**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; % toleranceeranceerance 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:

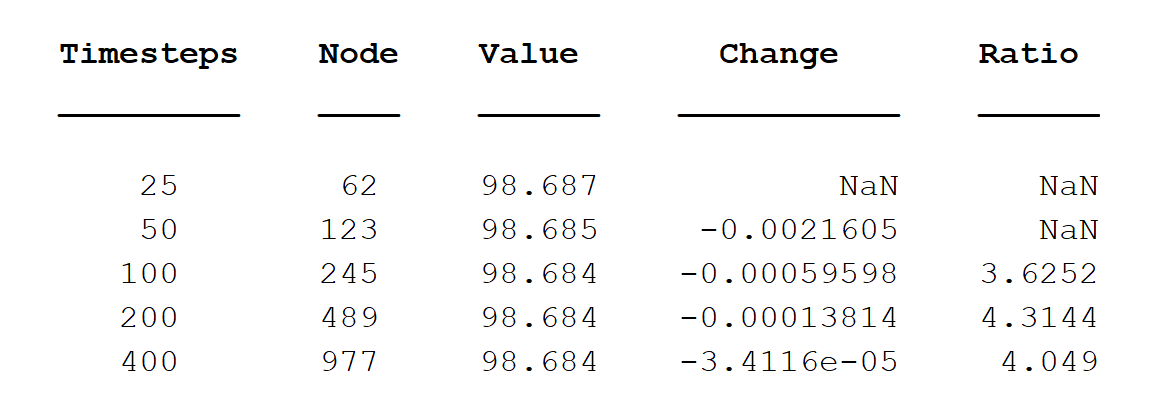


Table1: The option value in different timesteps and Node implemented by const delt method

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:

