CS 676 Assignment 4

Ying Kit Hui (Graduate student)

April 4, 2022

Listings

Precautions: Note that some of the discussions are in the comments of the code. They are usually under a separate section inside the codes. This is done because 1) avoid too much replication and 2) lack of time to edit it and put it in latex format. Please let me know if you strongly prefer them to be presented in latex.

1 Q1

1.1 Q1a

Listing 1: Q1a.m

```
%% Q1a

clearvars
close all
% parameters initialization
S0 = 76;
T = 1;
Deltatau = T/25;
alpha = 15;
r = 0.03;
K = S0;

% check S
S = [ 0:0.1*S0:0.4*S0, ...
    0.45*S0:0.05*S0:0.8*S0, 0.82*S0 :0.02*S0 :0.9 *S0,...
    0.91*S0 :0.01*S0 :1.1*S0, 1.12*S0 :0.02*S0 :1.2*S0, ...
    1.25*S0 :0.05*S0 :1.6*S0, 1.7*S0 :0.1*S0 :2*S0,...
    2.2*S0, 2.4*S0, 2.8*S0, 3.6*S0, 5*S0, 7.5*S0, 10*S0];
S = S';
Scell = {};
Scell{1} = S;

Vold = max(K-S,0).^3;
figure(1)
```

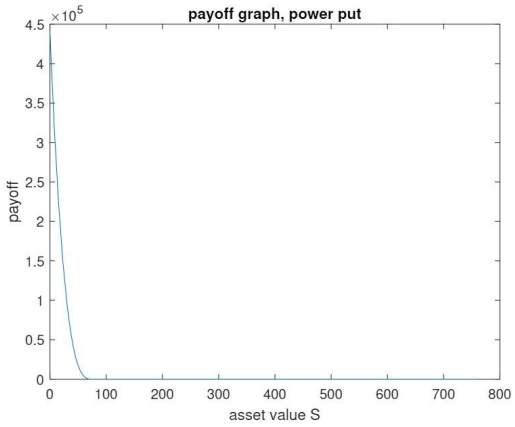
```
plot(S, Vold)
title('payoff graph, power put')
xlabel('asset value S')
ylabel('payoff')

Deltataulist = zeros(5,1);
Deltataulist(1) = Deltatau;
% 4 refinements
for i = 1:4
    Sold = Scell{i};
    Snew = [Sold;(Sold(1:end-1) + Sold(2:end)) / 2 ];
    Snew = sort(Snew);
    Scell{i+1} = Snew;
    Deltataulist(i+1) = Deltatau/(2^i);
end

% make convergence table
Fimplicittable = zeros(length(Deltataulist),5);
CNtable = zeros(length(Deltataulist),5);
CNRantable = zeros(length(Deltataulist),5);

for i = 1:5
    % obtain option value at t=0, S = S0 using the corresponding grid and
    % time step
    [V0FL, V0CN,V0CNRan] = powerputpricer(Scell{i}, Deltataulist(i), S0, T, K
        , r);

    % nodes number
    Fimplicittable(i,1) = size(Scell{i},1);
    CNtable(i,1) = size(Scell{i},1);
    CNRantable(i,1) = size(Scell{i},1);

    % timesteps number
    Fimplicittable(i,2) = 25 * 2^(i-1);
    CNtable(i,2) = 25 * 2^(i-1);
    CNRantable(i,2) = 25 * 2^(i-1);

    % option Value
    Fimplicittable(i,3) = V0FL;
    CNtable(i,3) = V0CN;
    CNRantable(i,3) = V0CNRan;

    % change in option value
    if i > 1
        Fimplicittable(i,4) = Fimplicittable(i,3) - Fimplicittable(i-1,3);
        CNtable(i,4) = CNtable(i,3) - CNtable(i-1,3);
        CNRantable(i,4) = CNRantable(i,3) - CNRantable(i-1,3);
    end

    % Ratio
    if i > 2
        Fimplicittable(i,5) = Fimplicittable(i-1,4)/Fimplicittable(i,4);
        CNtable(i,5) = CNtable(i-1,4)/CNtable(i,4);
        CNRantable(i,5) = CNRantable(i-1,4)/CNRantable(i,4);
    end
end

Fimplicittable = array2table(Fimplicittable, 'VariableNames', ...
    {'Nodes','timesteps','Value','Change','Ratio'})
CNtable = array2table(CNtable, 'VariableNames', ...
    {'Nodes','timesteps','Value','Change','Ratio'})
CNRantable = array2table(CNRantable, 'VariableNames', ...
    {'Nodes','timesteps','Value','Change','Ratio'})

%% Q1a Discussion
```

```
% Note that fully implicit method has convergence rate O(\Delta tau, (\Delta
    S)^2)
% and hence it does not satisfy (4) in our question. Thus we don't expect
% the ratio for the convergence table of fully implicit method to be 4. In
% fact, we observe that such ratio is around 3, reflecting that the
% convergence rate is slightly better than first order (since we have
% O((\Delta S)^2)) but not as good as second order (since we have O(\Delta
% tau)).
%
% Note that Crank-Nicolson method has convergence rate  O((\Delta tau)^2,
% (\Delta S)^2). Hence, we expect the ratio for the convergence table of CN
% method to be 4. We observe that such ratio is indeed roughly 4,
% consistent with the theory regarding the rate of convergence.
% Note that generally non-smooth payoff will cause oscillations in
% solution, resulting in slow convergence of CN method. However, in our
% case, the payoff function of the power put is quite smooth since we are
% taking cubic power. The smoothness of the payoff can also be seen in
% figure 1. At least, the payoff is continuous and the fact that K = S0 and
% we have a node at S0 = K means that no smoothing is required and we
% expect the usual second order convergence.
%
% For CN-Rannacher, the purpose of it is to do some smoothing so that
% second order convergence is attained even for nonsmooth payoff. But in
% our case, the payoff is quite smooth. But it does not prohibit
% CN-Rannacher to achieve second order convergence, as demostrated in the
% Ratio column of its convergence table being around 4.

function [V0fullyimplicit, V0CN,V0CNRan] = powerputpricer(S, Deltatau, S0, T,
     K, r)
% output option value at t =0, S =S0, using fully implicit, Crank Nicolson,
% and CN-Rannacher

%%% Fully implicit
%%% only compute initial option value
%%% can easily modified to obtain options values at each grid points
m = length(S);
Vold = zeros(m,1);
Vnew = Vold;
N = T / Deltatau;
% initialize payoff
Vold = max(K-S,0).^3;

% preprocessing step for simple BS equation
% so that we have positive coefficient discretization
% these will also be used for CN and CN-Rannacher
Smid = S(2:end-1);
Sleft = S(1:end-2);
Sright = S(3:end);
sigma = sigmalvf(Smid);
alphacentral = sigma.^2 .* Smid .^2 ./ ((Smid - Sleft).* (Sright - Sleft)) -
    ...
    r* Smid ./(Sright - Sleft);
betacentral = sigma.^2 .* Smid .^2 ./ ((Sright - Smid).* (Sright - Sleft)) +
    ...
    r* Smid ./(Sright - Sleft);

alphaups = sigma.^2 .* Smid .^2 ./ ((Smid - Sleft).* (Sright - Sleft));
betaups = sigma.^2 .* Smid .^2 ./ ((Sright - Smid).* (Sright - Sleft)) + ...
    r* Smid ./(Sright - Smid);

idx1 = alphacentral <0;
idx2 = betacentral <0;
idx = idx1 | idx2;

alphavec = alphacentral;
betavec = betacentral;
```

```
alphavec(idx) = alphaups(idx);
betavec(idx) = betaups(idx);

% build the matrix M by spdiags
Bin = Deltatau* [[[-alphavec;0];0], [[r; (alphavec + betavec +r)];0],[[0;0];-
    betavec]];
M = spdiags(Bin, -1:1, m, m);
% spy(M) % see the nonzero entries of the matrix M
% Note that our M is time independent
I = speye(m);
[L,U,P] = lu(M+I);
% increase performance since the LU factorization is only computed once and
% being reused in each linear system solving
% to solve (M+I)x = b, y =L\ (P*b), x = U\y

for n = 1 : N
    y = L\(P*Vold);
    Vnew = U\y;
    Vold = Vnew;
end
S0id = S0 == S;
V0fullyimplicit = Vnew(S0id);


%%% Crank-Nicolson
%%% only compute initial option value
%%% can easily modified to obtain options values at each grid points
Vold = zeros(m,1);
Vnew = Vold;
% initialize payoff
Vold = max(K-S,0).^3;

Bin = Deltatau/2* [[[-alphavec;0];0], [[r; (alphavec + betavec +r)
    ];0],[[0;0];-betavec]];
Mhat = spdiags(Bin, -1:1, m, m);
% Note that our Mhat is time independent
I = speye(m);
[L1,U1,P1] = lu(Mhat+I);


for n = 1 : N
    y = L1\(P1*(I-Mhat)*Vold);
    Vnew = U1\y;
    Vold = Vnew;
end
V0CN = Vnew(S0id);

%%% CN-Rannacher
Vold = zeros(m,1);
Vnew = Vold;
% initialize payoff
Vold = max(K-S,0).^3;

% two steps of fully implicit
for n = 1 : 2
    y = L\(P*Vold);
    Vnew = U\y;
    Vold = Vnew;
end

% use CN after that
for n = 1 : N -2
    y = L1\(P1*(I-Mhat)*Vold);
    Vnew = U1\y;
    Vold = Vnew;
end
```

```
V0CNRan = Vnew(S0id);

end

function out = sigmalvf(S)
% note that the local volatility function actually is independent of t
% vectorized output, assume column vector input
    alpha = 15;
    S0 = 76;
    out = alpha./ (max(S0 + (S- S0)./ (2*S0), 10^(-8)));
end
```

1.1.1 result



```
Fimplicittable =

  5×5 table

    Nodes    timesteps    Value      Change      Ratio
    _____    _________    ______    ________    ______

      62         25        1422.7           0         0
     123         50        1418.6      -4.1154        0
     245        100        1417.3      -1.2681    3.2454
     489        200        1416.9     -0.42717    2.9685
     977        400        1416.7     -0.16109    2.6518


CNtable =

  5×5 table

    Nodes    timesteps    Value      Change      Ratio
    _____    _________    ______    ________    ______

      62         25          1421           0         0
     123         50        1417.7      -3.2983        0
     245        100        1416.9     -0.84009    3.9262
     489        200        1416.7     -0.21094    3.9827
     977        400        1416.6     -0.05279    3.9958


CNRantable =

  5×5 table

    Nodes    timesteps    Value      Change       Ratio
    _____    _________    ______    _________    ______

      62         25        1421.2           0         0
     123         50        1417.8        -3.386       0
     245        100        1416.9      -0.86402    3.9189
     489        200        1416.7      -0.21719    3.9782
     977        400        1416.6     -0.054384    3.9936
```

