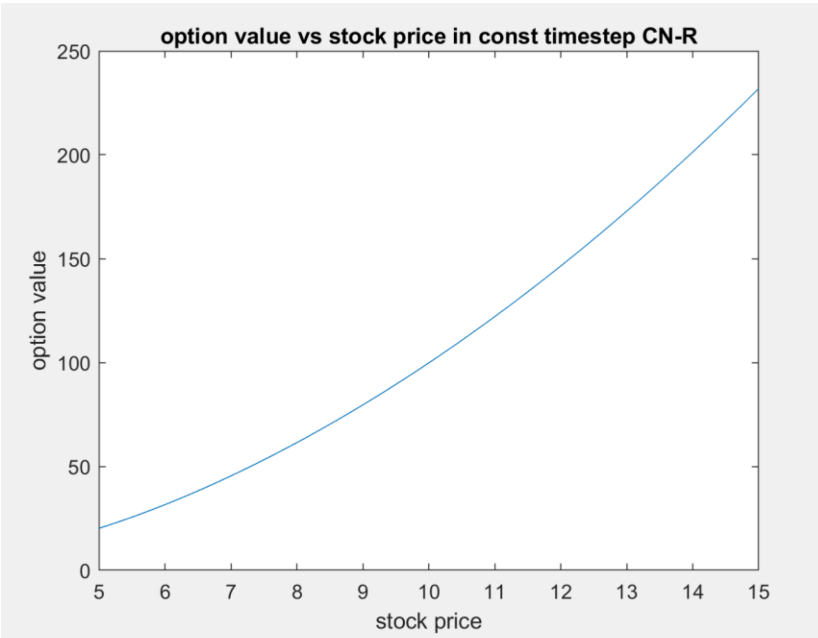

Figure1. The option value vs tock price in const timestep CN-R

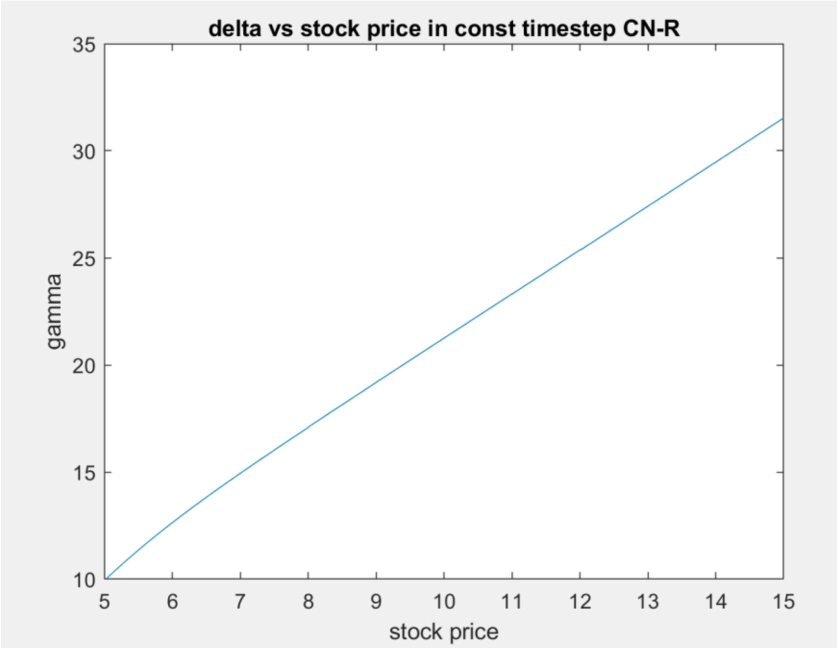

Figure2. The delta vs tock price in const timestep CN-R

From the these two plots, we can see that the option price vs stock price plot is a little bit quadratic while the delta is not exactly linear, actually it is a little bit convex. Furthermore, I don't see any no oscillation.