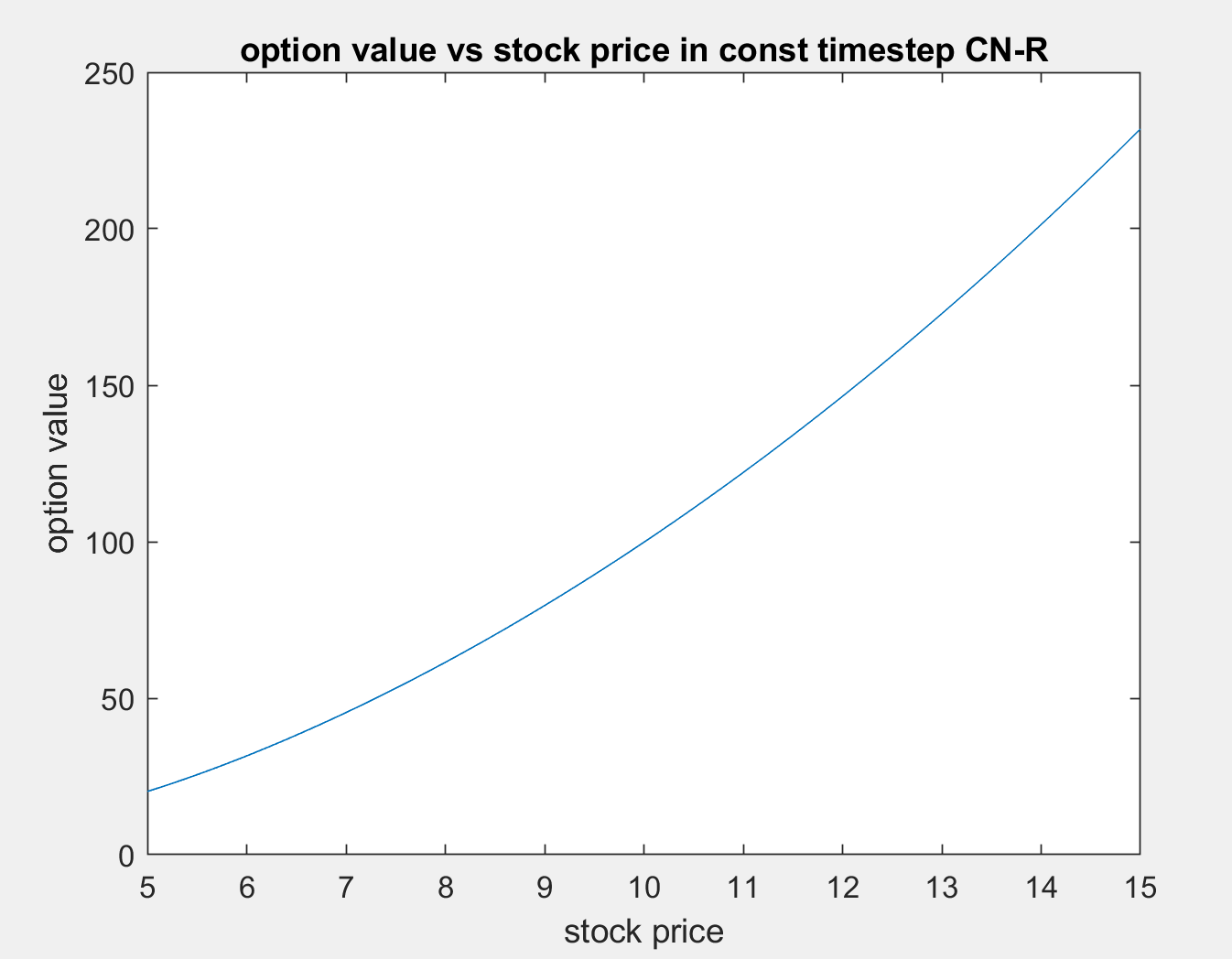


Figure1. The option value vs tock price in const timestep CN-R

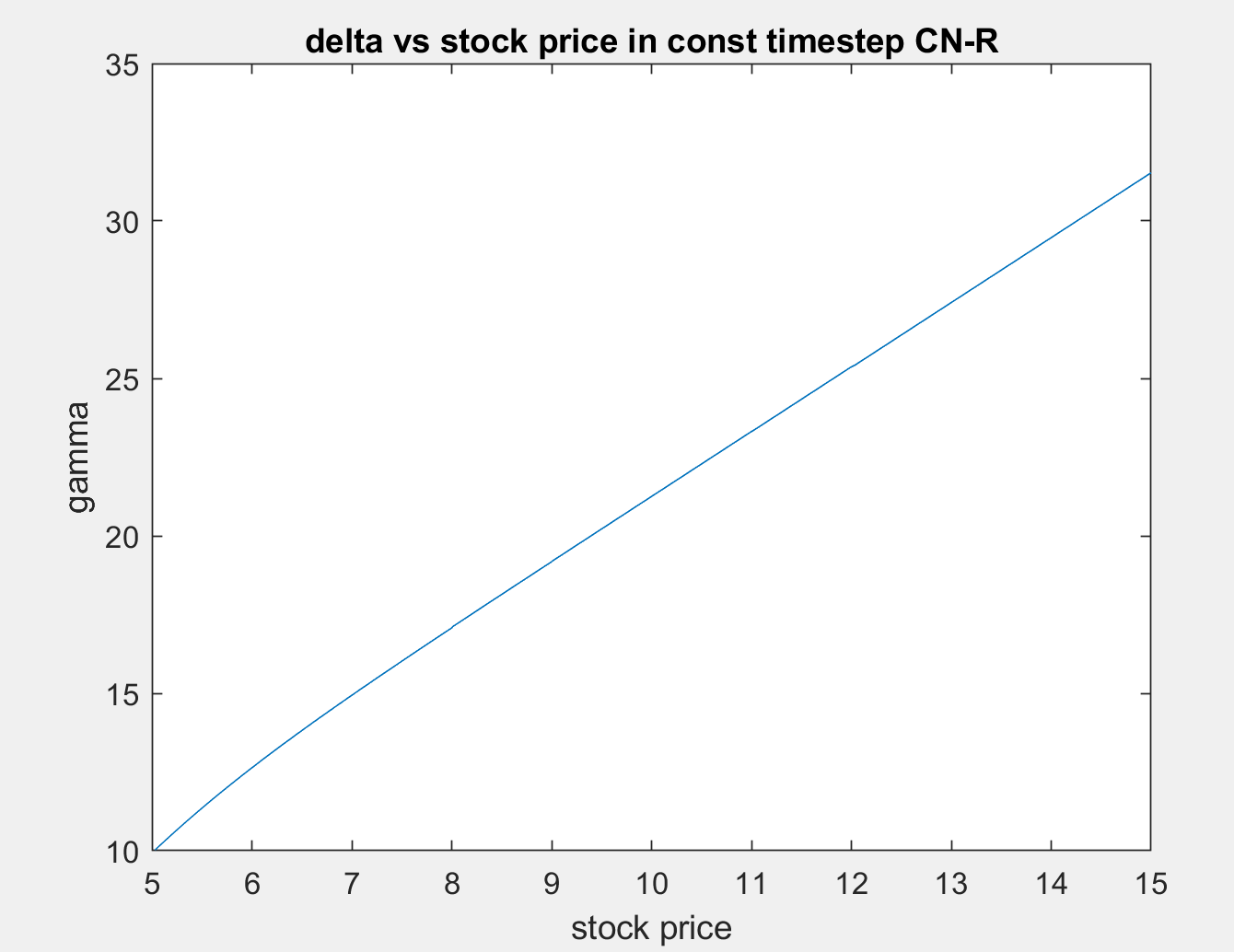


Figure2. The delta vs tock price in const timestep CN-R

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; % toleranceeranceerance 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'};