1.2 Q1b

Listing 2: Q1b.m

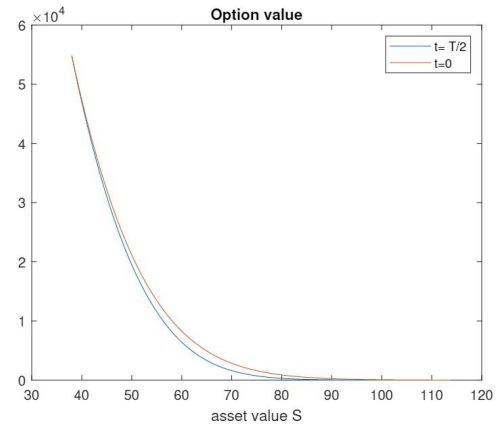
```
%% Q1b
close all
% parameters initialization
S0 = 76;
T = 1;
Deltatau = T/25;
alpha = 15;
r = 0.03;
K = S0;

% check S
S = [ 0:0.1*S0:0.4*S0, ...
    0.45*S0:0.05*S0:0.8*S0, 0.82*S0 :0.02*S0 :0.9 *S0,...
    0.91*S0 :0.01*S0 :1.1*S0, 1.12*S0 :0.02*S0 :1.2*S0, ...
    1.25*S0 :0.05*S0 :1.6*S0, 1.7*S0 :0.1*S0 :2*S0,...
    2.2*S0, 2.4*S0, 2.8*S0, 3.6*S0, 5*S0, 7.5*S0, 10*S0];
S = S';
Scell = {};
Scell{1} = S;

Deltataulist = zeros(5,1);
Deltataulist(1) = Deltatau;
% 4 refinements
for i = 1:4
    Sold = Scell{i};
    Snew = [Sold;(Sold(1:end-1) + Sold(2:end)) / 2 ];
    Snew = sort(Snew);
```

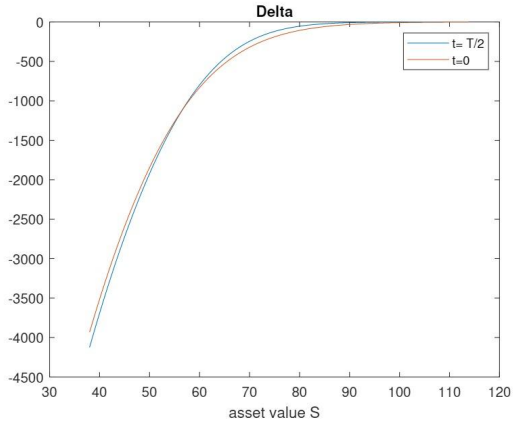
```
    Scell{i+1} = Snew;
    Deltataulist(i+1) = Deltatau/(2^i);
end

% USE the finest grid for CN-Rannacher
S = Scell{end};
Deltatau = Deltataulist(end);


% preprocessing step for simple BS equation
% so that we have positive coefficient discretization
% these will also be used for CN and CN-Rannacher
m = length(S);
N = T / Deltatau;

Smid = S(2:end-1);
Sleft = S(1:end-2);
Sright = S(3:end);
sigma = sigmalvf(Smid);
alphacentral = sigma.^2 .* Smid .^2 ./ ((Smid - Sleft).* (Sright - Sleft)) - ...
    r* Smid ./(Sright - Sleft);
betacentral = sigma.^2 .* Smid .^2 ./ ((Sright - Smid).* (Sright - Sleft)) + ...
    r* Smid ./(Sright - Sleft);

alphaups = sigma.^2 .* Smid .^2 ./ ((Smid - Sleft).* (Sright - Sleft));
betaups = sigma.^2 .* Smid .^2 ./ ((Sright - Smid).* (Sright - Sleft)) + ...
    r* Smid ./(Sright - Smid);

idx1 = alphacentral <0;
idx2 = betacentral <0;
idx = idx1 | idx2;

alphavec = alphacentral;
betavec = betacentral;
alphavec(idx) = alphaups(idx);
betavec(idx) = betaups(idx);

% build the matrix M by spdiags
Bin = Deltatau* [[[-alphavec;0];0], [[r; (alphavec + betavec +r)];0],[[0;0];-betavec
    ]];
M = spdiags(Bin, -1:1, m, m);
% spy(M) % see the nonzero entries of the matrix M
% Note that our M is time independent
I = speye(m);
[L,U,P] = lu(M+I);
% increase performance since the LU factorization is only computed once and
% being reused in each linear system solving
% to solve (M+I)x = b, y =L\ (P*b), x = U\y

Bin = Deltatau/2* [[[-alphavec;0];0], [[r; (alphavec + betavec +r)];0],[[0;0];-betavec
    ]];
Mhat = spdiags(Bin, -1:1, m, m);
% Note that our Mhat is time independent
I = speye(m);
[L1,U1,P1] = lu(Mhat+I);

%%% CN-Rannacher
Vold = zeros(m,1);
Vnew = Vold;
% initialize payoff
Vold = max(K-S,0).^3;

% two steps of fully implicit
for n = 1 : 2
    y = L\(P*Vold);
    Vnew = U\y;
    Vold = Vnew;
end

% use CN after that
Vbeforehalf = zeros(m,1);
Vafterhalf = zeros(m,1);
```

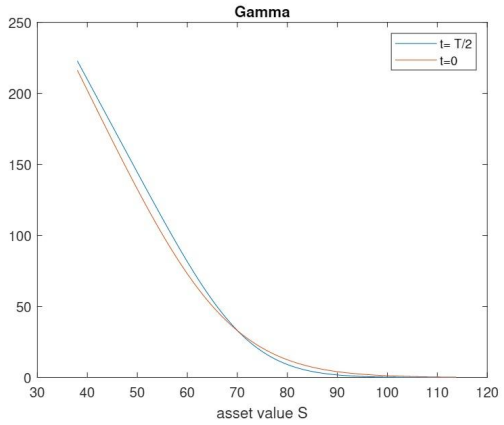
```
Vathalf = zeros(m,1);
Vbefore0 = zeros(m,1);
for n = 1 : N-2
    y = L1\(P1*(I-Mhat)*Vold);
    Vnew = U1\y;
    Vold = Vnew;
    if n == N/2 -3   % save the data at the timestep just after t= T/2,
                     % after in terms of t
        Vbeforehalf = Vold;
    end
    if n == N/2 -1   % save the data at the timestep just before t = T/2,
                     % before in terms of t
        Vafterhalf = Vold;
    end
    if n == N/2- 2   % save the data at the t= T/2
        Vathalf = Vold;
    end
    if n == N-2 -1   % save the data at the timestep just before t = 0
        Vbefore0 = Vold;
    end

end

% index for S in the range = [0.5 S0, 1.5 S0]
idx = (S <= 1.5*S0) & (S >= 0.5 * S0);
idx = find(idx);
index1 = idx(1);
index2 = idx(end);

Vat0 = Vold; % the data at t = 0

% Central difference for delta and gamma at t = T/2
deltaxright = S(index1 + 1: index2 + 1) - S(index1:index2);
deltaxleft = S(index1:index2) - S(index1 - 1:index2-1);
deltafright = Vathalf(index1 + 1: index2 + 1) - Vathalf(index1:index2);
deltafleft = Vathalf(index1:index2) - Vathalf(index1 - 1:index2-1);
gammaathalf = (deltafright ./ deltaxright - deltafleft./deltaxleft) ...
    ./ ((deltaxright + deltaxleft)/2);
% not sure about this, I think this might improve the order of delta a bit
deltaathalf = (Vathalf(index1+1: index2+1)-Vathalf(index1-1:index2-1))...
    ./(deltaxright + deltaxleft) - gammaathalf /2 .*(deltaxright-deltaxleft);

% Central difference for delta and gamma at t = 0

deltafright0 = Vat0(index1 + 1: index2 + 1) - Vat0(index1:index2);
deltafleft0 = Vat0(index1:index2) - Vat0(index1 - 1:index2-1);
gammaat0 = (deltafright0 ./ deltaxright - deltafleft0./deltaxleft) ...
    ./ ((deltaxright + deltaxleft)/2);
% not sure about this, I think this might improve the order of delta a bit
deltaat0 = (Vat0(index1+1: index2+1)-Vat0(index1-1:index2-1))...
    ./(deltaxright + deltaxleft) - gammaat0 /2 .*(deltaxright-deltaxleft);

figure(1)
plot(S(idx), Vathalf(idx))
hold on
plot(S(idx),Vat0(idx))
title('Option value')
xlabel('asset value S')
legend('t= T/2','t=0')
hold off

figure(2)
plot(S(idx), deltaathalf)
hold on
plot(S(idx), deltaat0)
title('Delta')
xlabel('asset value S')
legend('t= T/2','t=0')
hold off

figure(3)
plot(S(idx), gammaathalf)
```

```
hold on
plot(S(idx), gammaat0)
title('Gamma')
xlabel('asset value S')
legend('t= T/2','t=0')
hold off

save Q1b.mat

%% Discussion
% For figure(1) option value plot, we observe that the option values at t =
% 0 are larger than that at t = T/2, for the same asset value S. This makes
% sense since the Greek Theta is generally negative for long call or long
% put, reflecting the fact that there are time values in options and as
% time goes, this time value decays. Note particularly that the gap between
% two curves is the most significant (relatively speaking) when S is near
% the strike, this is because at the far left or the far right, the time
% value of the option is almost negligible. On th far right, the value of
% the option is almost 0, and we expect 0 payoff. On the far left, it might
% best to exercise right now. (not sure)
%
% For figure(2) delta plot, we see a more complicated relationship. First
% of all, the delta are all nonpositive. This is because as S increases,
% our expected payoff will decrease, since we are holding power put. Note
% that two curves intersect, let's denote the intersection by x* (around
% 56). Also, we consider absolute value of delta in the following
% comparison of size. So we say size of Delta 1 is larger than that of
% Delta 2, we actually mean that the absolute value of Delta 1 is larger
% than absolute value of Delta 2. Before x*, the size delta for t = 0 is
% less than that of t = T/2. This makes sense since at T/2, any increase in
% asset value at region that already has low asset value (i.e. high payoff)
% will very likely result in a decrease of final payoff, whereas at t =0,
% that phenomenon is less severe. (not sure about this interpretation)
%
% For figure(3) gamma plot, we see a more complicated relationship. First
% of all, the gamma are all nonpositive, indicating that the option value
% is convex and delta is increasing function of S.  Note
% that two curves intersect, let's denote the intersection by x** (around
% 70). Before x**, option value for t = T/2 is more convex. After x**,
% option value for t =0 is more convex.
%
% I am not sure how can I intuitively or financially interpret these plots.

function out = sigmalvf(S)
% note that the local volatility function actually is independent of t
% vectorized output, assume column vector input
    alpha = 15;
    S0 = 76;
    out = alpha./ (max(S0 + (S- S0)./ (2*S0), 10^(-8)));
end
```

1.2.1 result


Option value
×10^4
6
5
4
3
2
1
0
30
40
50
60
70
80
90
100
110
120
asset value S
t= T/2
t=0


Delta
0
-500
-1000
-1500
-2000
-2500
-3000
-3500
-4000
-4500
30
40
50
60
70
80
90
100
110
120
asset value S
t= T/2
t=0


Gamma
t= T/2
t=0
250
200
150
100
50
0
30
40
50
60
70
80
90
100
110
120
asset value S