Delt_selector.m

```
function [result,iteration] = delt_select(S,N)
alpha_parameter =0.8;     % constant related to alpha
r = .02;                  % risk free rate
T = 1;                    % time to expiry
K = 10;                   % strike price
S0 = 10;                  % initial stock price
delt = T/N;               % time interval
Large = 1e6;              % penalty coefficient
tolerance = 1/Large;      % toleranceeranceeerance term
dnorm = 0.1;

Vs = max(K-S.^2, S.^2 - K)';
m = length(S);  %number of grids in row

alpha_central = zeros(m,1);
beta_central = zeros(m,1);
alpha_forward = zeros(m,1);
beta_forward  = zeros(m,1);
alpha_backward = zeros(m,1);
beta_backward = zeros(m,1);
alpha = zeros(m,1);
beta = zeros(m,1);
```

```
for i = 2: m - 1
    %%central alpha and beta formula
    alpha_central(i) = alpha_parameter^2*S(i)/((S(i) - S(i-1))*(S(i+1) - S(i-
1)))...
        -r*S(i)/(S(i+1) - S(i-1));
    beta_central(i) = alpha_parameter^2*S(i)/((S(i+1) - S(i))*(S(i+1) - S(i-1)))...
        +r*S(i)/(S(i+1) - S(i-1));

    %forward alpha and beta formula
    alpha_forward(i) = alpha_parameter^2*S(i)/((S(i) - S(i-1))*(S(i+1) - S(i-1)));
    beta_forward(i) = alpha_parameter^2*S(i)/((S(i+1) - S(i))*(S(i+1) - S(i-1)))...
        + r*S(i)/(S(i+1) - S(i));  % same as beta_central

    %%backward alpha and beta formula
    alpha_backward(i) = alpha_parameter^2*S(i) / ((S(i) - S(i-1))*(S(i+1) - S(i-
1))) ...
        - r * S(i) / (S(i+1) - S(i));
    beta_backward(i) = alpha_parameter^2*S(i)/((S(i+1) - S(i))*(S(i+1) - S(i-1)));
end

%% choosing parameter

for i = 2:m-1
   if(alpha_central(i) >=0 && beta_central(i) >=0)
       alpha(i) = alpha_central(i);
       beta(i) = beta_central(i);
   elseif (alpha_forward(i) >=0 && beta_forward(i) >=0)
       alpha(i) = alpha_forward(i);
       beta(i) = beta_forward(i);
   else
       alpha(i) = alpha_backward(i);
       beta(i) = beta_backward(i);
   end
end


%% time step initialization
delt_sum = delt;
delt_old = delt;

%% CN-Rannacher time stepping

V_3 = zeros(m, N+1);
V_init_3 = max(K - S.^2, S.^2 - K)';
V_3(:,1) = V_init_3;
V_old_3 = V_init_3;
V_new_3 = V_old_3;
V_new3n = [];

%% for the first two implicit methods


for i = 1:2

    M_matrix = [delt_old.*-alpha, delt_old.*(alpha + beta + r), delt_old.*-beta];
    M = spdiags(M_matrix, [-1,0,1], m-1, m);
    M = full([M;zeros(1,m)]);
    I = eye(m);
    t = 1;
    while t > tolerance
        pv = Large *(V_new_3 < Vs);
        PV = diag(pv);
        rhs1 = V_old_3 + PV * Vs; % RHS of the equation
        AP = sparse(spdiags(ones(m,1),0,m,m)+M+PV);
        [L3,U3,P3,Q3] = lu(AP);
        V_new3n = Q3 * ((L3*U3)\(P3* rhs1));% compute (Vn+1)(k+1)
        t = max(abs(V_new3n - V_new_3)./(max(ones(m,1), abs(V_new3n))));
        V_new_3 = V_new3n;
```

```
    end
    MaxRelChange = max(abs(V_new_3 -
V_old_3)./(max(max(1,abs(V_new_3)),abs(V_old_3))));
    delt_new = (dnorm/MaxRelChange)*delt_old;
    delt_sum = delt_sum + delt_new;

    delt_old = delt_new;


    V_old_3 = V_new_3;
end
iteration = 2;

%% for the rest, use CN-R method


while delt_sum < T

    M_matrix = [delt_old.*-alpha/2, delt_old.*(alpha + beta + r)/2, delt_old.*-
beta/2];
    M = spdiags(M_matrix, [-1,0,1], m-1, m);
    M = full([M;zeros(1,m)]);
    I = eye(m);
    t =1;
    while t > tolerance
        pv = Large *(V_new_3 < Vs);
        PV = diag(pv);
        rhs2 = (I - M)* V_old_3 + PV*Vs; % RHS of the equation
        AP2 = sparse(spdiags(ones(m,1),0,m,m)+M+PV);
        %AP2 = sparse(I + 0.5*M + PV);
        [L4,U4,P4,Q4] = lu(AP2);
        V_new3n = Q4 * ((L4*U4)\(P4* rhs2));% compute (Vn+1)(k+1)
        t = max(abs(V_new3n - V_new_3)./(max(1, abs(V_new3n))));
         % compute relative change
         V_new_3 = V_new3n;

    end
    MaxRelChange = max(abs(V_new_3 -
V_old_3)./(max(max(1,abs(V_new_3)),abs(V_old_3))));
    delt_new = (dnorm/MaxRelChange)*delt_old;
    delt_sum = delt_sum + delt_new;
    delt_old = delt_new;
    i = i + 1;
    %sprintf("the value of delt_sum is %d  and the price value is : %d ",delt_sum,
V_new_3(S == S0))

    V_old_3 = V_new_3;
    iteration = iteration + 1;
end

 %% determine the last delt value

delt_old = T-(delt_sum - delt_new);  %% last time step
if delt_old >0

    M_matrix = [delt_old.*-alpha/2, delt_old.*(alpha + beta + r)/2, delt_old.*-
beta/2];
    M = spdiags(M_matrix, [-1,0,1], m-1, m);
    M = full([M;zeros(1,m)]);
    I = eye(m);
    t = 1
    while t > tolerance
  pv = Large *(V_new_3 < Vs);
        PV = diag(pv);
        rhs2 = (I - M)* V_old_3 + PV*Vs; % RHS of the equation
        AP2 = sparse(spdiags(ones(m,1),0,m,m)+M+PV);
        %AP2 = sparse(I + 0.5*M + PV);
        [L4,U4,P4,Q4] = lu(AP2);
```

```
        V_new3n = Q4 * ((L4*U4)\(P4* rhs2));% compute (Vn+1)(k+1)
        t = max(abs(V_new3n - V_new_3)./(max(1, abs(V_new3n))));
         % compute relative change
         V_new_3 = V_new3n;
    end
    V_old_3 = V_new_3;
end
iteration = iteration + 1;
X1 = sprintf('The option value for fully implict in N = %d, S= %d,
is: %d',N,length(S),V_new_3(S == S0));
disp(X1)
result = V_new_3(S == S0);
end
```