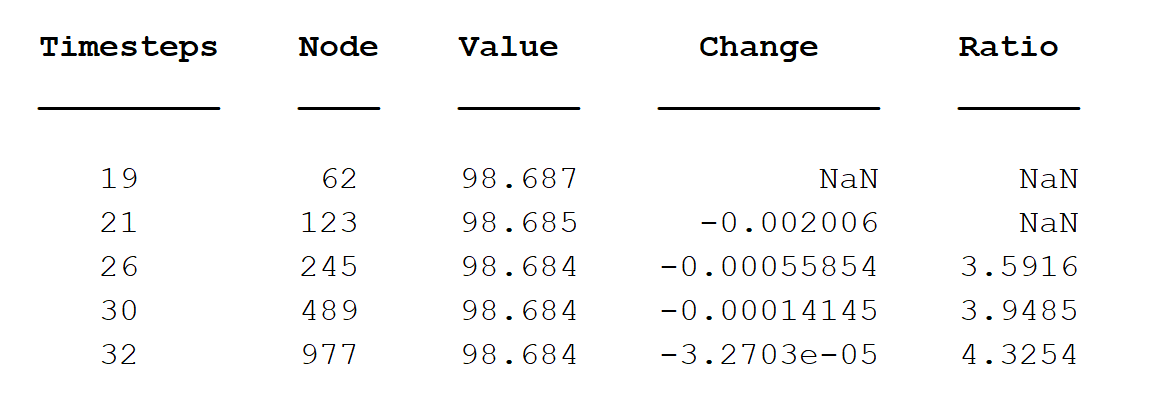


Table 2: The option value in different timesteps and Node implemented by penalty method

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')

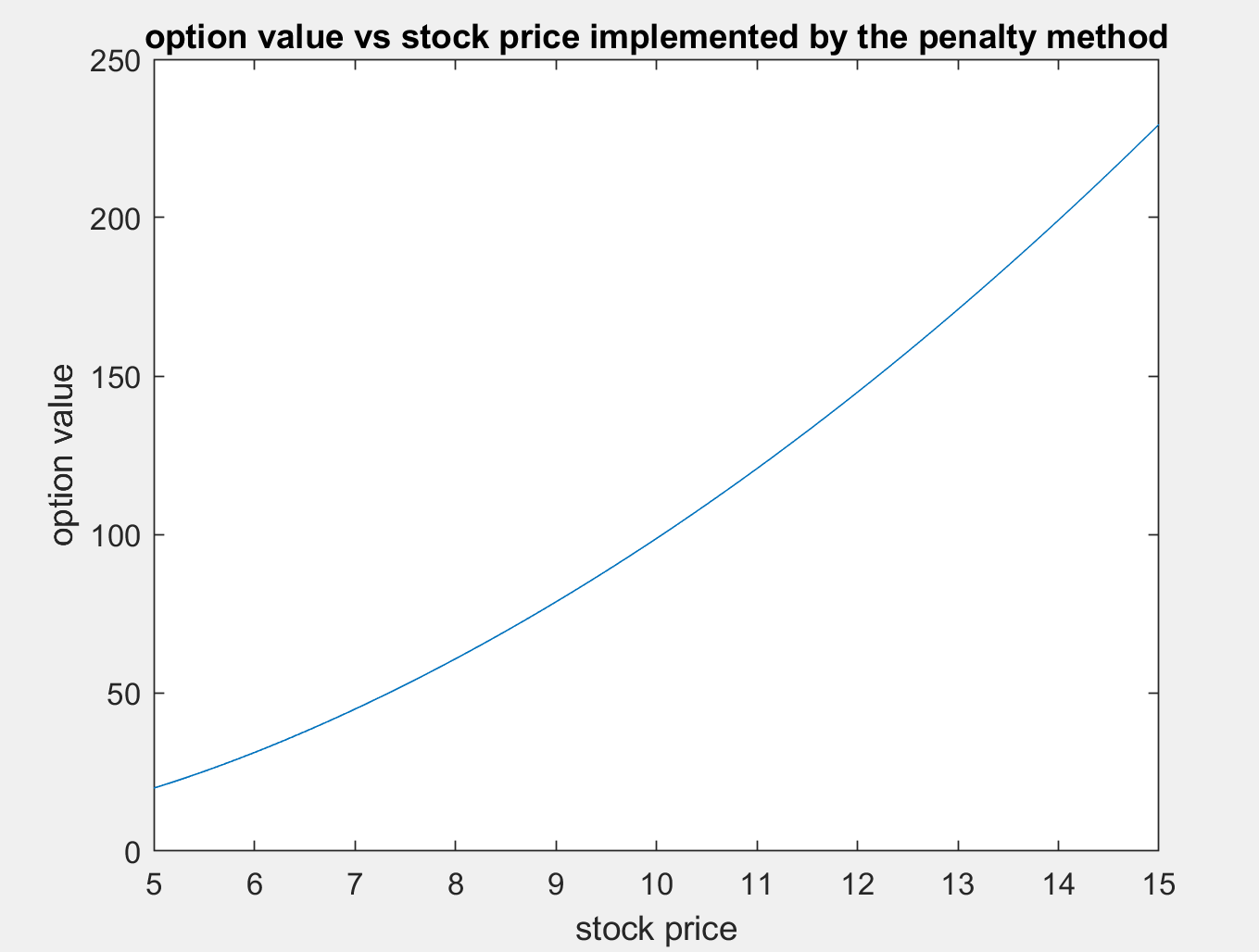


Figure3. The option value vs tock price in penalty method CN-R



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.