2 Q2

2.1 Q2a

Listing 3: Q2a.m

```
%% Q2a, variable timestepping and constant timestepping

clearvars
close all
%%  parameters initialization
S0 = 76;
T = 1;
Deltatau = T/25; % initial time step
dnorm = 0.1;
alpha = 15;
r = 0.03;
K = S0;

S = [ 0:0.1*S0:0.4*S0, ...
    0.45*S0:0.05*S0:0.8*S0, 0.82*S0 :0.02*S0 :0.9 *S0,...
    0.91*S0 :0.01*S0 :1.1*S0, 1.12*S0 :0.02*S0 :1.2*S0, ...
    1.25*S0 :0.05*S0 :1.6*S0, 1.7*S0 :0.1*S0 :2*S0,...
    2.2*S0, 2.4*S0, 2.8*S0, 3.6*S0, 5*S0, 7.5*S0, 10*S0];
S = S';

%% Now do refinements
Scell = {};
Scell{1} = S;
% list of initial timestep
Deltataulist = zeros(5,1);
Deltataulist(1) = Deltatau;
dnormlist = zeros(5,1);
dnormlist(1) = dnorm;
% 4 refinements
for i = 1:4
    Sold = Scell{i};
    Snew = [Sold;(Sold(1:end-1) + Sold(2:end)) / 2 ];
    Snew = sort(Snew);
    Scell{i+1} = Snew;
    Deltataulist(i+1) = Deltatau/(4^i);
    dnormlist(i+1) = dnorm/(2^i);
end

% make convergence table
CNRvartable = zeros(length(Deltataulist),7);
CNRconstanttable = zeros(length(Deltataulist),7);
vartimetotal =0;
contimetotal =0;
for i = 1:5
    % obtain option value at t=0, S = S0 using the corresponding grid and
    % time step
    Deltatau = Deltataulist(i);
    S = Scell{i};
    dnorm = dnormlist(i);


    vartime = tic;
    [V0var, steptotalvar, countervar] = CNR(S, Deltatau, S0, T, K, r,dnorm,1)
        ;
    vartime = toc(vartime);
    vartimetotal = vartimetotal + vartime;

    contime = tic;
    [V0con, steptotalcon, countercon] = CNR(S, Deltatau, S0, T, K, r,dnorm,0)
        ;
    contime = toc(contime);
```

```
    contimetotal = contimetotal + contime;

    % nodes number
    CNRvartable(i,1) = size(Scell{i},1);
    CNRconstanttable(i,1) = size(Scell{i},1);

    % timesteps number
    CNRvartable(i,2) = T/ Deltatau;
    CNRconstanttable(i,2) = T/ Deltatau;

    % option Value
    CNRvartable(i,3) = V0var;
    CNRconstanttable(i,3) = V0con;
    % change in option value
    if i > 1
        CNRvartable(i,4) = CNRvartable(i,3) - CNRvartable(i-1,3);
        CNRconstanttable(i,4) = CNRconstanttable(i,3) - CNRconstanttable(i
            -1,3);
    end

    % Ratio
    if i > 2
        CNRvartable(i,5) = CNRvartable(i-1,4)/CNRvartable(i,4);
        CNRconstanttable(i,5) = CNRconstanttable(i-1,4)/CNRconstanttable(i,4)
            ;
    end

    % counter
    CNRvartable(i,6) = countervar;
    CNRconstanttable(i,6) = countercon;

    % total iterative steps
    CNRvartable(i,7) = steptotalvar;
    CNRconstanttable(i,7) = steptotalcon;

end

CNRvartable = array2table(CNRvartable, 'VariableNames', ...
    {'Nodes','initial timestep','Value','Change','Ratio','# timestepping','
        iterative steps' })
vartimetotal

CNRconstanttable = array2table(CNRconstanttable, 'VariableNames', ...
    {'Nodes','timesteps','Value','Change','Ratio','# timestepping','iterative
        steps' })
contimetotal

%% Discussion
% For both methods, we refine the grids as in Q1, doubling the number of
% intervals.
%
% For CN-Rannacher with variable timestepping, at each refinement, we
% reduce the initial timestep by a factor of 4, and reduce dnorm by a
% factor of 2.
%
% For CN-Rannacher with constant timesteppin, at each refinement, we reduce
% the constant timestepping by a factor of 4.
%
% We see that CN-Rannacher with variable timestepping does give us
% quardartic convergence, since the ratio in CNRvartable is around 4.
% This is expected.
%
% Note that for the convergence table of CN-Rannacher with constant
% timestepping, at each refinement we are dividing the constant timestep by
% 4. This might explain why we also observe quardartic convergence in such
% case. I suspect that the (convergence) error might only depend slightly
```

```
% on \Delta S, so that the number of nodes does not greatly affect the
% convergence rate. If this is true, this explains why we obtain quardartic
% convergence. Another reason for that might be the powerput payoff is
% quite smooth so that quardartic convergence can still be achieved.
%
% Note that if we change the refinement to timestep /2, still observe the
% quardartic convergence... (WHY???) Perhaps it is because the powerput
% payoff is quite smooth. I have experimented my codes on other examples in
% the lecture slides and obtain similar results.
%
% Lastly, in the setting we consider here, the runtime of CNR-variable
% timestepping is less than that of CNR-constant timestepping.

function [V0, steptotal, counter] = CNR(S, Deltatau, S0, T, K, r,dnorm,opt)
%% Description
% Inputs:
% opt = 1 for variable time, else for constant timestepping
% when opt = 1, Deltatau is the initial timestep
% when opt is else, Deltatau is the constant timestep
% Outputs:
% V0, option value at S0 at t = 0
% steptotal, the total number of steps used for iterative solvers
% counter, the total time of choosing variable timestep; counter is just
% T/Deltatau when constant timestepping

%%  preprocessing step for simple BS equation
% so that we have positive coefficient discretization
% these will also be used for CN and CN-Rannacher
Smid = S(2:end-1);
Sleft = S(1:end-2);
Sright = S(3:end);
sigma = sigmalvf(Smid);
alphacentral = sigma.^2 .* Smid .^2 ./ ((Smid - Sleft).* (Sright - Sleft)) -...
    r* Smid ./(Sright - Sleft);
betacentral = sigma.^2 .* Smid .^2 ./ ((Sright - Smid).* (Sright - Sleft)) +...
    r* Smid ./(Sright - Sleft);

alphaups = sigma.^2 .* Smid .^2 ./ ((Smid - Sleft).* (Sright - Sleft));
betaups = sigma.^2 .* Smid .^2 ./ ((Sright - Smid).* (Sright - Sleft)) + ...
    r* Smid ./(Sright - Smid);

idx1 = alphacentral <0;
idx2 = betacentral <0;
idx = idx1 | idx2;

alphavec = alphacentral;
betavec = betacentral;
alphavec(idx) = alphaups(idx);
betavec(idx) = betaups(idx);
%% Some common data needed for constant and variable timestepping
m = length(S);
Vold = zeros(m,1);
Vnew = Vold;
N = T / Deltatau;
% initialize payoff
Vold = max(K-S,0).^3;

% penalty method data, tolerance and Large (penalty coefficient)
tol = 10^(-6); % as suggested by piazza
Large = 1/tol;

% Vstar, payoff at the gridpoints, for American option payoff constraints
Vstar = max(K-S,0).^3;

```

```
Bin = [[[-alphavec;0];0], [[r; (alphavec + betavec +r)];0],[[0;0];-betavec]];
% generic bin, used repeatedly below

steptotal = 0;

%% opt = 1, Use variable timestep
%%% also obtain number of steps to reach tol
%%% also need to update the Mhat and M
if opt == 1
    dtold = Deltatau;
    dtnew = 0;
    currenttau = 0;
    D = 1; % p.32 in lec 18, D = 1 fine for dollars
    counter = 0;
    % CN-Rannacher, variable time
    while currenttau < T
        counter = counter +1;

        if currenttau + dtold >= T
            dtold = T - currenttau;
            currenttau = T;
        else
            currenttau = currenttau + dtold;
        end

        if counter <= 2 % first two steps are fully implicit
            % build M, using correct \Delta tau
            M = spdiags(dtold*Bin, -1:1, m, m);
            [Vnew,step]= iterativesolver(Vold, Vstar, tol, Large, M,m,0);
        else
            % build Mhat, using correct \Delta tau
            Mhat = spdiags((dtold/2) *Bin, -1:1, m, m);
            [Vnew,step]= iterativesolver(Vold, Vstar, tol, Large, Mhat,m,1);
        end

        steptotal = steptotal + step;

        % chooose timestep at the end
        % assume Vold is still storing previous data, not current data
        inter = max(D,max(abs(Vnew), abs(Vold)));
        maxrelchange = max(abs(Vnew - Vold)./inter);
        dtnew = (dnorm/maxrelchange) * dtold;
        dtold = dtnew;
        % update V
        Vold = Vnew;
    end

    S0id = S0 == S;
    V0 = Vnew(S0id);
else
    %% opt is not 1, use constant timestepping
    Mhat = spdiags((Deltatau/2)*Bin, -1:1, m, m);
    M = spdiags(Deltatau*Bin, -1:1, m, m);
    % two steps of Fully Implicit
    for i = 1:2
        [Vnew,step]= iterativesolver(Vold, Vstar, tol, Large, M,m,0);
        steptotal = steptotal + step;
        Vold = Vnew;
    end

    for i = 1: N-2
        [Vnew,step]= iterativesolver(Vold, Vstar, tol, Large, Mhat,m,1);
        steptotal = steptotal + step;
        Vold = Vnew;
    end
    S0id = S0 == S;
```

```
    V0 = Vnew(S0id);
    counter = N;
end

end

function [Vnew,step]= iterativesolver(Vold, Vstar, tol, Large, genM,m,opt)
% opt = 1 for CN, else fully implicit
% genM= Mhat when opt = 1 (i.e. CN), o/w genM = M (i.e. fully implicit)

error = 10000; % set the error to be some large number
step = 0;
Viterold = Vold;
I = speye(m);
while error >= tol
    % Vold is V^n
    % Viterold is (V^{n+1})^k, Viternew will be (V^{n+1})^(k+1)
    largediagonal = zeros(m,1);
    largeidx = Viterold < Vstar;
    largediagonal(largeidx) = Large;
    P = spdiags(largediagonal, 0,m,m);
    if opt == 1 % CN
        y = (I - genM)*Vold + P* Vstar;
        Viternew = (I + genM + P) \ y;
    else % fully implicit
        y = Vold + P* Vstar;
        Viternew = (I + genM + P) \ y;
    end

    inter = abs(Viternew - Viterold)./max(1,abs(Viternew));
    error = max(inter);
    step = step + 1;
    if error < tol
        Vnew = Viternew;
        break;
    end
    Viterold = Viternew;
end

% why
% largediagonal = zeros(m,1);
% largeidx = Vnew < Vstar;
% largediagonal(largeidx) = Large;
% P = spdiags(largediagonal, 0,m,m);
% sum(P,'all')
end

function out = sigmalvf(S)
% note that the local volatility function actually is independent of t
% vectorized output, assume column vector input
    alpha = 15;
    S0 = 76;
    out = alpha./ (max(S0 + (S- S0)./ (2*S0), 10^(-8)));
end
```