The following is used to calculate the table

```
K = 10;
N = 25;
S = [0:0.1*K:0.4*K,...  %input S value
    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];
% for node 62
L = length(S);
[V_S,NS] = delt_select(S,N);

% for node 123
N1 = 50;
S1 = movmean(S,2);
S1 = [S,S1];
S1 = sort(S1);
S1 = S1(2:end);
L1 = length(S1);
[V_S1,NS1 ]=  delt_select(S1,N1);

% for node 245
N2 = 100;
S2 = movmean(S1,2);
S2 = [S1,S2];
S2 = sort(S2);
S2 = S2(2:end);
L2 = length(S2);
[V_S2,NS2] =  delt_select(S2,N2);

% for node 489 and timestep 200
N3 = 200;
S3 = movmean(S2,2);
S3 = [S2,S3];
S3 = sort(S3);
S3 = S3(2:end);
L3 = length(S3);
[V_S3,NS3] =  delt_select(S3,N3);

% for node 977 and timestep 400
N4 = 400;
S4 = movmean(S3,2);
S4 = [S3,S4];
S4 = sort(S4);
S4 = S4(2:end);
L4 = length(S4);
[V_S4,NS4] =  delt_select(S4,N4);

T_S1 =table([NS;NS1;NS2;NS3;NS4],...
```

```
        [L;L1;L2;L3;L4],...
        [V_S;V_S1;V_S2;V_S3;V_S4],...
        [NaN;V_S1-V_S;V_S2-V_S1;V_S3-V_S2;V_S4-V_S3],...
        [NaN;NaN;(V_S1-V_S)/(V_S2-V_S1);(V_S2-V_S1)/(V_S3-V_S2);(V_S3-
V_S2)/(V_S4-V_S3)]);
T_S1.Properties.VariableNames ={'Timesteps','Node','Value','Change','Ratio'};
```