2.1.1 result

```
CNRvartable =

  5×7 table

    Nodes    initial timestep    Value      Change      Ratio    # timestepping    iterative steps
    _____    ________________    ______    _________    ______    ______________    _______________

      62            25           1421.2           0         0          111               223
     123           100           1417.8      -3.4114        0          336               673
```

```
   245         400        1416.9     -0.85901    3.9713         805          1611
   489        1600        1416.7     -0.21495    3.9963        1712          3425
   977        6400        1416.7    -0.053749    3.9991        3490          6981


vartimetotal =

    3.0157


CNRconstanttable =

  5×7 table

    Nodes    timesteps    Value      Change      Ratio     # timestepping    iterative steps
    _____    _________    ______    _________    ______    ______________    _______________

       62          25     1421.2           0         0                25                 52
      123         100     1417.8      -3.4145         0               100                202
      245         400     1416.9     -0.86003    3.9702               400                802
      489        1600     1416.7     -0.21516    3.9972              1600               3201
      977        6400     1416.7    -0.053796    3.9995              6400              12801


contimetotal =

    3.4458
```

2.2 Q2b

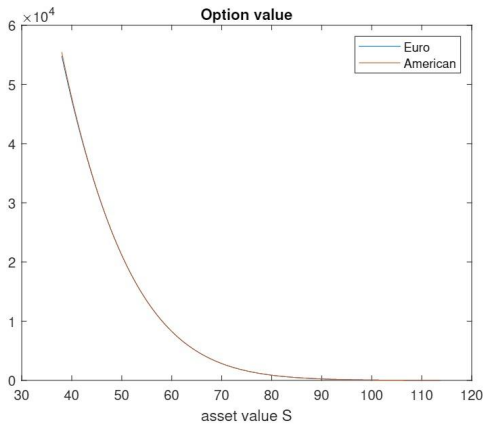
Listing 4: Q2b.m

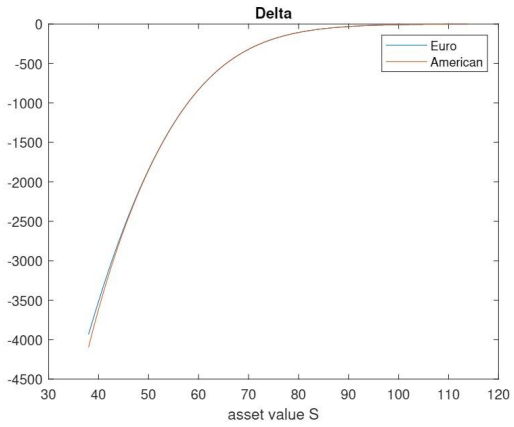
```
%% Q2b
% use CNRall, a version of CNR in Q2a that outputs the whole options value
% associated to the grid at t = 0

clearvars
close all
%%  parameters initialization
S0 = 76;
T = 1;
Deltatau = T/25; % initial time step
dnorm = 0.1;
alpha = 15;
r = 0.03;
K = S0;

S = [ 0:0.1*S0:0.4*S0, ...
    0.45*S0:0.05*S0:0.8*S0, 0.82*S0 :0.02*S0 :0.9 *S0,...
    0.91*S0 :0.01*S0 :1.1*S0, 1.12*S0 :0.02*S0 :1.2*S0, ...
    1.25*S0 :0.05*S0 :1.6*S0, 1.7*S0 :0.1*S0 :2*S0,...
    2.2*S0, 2.4*S0, 2.8*S0, 3.6*S0, 5*S0, 7.5*S0, 10*S0];
S = S';

%% Now do refinements
Scell = {};
Scell{1} = S;
% list of initial timestep
Deltataulist = zeros(5,1);
Deltataulist(1) = Deltatau;
dnormlist = zeros(5,1);
dnormlist(1) = dnorm;
% 4 refinements
for i = 1:4
    Sold = Scell{i};
    Snew = [Sold;(Sold(1:end-1) + Sold(2:end)) / 2 ];
    Snew = sort(Snew);
    Scell{i+1} = Snew;
    Deltataulist(i+1) = Deltatau/(4^i);
    dnormlist(i+1) = dnorm/(2^i);
end
% use finest grid
```

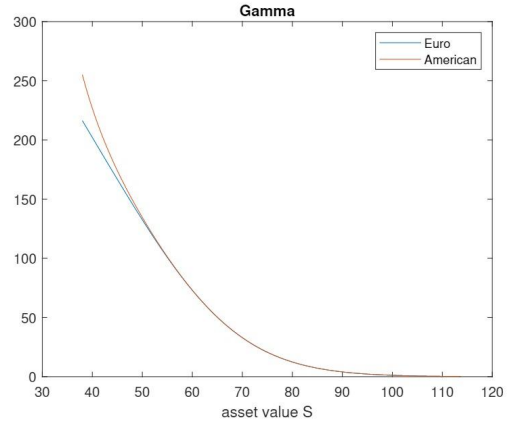
```
S = Scell{end};
Deltatau = Deltataulist(end);
dnorm = dnormlist(end);

%% load data for european, obtain data for American, plot graphs

filename = 'Q1b.mat';
myVars = {'Vat0','deltaat0','gammaat0'};

% load data from Q1b
Euro = load(filename, myVars{:});

[V0, steptotal, counter] = CNRall(S, Deltatau, S0, T, K, r,dnorm,1);

Vat0 = V0;

% Central difference for delta and gamma at t = 0

idx = (S <= 1.5*S0) & (S >= 0.5 * S0);
idx = find(idx);
index1 = idx(1);
index2 = idx(end);

deltaxright = S(index1 + 1: index2 + 1) - S(index1:index2);
deltaxleft = S(index1:index2) - S(index1 - 1:index2-1);

deltafright0 = Vat0(index1 + 1: index2 + 1) - Vat0(index1:index2);
deltafleft0 = Vat0(index1:index2) - Vat0(index1 - 1:index2-1);
gammaat0 = (deltafright0 ./ deltaxright - deltafleft0./deltaxleft) ...
    ./ ((deltaxright + deltaxleft)/2);
% not sure about this, I think this might improve the order of delta a bit
deltaat0 = (Vat0(index1+1: index2+1)-Vat0(index1-1:index2-1))...
    ./(deltaxright + deltaxleft) - gammaat0 /2 .*(deltaxright-deltaxleft);

figure(1)
plot(S(idx), Euro.Vat0(idx))
hold on
plot(S(idx),Vat0(idx))
title('Option value')
xlabel('asset value S')
legend('Euro','American')
hold off

figure(2)
plot(S(idx), Euro.deltaat0)
hold on
plot(S(idx), deltaat0)
title('Delta')
xlabel('asset value S')
legend('Euro','American')
hold off

figure(3)
plot(S(idx), Euro.gammaat0)
hold on
plot(S(idx), gammaat0)
title('Gamma')
xlabel('asset value S')
legend('Euro','American')
hold off

figure(4)
plot(S(idx), (Vat0(idx)- Euro.Vat0(idx)) )
title('American - European')
xlabel('asset value S')

figure(5)
plot(S(idx), (Vat0(idx)- Euro.Vat0(idx))./Euro.Vat0(idx) )
title('American - European, relative change')
xlabel('asset value S')

%% Discussion
```

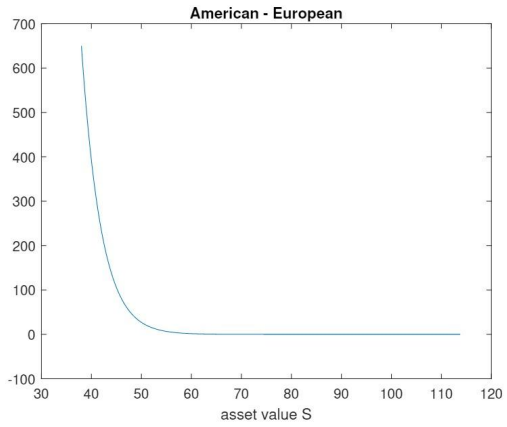
```
% Generally, we see that the European options value is smaller than or
% equal to  that of American options. This is because American options can
% exercise early and hence its value must be >= European options. However,
% such difference is very small in our case. This is because r = 0.03 is
% quite small and the interval [0.5 *S0, 1.5*S0] that we consider is not
% very at the money for the power put options, so that the early exercise
% value is not very large. But note that the as S decreases (and so payoff
% increases), such early exercise value is larger and a larger difference
% is thus shown on the left of the option value plot.
%
% Note that if we set r = 0.5, then a much bigger difference will be seen.
% If we set r = 0, then no difference will be seen.
```

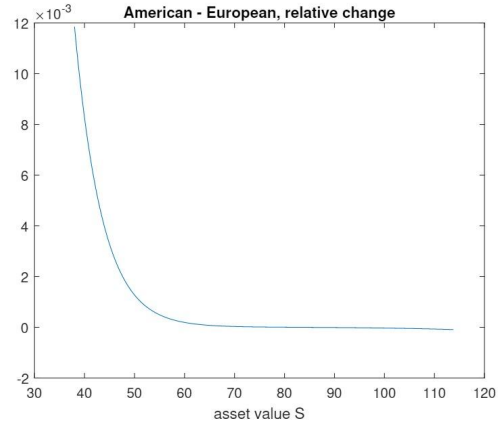
2.2.1 result




Delta
Euro
American
0
-500
-1000
-1500
-2000
-2500
-3000
-3500
-4000
-4500
30
40
50
60
70
80
90
100
110
120
asset value S


Gamma
Euro
American
300
250
200
150
100
50
0
30
40
50
60
70
80
90
100
110
120
asset value S


American - European
700
600
500
400
300
200
100
0
-100
30
40
50
60
70
80
90
100
110
120
asset value S


American - European, relative change
×10^-3
12
10
8
6
4
2
0
-2
30
40
50
60
70
80
90
100
110
120
asset value S

2.2.2 functions used

Listing 5: CNRall.m

```
function [V0, steptotal, counter] = CNRall(S, Deltatau, S0, T, K, r,dnorm,opt)
%% Description
% Inputs:
% opt = 1 for variable time, else for constant timestepping
% when opt = 1, Deltatau is the initial timestep
% when opt is else, Deltatau is the constant timestep
% Outputs:
% V0, option value at S (all the grid points) at t = 0
% steptotal, the total number of steps used for iterative solvers
% counter, the total time of choosing variable timestep; counter is just
% T/Deltatau when constant timestepping

%%  preprocessing step for simple BS equation
% so that we have positive coefficient discretization
% these will also be used for CN and CN-Rannacher
Smid = S(2:end-1);
Sleft = S(1:end-2);
Sright = S(3:end);
sigma = sigmalvf(Smid);
alphacentral = sigma.^2 .* Smid .^2 ./ ((Smid - Sleft).* (Sright - Sleft)) - ...
    r* Smid ./(Sright - Sleft);
betacentral = sigma.^2 .* Smid .^2 ./ ((Sright - Smid).* (Sright - Sleft)) + ...
    r* Smid ./(Sright - Sleft);

alphaups = sigma.^2 .* Smid .^2 ./ ((Smid - Sleft).* (Sright - Sleft));
betaups = sigma.^2 .* Smid .^2 ./ ((Sright - Smid).* (Sright - Sleft)) + ...
    r* Smid ./(Sright - Smid);

idx1 = alphacentral <0;
idx2 = betacentral <0;
idx = idx1 | idx2;

alphavec = alphacentral;
betavec = betacentral;
alphavec(idx) = alphaups(idx);
betavec(idx) = betaups(idx);
%% Some common data needed for constant and variable timestepping
m = length(S);
Vold = zeros(m,1);
Vnew = Vold;
N = T / Deltatau;
% initialize payoff
Vold = max(K-S,0).^3;

% penalty method data, tolerance and Large (penalty coefficient)
tol = 10^(-6); % as suggested by piazza
Large = 1/tol;

% Vstar, payoff at the gridpoints, for American option payoff constraints
Vstar = max(K-S,0).^3;

Bin = [[[-alphavec;0];0], [[r; (alphavec + betavec +r)];0],[[0;0];-betavec]];
% generic bin, used repeatedly below

steptotal = 0;

%% opt = 1, Use variable timestep
%%% also obtain number of steps to reach tol
%%% also need to update the Mhat and M
if opt == 1
    dtold = Deltatau;
    dtnew = 0;
    currenttau = 0;
    D = 1; % p.32 in lec 18, D = 1 fine for dollars
    counter = 0;
    % CN-Rannacher, variable time
    while currenttau < T
        counter = counter +1;
```

```
69
70          if currenttau + dtold >= T
71              dtold = T - currenttau;
72              currenttau = T;
73          else
74              currenttau = currenttau + dtold;
75          end
76
77          if counter <= 2 % first two steps are fully implicit
78              % build M, using correct \Delta tau
79              M = spdiags(dtold*Bin, -1:1, m, m);
80              [Vnew,step]= iterativesolver(Vold, Vstar, tol, Large, M,m,0);
81          else
82              % build Mhat, using correct \Delta tau
83              Mhat = spdiags((dtold/2) *Bin, -1:1, m, m);
84              [Vnew,step]= iterativesolver(Vold, Vstar, tol, Large, Mhat,m,1);
85          end
86
87          steptotal = steptotal + step;
88
89          % chooose timestep at the end
90          % assume Vold is still storing previous data, not current data
91          inter = max(D,max(abs(Vnew), abs(Vold)));
92          maxrelchange = max(abs(Vnew - Vold)./inter);
93          dtnew = dnorm/maxrelchange * dtold;
94          dtold = dtnew;
95          % update V
96          Vold = Vnew;
97      end
98
99      V0 = Vnew;
100 else
101     %% opt is not 1, use constant timestepping
102     Mhat = spdiags((Deltatau/2)*Bin, -1:1, m, m);
103     M = spdiags(Deltatau*Bin, -1:1, m, m);
104     % two steps of Fully Implicit
105     for i = 1:2
106         [Vnew,step]= iterativesolver(Vold, Vstar, tol, Large, M,m,0);
107         steptotal = steptotal + step;
108         Vold = Vnew;
109     end
110
111     for i = 1: N-2
112         [Vnew,step]= iterativesolver(Vold, Vstar, tol, Large, Mhat,m,1);
113         steptotal = steptotal + step;
114         Vold = Vnew;
115     end
116     V0 = Vnew;
117     counter = N;
118 end
119
120 end
121
122 function [Vnew,step]= iterativesolver(Vold, Vstar, tol, Large, genM,m,opt)
123 % opt = 1 for CN, else fully implicit
124 % genM= Mhat when opt = 1 (i.e. CN), o/w genM = M (i.e. fully implicit)
125
126 error = 10000; % set the error to be some large number
127 step = 0;
128 Viterold = Vold;
129 I = speye(m);
130 while error >= tol
131     % Vold is V^n
132     % Viterold is (V^{n+1})^k, Viternew will be (V^{n+1})^(k+1)
133     largediagonal = zeros(m,1);
134     largeidx = Viterold < Vstar;
135     largediagonal(largeidx) = Large;
136     P = spdiags(largediagonal, 0,m,m);
137     if opt == 1 % CN
138         y = (I - genM)*Vold + P* Vstar;
139         Viternew = (I + genM + P) \ y;
140     else % fully implicit
```

```
        y = Vold + P* Vstar;
        Viternew = (I + genM + P) \ y;
    end

    inter = abs(Viternew - Viterold)./max(1,abs(Viternew));
    error = max(inter);
    step = step + 1;
    if error < tol
        Vnew = Viternew;
        break;
    end
    Viterold = Viternew;
end

end

function out = sigmalvf(S)
% note that the local volatility function actually is independent of t
% vectorized output, assume column vector input
    alpha = 15;
    S0 = 76;
    out = alpha./ (max(S0 + (S- S0)./ (2*S0), 10^(-8)));
end
```

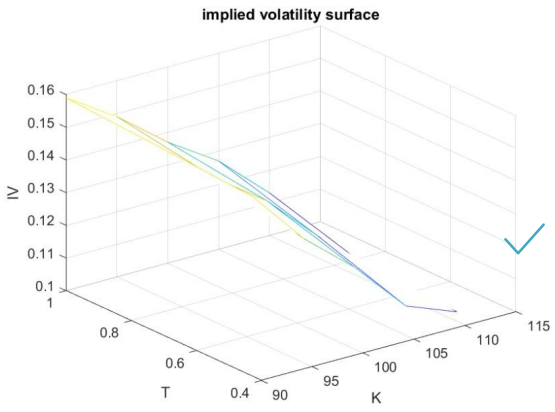
3 Q3

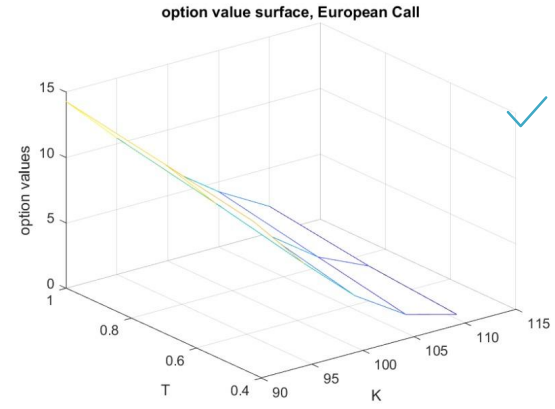
3.1 Q3a

Listing 6: Q3a.m

```
%% Q3a

clear vars
close all

S0 =100;
r = 0.03;

Klist = [0.9, 0.95, 1, 1.05, 1.1]* S0;
Tlist = [0.425, 0.695, 1];

% IV table
table2 = zeros(3,5);
table2(1,:) = [0.155, 0.138, 0.125, 0.109, 0.103];
table2(2,:) = [0.157, 0.144, 0.133, 0.118, 0.104];
table2(3,:) = [0.159, 0.149, 0.137, 0.127, 0.113];

[X,Y] = meshgrid(Klist, Tlist);

figure(1)
mesh(X,Y, table2)
xlabel('K')
ylabel('T')
zlabel('IV')
title('implied volatility surface')

optiontable = zeros(3,5);

for j = 1:5
    for i = 1:3
        [Call , put] = blsprice(S0, Klist(j), r, Tlist(i), table2(i,j));
        optiontable(i,j) = Call;
    end
end

figure(2)
mesh(X,Y, optiontable)
xlabel('K')
ylabel('T')
zlabel('option values')
title('option value surface, European Call')

%% Q3a Discussion
% We see that implied volatility surface and option value surface have
% similar shape. Maybe option value surface is a little bit smoother?
```

3.1.1 result


implied volatility surface
IV
0.16
0.15
0.14
0.13
0.12
0.11
0.1
T
1
0.8
0.6
0.4
K
90
95
100
105
110
115


option value surface, European Call
option values
15
10
5
0
T
1
0.8
0.6
0.4
K
90
95
100
105
110
115

3.2 Q3b

Listing 7: callpricer_CNR.m

```
%% Q3b, only CNR

function V0CNRan = callpricer_CNR(S, Deltatau, x, T, K, r)
% output option value at t =0, for all gridpoints, using fully implicit,
% Crank Nicolson, and CN-Rannacher
%
% input: x is the set of parameters. x =(w_1,w_2,u_1,u_2,v_1,v_2)', used in
% local volatility

m = length(S);
Vold = zeros(m,1);
Vnew = Vold;
N = T / Deltatau;
%% Boundary condition concern:
% initialize payoff, call option payoff
% Note that we adopt just the payoff as the boundary condition. By p.9 in
% Lec16, it is suggested to use S_m for call option. But I think the
% difference is small and adopt S_m - K instead, since S_m is quite large.
Vold = max(S-K,0);

%% preprocessing step for simple BS equation
% so that we have positive coefficient discretization
% these will also be used for CN and CN-Rannacher
Smid = S(2:end-1);
Sleft = S(1:end-2);
Sright = S(3:end);
sigma = sigmalvf(Smid,x);
alphacentral = sigma.^2 .* Smid .^2 ./ ((Smid - Sleft).* (Sright - Sleft)) -...
    r* Smid ./(Sright - Sleft);
betacentral = sigma.^2 .* Smid .^2 ./ ((Sright - Smid).* (Sright - Sleft)) +...
    r* Smid ./(Sright - Sleft);

alphaups = sigma.^2 .* Smid .^2 ./ ((Smid - Sleft).* (Sright - Sleft));
betaups = sigma.^2 .* Smid .^2 ./ ((Sright - Smid).* (Sright - Sleft)) + ...
    r* Smid ./(Sright - Smid);

idx1 = alphacentral <0;
idx2 = betacentral <0;
idx = idx1 | idx2;

alphavec = alphacentral;
betavec = betacentral;
alphavec(idx) = alphaups(idx);
betavec(idx) = betaups(idx);

%% Fully implicit

% build the matrix M by spdiags
Bin = Deltatau* [[[-alphavec;0];0], [[r; (alphavec + betavec +r)];0],[[0;0];-...
    betavec]];
M = spdiags(Bin, -1:1, m, m);
% spy(M) % see the nonzero entries of the matrix M
% Note that our M is time independent
I = speye(m);
[L,U,P] = lu(M+I);
% increase performance since the LU factorization is only computed once and
% being reused in each linear system solving
% to solve (M+I)x = b, y =L\ (P*b), x = U\y

% for n = 1 : N
%     y = L\(P*Vold);
```

```
%     Vnew = U\y;
%     Vold = Vnew;
% end
% V0fullyimplicit = Vnew;


%% Crank-Nicolson
%%% only compute initial option value
%%% can easily modified to obtain options values at each grid points
% Vold = zeros(m,1);
% Vnew = Vold;
% % initialize payoff, call option payoff
% Vold = max(S-K,0);

Bin = Deltatau/2* [[[-alphavec;0];0], [[r; (alphavec + betavec +r)
    ];0],[[0;0];-betavec]];
Mhat = spdiags(Bin, -1:1, m, m);
% Note that our Mhat is time independent
I = speye(m);
[L1,U1,P1] = lu(Mhat+I);


% for n = 1 : N
%     y = L1\(P1*(I-Mhat)*Vold);
%     Vnew = U1\y;
%     Vold = Vnew;
% ends
% V0CN = Vnew;

%% CN-Rannacher
Vold = zeros(m,1);
Vnew = Vold;
% initialize payoff, call option payoff
Vold = max(S-K,0);

% two steps of fully implicit
for n = 1 : 2
    y = L\(P*Vold);
    Vnew = U\y;
    Vold = Vnew;
end

% use CN after that
for n = 1 : N-2
    y = L1\(P1*(I-Mhat)*Vold);
    Vnew = U1\y;
    Vold = Vnew;
end
V0CNRan = Vnew;

end

function out = sigmalvf(S,x)
%%  local volatility by simple neural network, (9) in asg4
%
% note that the local volatility function actually is independent of t
% vectorized output, assume column vector input
%
% inputs: S, gridpoints of S
%         x, set of parameters. x =(w_1,w_2,u_1,u_2,v_1,v_2)'
w1 = x(1);
w2 = x(2);
u1 = x(3);
u2 = x(4);
v1 = x(5);
v2 = x(6);
```

```
y1 = 1./(u1 + v1 * S);
y2 = 1./(u2 + v2 *S);
out = 1./(1 + exp(w1 * y1 + w2* y2));
end
```