Timesteps	Node	Value	Change	Ratio
19	62	98.687	NaN	NaN
21	123	98.685	-0.002006	NaN
26	245	98.684	-0.00055854	3.5916
30	489	98.684	-0.00014145	3.9485
32	977	98.684	-3.2703e-05	4.3254

Table 2: The option value in different timesteps and Node implemented by penalty method

From the table, we can see that the convergence of selected delt is around 4 and means quadratic convergence. And the option value do converge to 98.684. Compared to the constant timestepping, selected delt implemented by the penalty method is way more efficient, which can be seen from the timesteps. And these two gives almost exactly same result.

```
S_CN_R = S(S>=5 & S <=15)';
V_CN_R = V_new_3(S>=5 & S <=15);
n = length(S_CN_R);

delta = diff(V_CN_R)./diff(S_CN_R);
gamma=(((V_CN_R(3:n) - V_CN_R(2:n-1)) ./ (S_CN_R(3:n) - S_CN_R(2:n-1))) ...
    -((V_CN_R(2:n-1) - V_CN_R(1:n-2)) ./ (S_CN_R(2:n-1) - S_CN_R(1:n-2)))) ...
    ./ ((S_CN_R(3:n) - S_CN_R(1:n-2))/2);

figure(1);
plot(S_CN_R,V_CN_R);
xlabel('stock price')
ylabel('option value')

figure(2);
S_CN_R_2 = S_CN_R(2:end);
plot(S_CN_R_2,delta)
xlabel('stock price')
ylabel('delta')
```

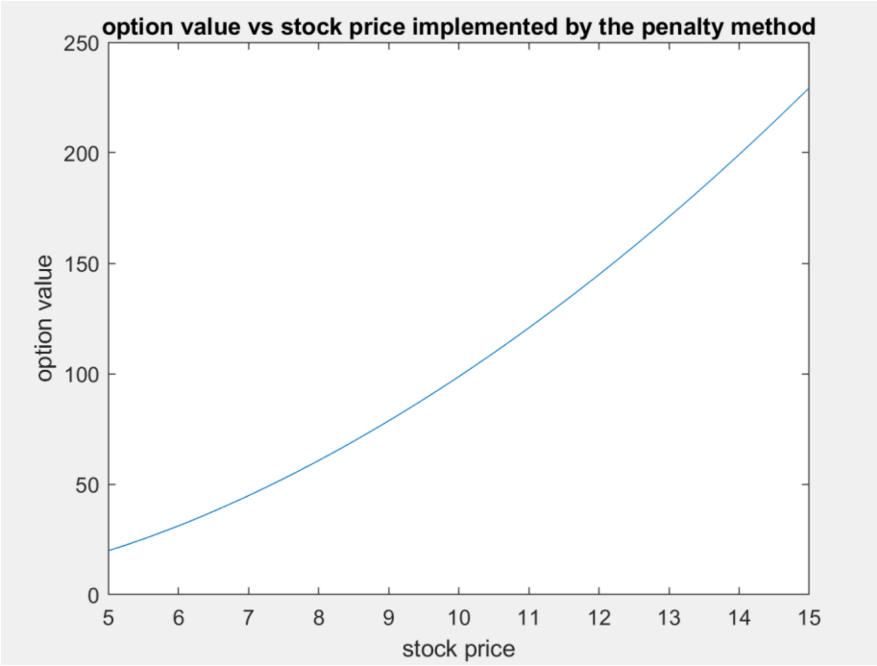

Figure3. The option value vs tock price in penalty method CN-R

Figure4. The delta vs tock price in penalty method CN-R

From the these two plots, we can see that the option price vs stock price plot is a little bit quadratic while the delta is not exactly linear, actually it is a little bit convex. Furthermore, I don't see any no oscillation. Compared to the two plots implemented by the const timestepping, I don't see a hugh difference between each other.