3.3 Q3c

Listing 8: myfun.m

```
%% Q3c

function [F,J] = myfun(x)
% marketvalue: V_0^mkt, a column vector

% parameters and data preparation
S0 =100;
r = 0.03;
Klist = [0.9, 0.95, 1, 1.05, 1.1]* S0;
Tlist = [0.425, 0.695, 1];
N = 100; % what should be the correct number of steps?
% can change to N = 100

S = [ 0:0.1*S0:0.4*S0, ...
    0.45*S0:0.05*S0:0.8*S0, 0.82*S0 :0.02*S0 :0.9 *S0,...
    0.91*S0 :0.01*S0 :1.1*S0, 1.12*S0 :0.02*S0 :1.2*S0, ...
    1.25*S0 :0.05*S0 :1.6*S0, 1.7*S0 :0.1*S0 :2*S0,...
    2.2*S0, 2.4*S0, 2.8*S0, 3.6*S0, 5*S0, 7.5*S0, 10*S0];
S = S';

% can do the refinements following
Sold = S;

for i = 1:2
    Snew = [Sold;(Sold(1:end-1) + Sold(2:end)) / 2 ];
    Snew = sort(Snew);
    Sold = Snew;
end

S = Snew;


% IV table
table2 = zeros(3,5);
table2(1,:) = [0.155, 0.138, 0.125, 0.109, 0.103];
table2(2,:) = [0.157, 0.144, 0.133, 0.118, 0.104];
table2(3,:) = [0.159, 0.149, 0.137, 0.127, 0.113];

market = zeros(3,5);

for j = 1:5
    for i = 1:3
        [Call , ~] = blsprice(S0, Klist(j), r, Tlist(i), table2(i,j));
        market(i,j) = Call;
    end
end

market = reshape(market, numel(market), []);

F = zeros(length(Tlist),length(Klist));
% Objective function values at x, stored as in table 2

S0id = S == S0;

for i = 1:length(Tlist)
    T = Tlist(i);
    Deltatau = T/N;
```

```
    for j = 1 : length(Klist)
        K = Klist(j);
        V0CNRan = callpricer_CNR(S, Deltatau, x, T, K, r);
        % only consider CNR, note that other methods can be obtained in
        % callpricer.m
        value = V0CNRan(S0id);
        F(i,j) = value;
    end
end

% Note that F now only contains value of options depending on x, not
% subtracted by the the market values yet

% reshape F, column by column
F = reshape(F, numel(F),[]);


% Note that we don't need to consider the constant market values when doing
% Jacobian. Hence we will subtract F by marketvalue later.
if nargout > 1 % Two output arguments
    change = sqrt(eps);
    Fnew = zeros(length(Tlist),length(Klist));
    J = zeros(numel(Fnew), length(x)); % Jacobian of the function evaluated
        at x

    for k = 1 : length(x)
        Fnew = zeros(length(Tlist),length(Klist));
        % calculate the k-th column of J
        dir = zeros(length(x), 1); % direction of derivative
        dir(k) = 1;
        for i = 1:length(Tlist)
            T = Tlist(i);
            Deltatau = T/N;
            for j = 1 : length(Klist)
                K = Klist(j);
                V0CNRan = callpricer_CNR(S, Deltatau, x+ change*dir, T, K, r)
                    ;
                % only consider CNR, note that other methods can be obtained
                    in
                % callpricer.m
                Fnew(i,j) = V0CNRan(S0id);
            end
        end
        Fnew = reshape(Fnew, numel(Fnew), []);
        J(:,k) = (Fnew - F) / change;
    end

end

F = F - marketv

end
```

3.4 Q3d

Listing 9: Q3d.m

```
%% Q3d

clear vars
close all

%% Data preparation
S0 =100;
r = 0.03;

```

```
Klist = [0.9, 0.95, 1, 1.05, 1.1]* S0;
Tlist = [0.425, 0.695, 1];

% IV table
table2 = zeros(3,5);
table2(1,:) = [0.155, 0.138, 0.125, 0.109, 0.103];
table2(2,:) = [0.157, 0.144, 0.133, 0.118, 0.104];
table2(3,:) = [0.159, 0.149, 0.137, 0.127, 0.113];

market = zeros(3,5);

for j = 1:5
    for i = 1:3
        [Call , put] = blsprice(S0, Klist(j), r, Tlist(i), table2(i,j));
        market(i,j) = Call;
    end
end

market = reshape(market, numel(market), []);

%% Levenberg-Marquardt method
options = optimset('Jacobian', 'on', 'Algorithm', 'levenberg-marquardt',...
    'Display', 'iter', 'MaxIter', 50);

% initial x
x0 = [0,0,1,1,0,0]';

% xopt: optimal parameters xopt
% cal_error: calibration error
[xopt, cal_error] = lsqnonlin(@myfun, x0, [],[], options)

% for more information on the iterative display, see
% https://www.mathworks.com/help/optim/ug/iterative-display.html#f92387

%% IV from true model and the IV from the optimal parameters
Vfromxopt = myfun(xopt) + market;
Vfromxopt = reshape(Vfromxopt, length(Tlist), length(Klist))

[X,Y] = meshgrid(Klist, Tlist);
IVfromxopt = blsimpv(S0, X, r, Y, Vfromxopt)

% IVtrue = blsimpv(S0, X,r,Y, reshape(market, 3,5))
% note that table 2 are the implied volatilties from the true model
table2

figure(1)
mesh(X,Y, table2,'FaceColor','blue')
xlabel('K')
ylabel('T')
zlabel('IV')
title('implied volatility surface')
hold on
mesh(X,Y, IVfromxopt,'FaceColor','red')
legend('true','from xpot')

relerrorIV = max(abs(IVfromxopt- table2)./table2,[],'all')

%% Discussion
% We see that in general, our implied volatilties calibration does a fairly
% good job, with the relative error being around 12%. That is, the implied
% volatilties from the option values computed by model as given by xopt,
% are roughly 88% accurate, when compared to the true model, i.e. the
% implied volatilities in table 2. Based on the simplicity of our model
% (simple neural network), this is already quite good.
%
% Some thoughts on the formulation of the minimization problem. Note that
```

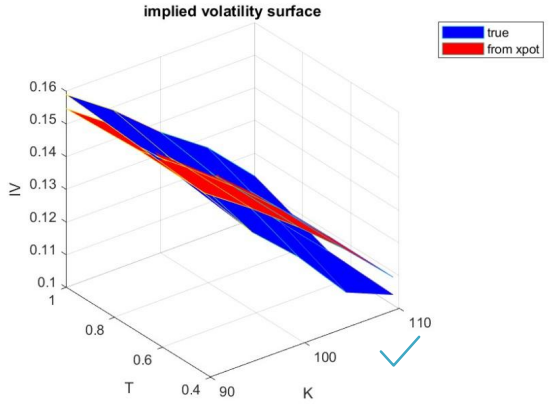
```
% the objective function is the 1/2 * sum of square of difference between
% option values from our model and option values from true model (or
% so-called market prices). What if we directly incorporate the difference
% between implied volatilties in the objective function? For instance, we
% can define F to be the sum of the square of the difference between the
% true implied volatilities and the implied volatilities from our model.
% What are the advantages and disadvantages of such method?
```

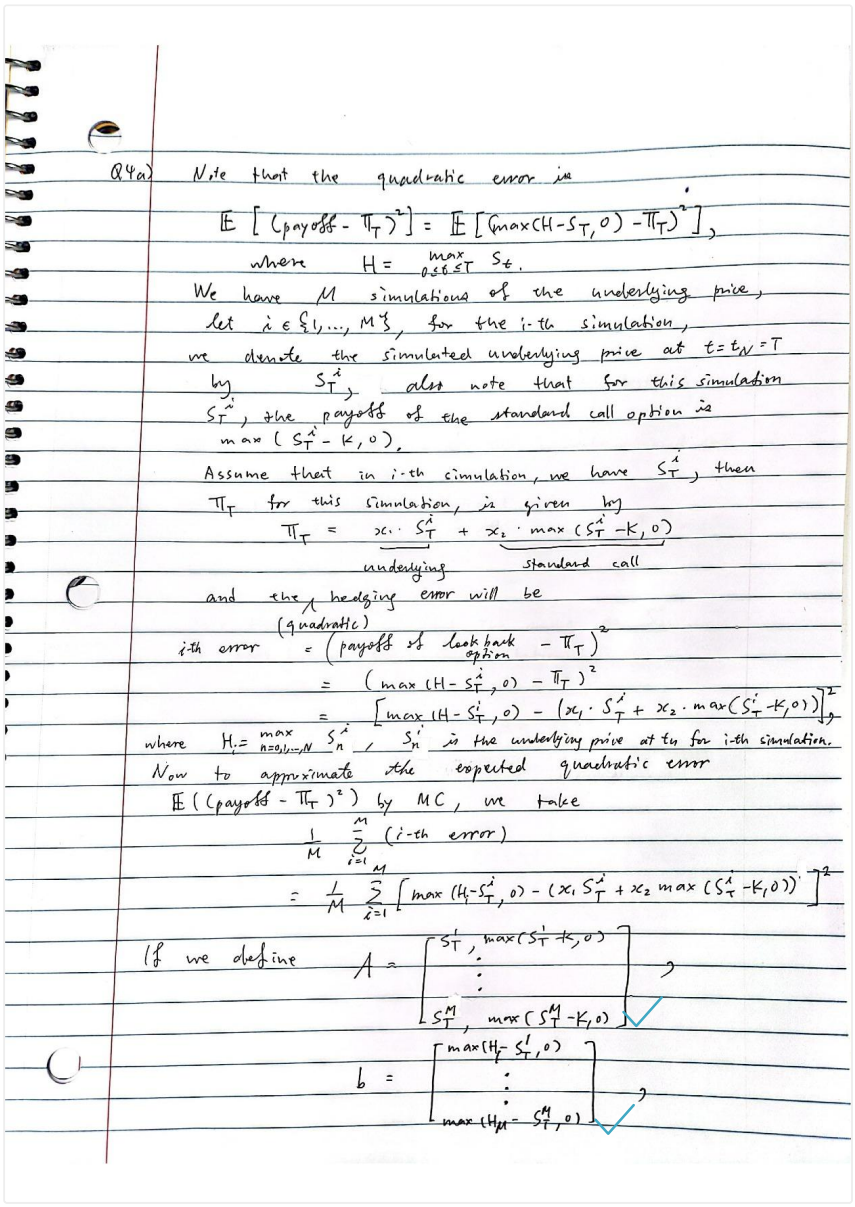
3.4.1 result

Iteration	Func-count	Residual	First-Order optimality	Lambda	Norm of step
0	1	1980.08	1.37e+03	0.01	
1	2	50.891	9.11e+03	0.001	1.01875
2	7	4.68525	2.97e+03	10	0.138677
3	8	1.31611	462	1	0.13255
4	10	1.16802	2e+03	10	0.132519
5	15	0.724439	97.4	100000	0.000422735
6	16	0.689181	174	10000	0.000411243
7	18	0.648882	173	100000	0.000432767
8	19	0.609822	149	10000	0.000427082
9	21	0.575115	118	100000	0.000404782
10	22	0.545613	89.8	10000	0.000375339
11	24	0.520967	67.6	100000	0.000344572
12	25	0.500456	51.1	10000	0.0003153
13	27	0.483341	39	100000	0.000288616
14	28	0.468978	30.1	10000	0.000264765
15	29	0.448214	1.13e+03	1000	0.0015855
16	30	0.442871	1.71e+03	100	0.00258785
17	31	0.3611	184	10	0.00131142
18	32	0.359626	43.8	1	0.005493
19	34	0.358937	34.8	10	0.00572245
20	35	0.358233	38	1	0.00589012
21	37	0.35749	42.7	10	0.00606335
22	38	0.356703	48	1	0.00624244
23	40	0.355871	53.8	10	0.00642753
24	41	0.354992	60.2	1	0.00661423
25	43	0.354066	67.1	10	0.0068002
26	44	0.353094	74.5	1	0.00697956
27	46	0.352077	82.1	10	0.00714784
28	47	0.351023	89.4	1	0.00729222
29	49	0.349937	95.9	10	0.00740341
30	50	0.348832	100	1	0.00746188
31	52	0.347724	101	10	0.00744865
32	53	0.346638	96.7	1	0.00733851
33	55	0.345606	87.1	10	0.00711444
34	56	0.344665	73.5	1	0.00677701
35	58	0.343843	58.6	10	0.00635237
36	59	0.343148	44.8	1	0.00587911
37	61	0.342568	33.5	10	0.00540346
38	62	0.342084	24.9	1	0.00495779
39	64	0.341679	18.7	10	0.00455527
40	65	0.341336	14.2	1	0.00419963
41	67	0.341041	10.9	10	0.00389149
42	68	0.340786	8.53	1	0.00362156
43	70	0.340563	6.79	10	0.00338887
44	71	0.340365	5.48	1	0.00318326
45	73	0.340189	4.49	10	0.00300441
46	74	0.340031	3.72	1	0.00284495
47	76	0.339888	3.12	10	0.00270487
48	77	0.339758	2.64	1	0.00257899
49	78	0.33948	182	0.1	0.0245411
50	80	0.338329	50.5	1	0.016586

```
Solver stopped prematurely.

lsqnonlin stopped because it exceeded the iteration limit,
options.MaxIterations = 5.000000e+01.


xopt =
```

```
   0.443060999978524
   0.604616972528008
   1.155723356881603
   1.086870579899331
   0.123508674879181
  -0.007532572532176

cal_error =

   0.338329243369058

Vfromxopt =

  11.711623569997183   7.565274404568785   4.136163782120330   1.720735068697083   0.440370920715182
  12.927035435653890   8.947034335766618   5.525697661161873   2.853969561159865   1.085357244629239
  14.176673782379037  10.288162733514328   6.849099907922974   3.996240518166629   1.872033627894845

IVfromxopt =

   0.155198783772656   0.145197309449309   0.134158618274803   0.121975207156851   0.108383241200401
   0.155773676942736   0.145440158550793   0.134283239095342   0.122102963506448   0.108616712470348
   0.154690574962364   0.144439857196952   0.133407563209495   0.121405132770362   0.108168667457043

table2 =

   0.155000000000000   0.138000000000000   0.125000000000000   0.109000000000000   0.103000000000000
   0.157000000000000   0.144000000000000   0.133000000000000   0.118000000000000   0.104000000000000
   0.159000000000000   0.149000000000000   0.137000000000000   0.127000000000000   0.113000000000000

relerrorIV =

   0.119038597769280
```





Q4a) Note that the quadratic error is

$$\mathbb{E}\left[(\text{payoff} - \Pi_T)^2\right] = \mathbb{E}\left[(\max(H - S_T, 0) - \Pi_T)^2\right],$$

where $H = \max_{0 \le t \le T} S_t$.

We have M simulations of the underlying price, let $i \in \{1, \dots, M\}$ for the i-th simulation, we denote the simulated underlying price at $t = t_N = T$ by S_T^i, also note that for this simulation S_T^i, the payoff of the standard call option is $\max(S_T^i - K, 0)$.

Assume that in i-th simulation, we have S_T^i, then Π_T for this simulation, is given by

$$\Pi_T = \underbrace{x_1 \cdot S_T^i}_{\text{underlying}} + \underbrace{x_2 \cdot \max(S_T^i - K, 0)}_{\text{standard call}}$$

and the (quadratic) hedging error will be

$$\begin{aligned} i\text{-th error} &= \left(\text{payoff of } \underset{\text{option}}{\text{look back}} - \Pi_T\right)^2 \\ &= \left(\max(H - S_T^i, 0) - \Pi_T\right)^2 \\ &= \left[\max(H - S_T^i, 0) - (x_1 \cdot S_T^i + x_2 \cdot \max(S_T^i - K, 0))\right]^2, \end{aligned}$$

where $H_i := \max_{n=0,1,\dots,N} S_n^i$, S_n^i is the underlying price at t_n for i-th simulation.

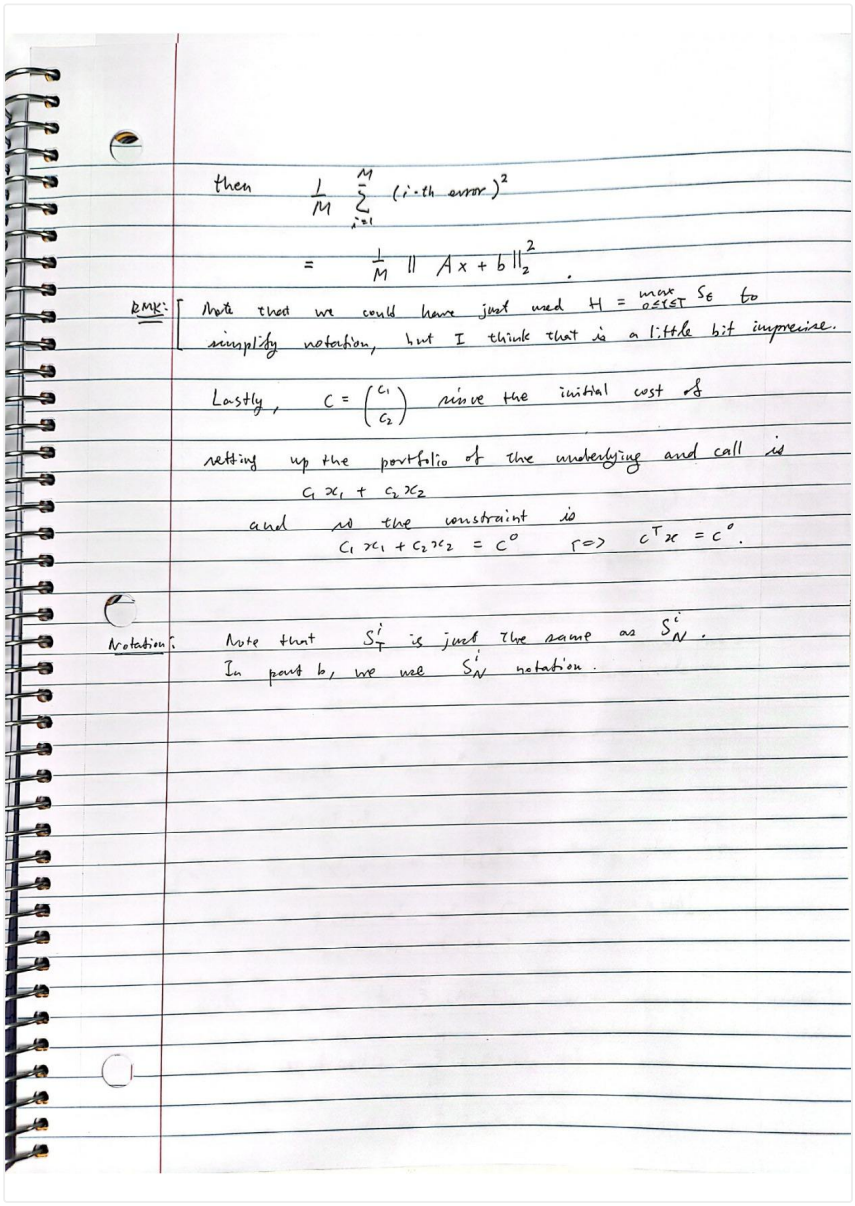
Now to approximate the expected quadratic error $\mathbb{E}\left((\text{payoff} - \Pi_T)^2\right)$ by MC, we take

$$\begin{aligned} &\frac{1}{M} \sum_{i=1}^{M} (i\text{-th error}) \\ &= \frac{1}{M} \sum_{i=1}^{M} \left[\max(H_i - S_T^i, 0) - (x_1 S_T^i + x_2 \max(S_T^i - K, 0))\right]^2 \end{aligned}$$

If we define

$$A = \begin{bmatrix} S_T^1, & \max(S_T^1 - K, 0) \\ \vdots & \vdots \\ S_T^M, & \max(S_T^M - K, 0) \end{bmatrix}$$

$$b = \begin{bmatrix} \max(H_1 - S_T^1, 0) \\ \vdots \\ \max(H_M - S_T^M, 0) \end{bmatrix}$$



then $\frac{1}{M}\sum_{i=1}^{M} (i\text{-th error})^2$

$= \frac{1}{M} \| Ax + b \|_2^2$.

RMK: Note that we could have just used $H = \max_{0 \le t \le T} S_t$ to simplify notation, but I think that is a little bit imprecise.

Lastly, $C = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}$ since the initial cost of setting up the portfolio of the underlying and call is $c_1 x_1 + c_2 x_2$

and so the constraint is

$c_1 x_1 + c_2 x_2 = c^0 \iff c^T x = c^0$.

Notation: Note that S_T^i is just the same as S_N^i.

In part b, we use S_N^i notation.

4 Q4

4.1 Q4b

Use the notation as in the question. Note that in the following pseudo code, vectorized approach is employed. Also, we assume operations between scalars and vectors will be done by broadcasting the scalars appropriately. Also, by $\max(x, y)$, where x, y are vectors of the same size, we mean a component-wise comparison and output a vector with the same size with each i-th entry being the largest element among $(x)_i, (y)_i$.

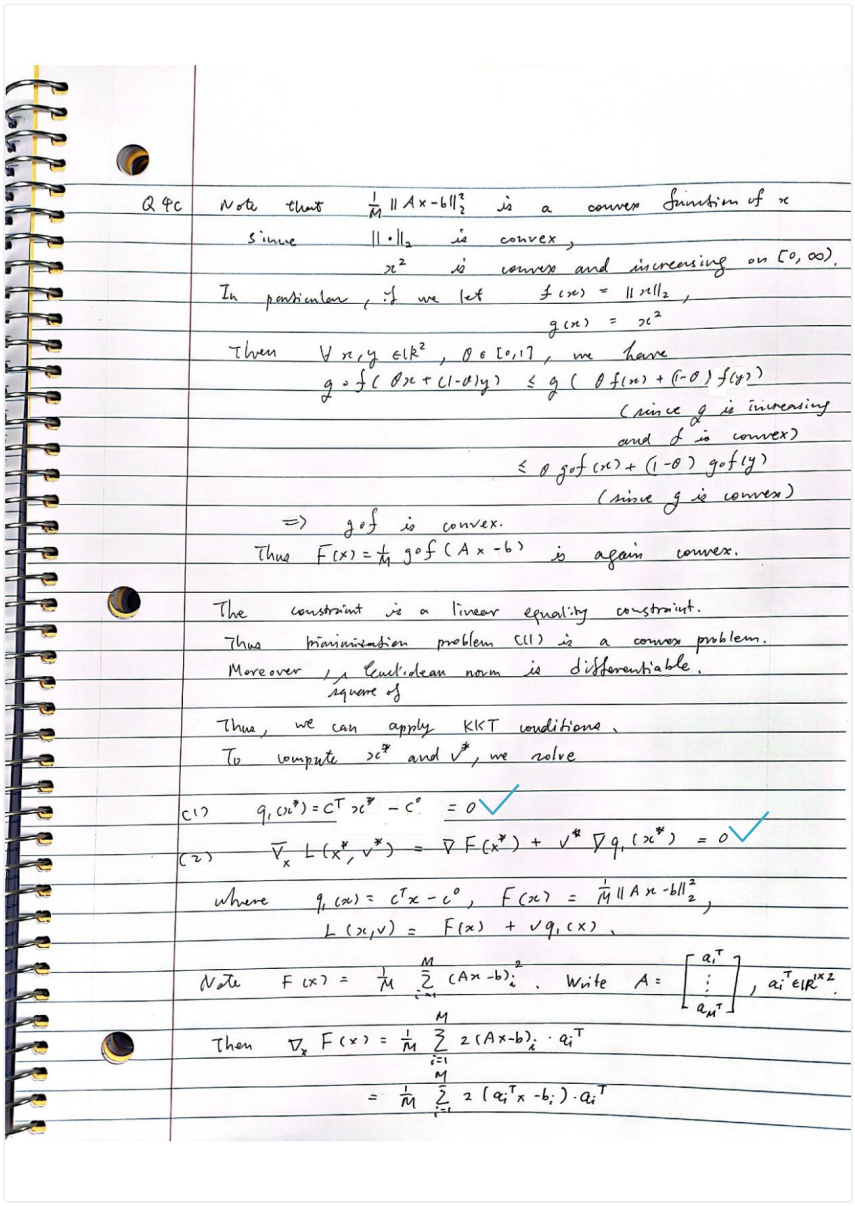
Note that we assume the existence of a function which returns a random variable $\phi \sim \mathcal{N}(0, 1)$. For simplicity, by $randn(M, 1)$ we mean it generates M random variables that are $\mathcal{N}(0, 1)$, stored in a row vector. Moreover, we make an additional assumption that whenever we call $randn(M, 1)$, the resulting random variables are independent among themselves and also independent of all previously generated random variables.

Lastly, we also uses $ones(M, 1)$ to mean it generates a M-by-1 row vector with each entry being 1. Also, we use $zeros(m, n)$ to mean a m-by-n matrix with entries all being 0.

Procedure 1 Monte Carlo method, A, b, c computation

Input: $M, N, T, K, \mu, \sigma, c_1, c_2$ ▷ scalars inputs
Output: A, b, c ▷ for optimization problem

$\Delta t = T/N$
$S_0 = ones(M, 1) \cdot c_1$
$H = S_0$
$A = zeros(M, 2)$
$b = zeros(M, 1)$
$c = [c_1, c_2]^T$
for $j = 1, \ldots, N$ **do**
  $\phi = randn(M, 1)$ ▷ M independent $\mathcal{N}(0, 1)$
  $S_j = S_{j-1} \cdot e^{(\mu - \frac{\sigma^2}{2})\Delta t + \sigma\phi\sqrt{\Delta t}}$
  $H = \max(H, S_j)$ ▷ component-wise
end for
$A_{:,1} = S_N$ ▷ first column of A
$A_{:,2} = \max(S_N - K, 0)$ ▷ second column
$b = \max(H - S_N, 0)$ ▷ component-wise



Q4c Note that $\frac{1}{M}\|Ax-b\|_2^2$ is a convex function of x

since $\|\cdot\|_2$ is convex,

x^2 is convex and increasing on $[0,\infty)$.

In particular, if we let $f(x) = \|x\|_2$,

$g(x) = x^2$

Then $\forall x, y \in \mathbb{R}^2$, $\theta \in [0,1]$, we have

$g \circ f(\theta x + (1-\theta)y) \leq g(\theta f(x) + (1-\theta) f(y))$

(since g is increasing and f is convex)

$\leq \theta\, g \circ f(x) + (1-\theta)\, g \circ f(y)$

(since g is convex)

$\Rightarrow$ $g \circ f$ is convex.

Thus $F(x) = \frac{1}{M} g \circ f(Ax-b)$ is again convex.

The constraint is a linear equality constraint.

Thus minimization problem (1) is a convex problem.

Moreover, Euclidean norm is differentiable.

Thus, we can apply KKT conditions.

To compute x^* and v^*, we solve

(1) $q_1(x^*) = c^T x^* - c^0 = 0$ ✓

(2) $\nabla_x L(x^*, v^*) = \nabla F(x^*) + v^* \nabla q_1(x^*) = 0$ ✓

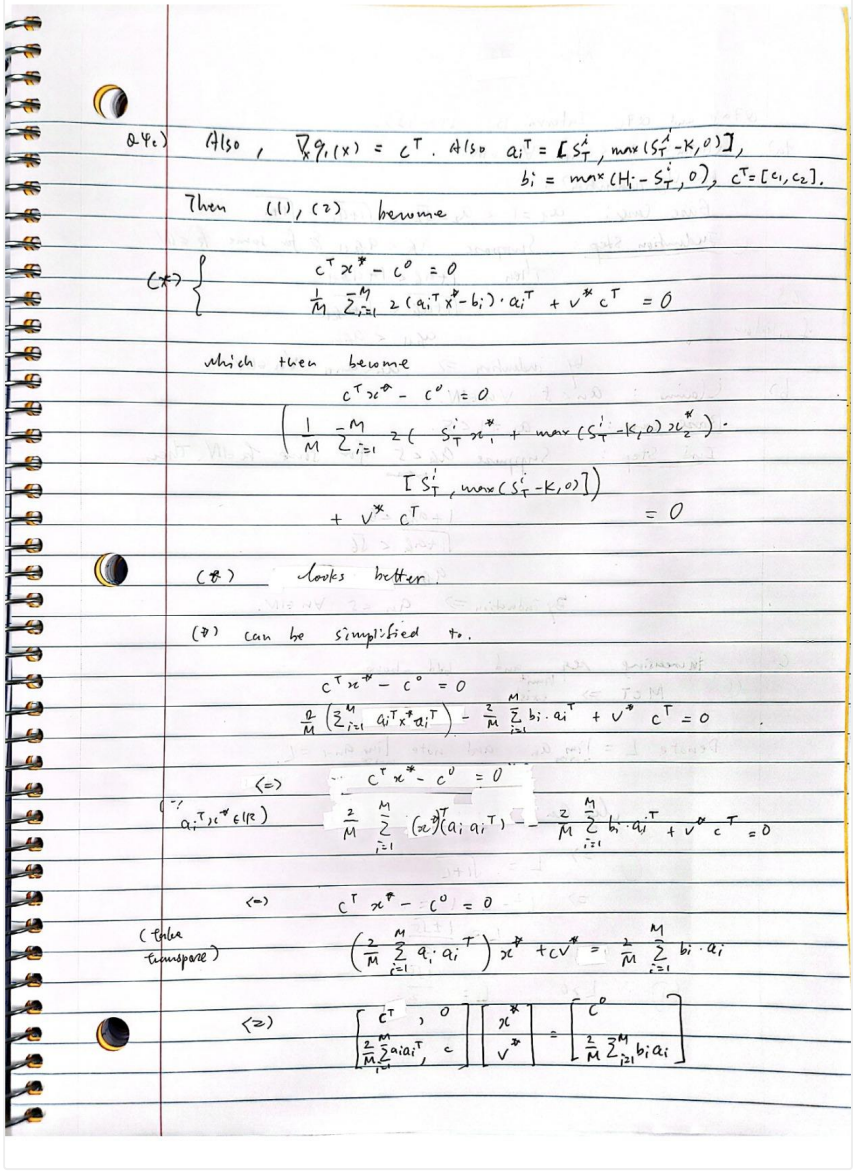
where $q_1(x) = c^T x - c^0$, $F(x) = \frac{1}{M}\|Ax-b\|_2^2$,

$L(x,v) = F(x) + v q_1(x)$,

Note $F(x) = \frac{1}{M}\sum_{i=1}^{M} (Ax-b)_i^2$. Write $A = \begin{bmatrix} a_1^T \\ \vdots \\ a_M^T \end{bmatrix}$, $a_i^T \in \mathbb{R}^{1\times 2}$.

Then $\nabla_x F(x) = \frac{1}{M}\sum_{i=1}^{M} 2(Ax-b)_i \cdot a_i^T$

$= \frac{1}{M}\sum_{i=1}^{M} 2(a_i^T x - b_i)\cdot a_i^T$



Q4c) Also, $\nabla g_i(x) = c^T$. Also $a_i^T = [S_T^i, \max(S_T^i - K, 0)]$,
$b_i = \max(H_i - S_T^i, 0)$, $c^T = [c_1, c_2]$.

Then (1), (2) become

$$(*) \begin{cases} c^T x^* - c^0 = 0 \\ \frac{1}{M} \sum_{i=1}^{M} 2(a_i^T x^* - b_i) \cdot a_i^T + v^* c^T = 0 \end{cases}$$

which then become

$$c^T x^* - c^0 = 0$$
$$\left(\frac{1}{M} \sum_{i=1}^{M} 2\left(S_T^i x_1^* + \max(S_T^i - K, 0) x_2^* \right) \cdot [S_T^i, \max(S_T^i - K, 0)] \right) + v^* c^T = 0$$

(*) looks better.

(*) can be simplified to.

$$c^T x^* - c^0 = 0$$
$$\frac{2}{M} \left(\sum_{i=1}^{M} a_i^T x^* a_i^T \right) - \frac{2}{M} \sum_{i=1}^{M} b_i \cdot a_i^T + v^* c^T = 0$$

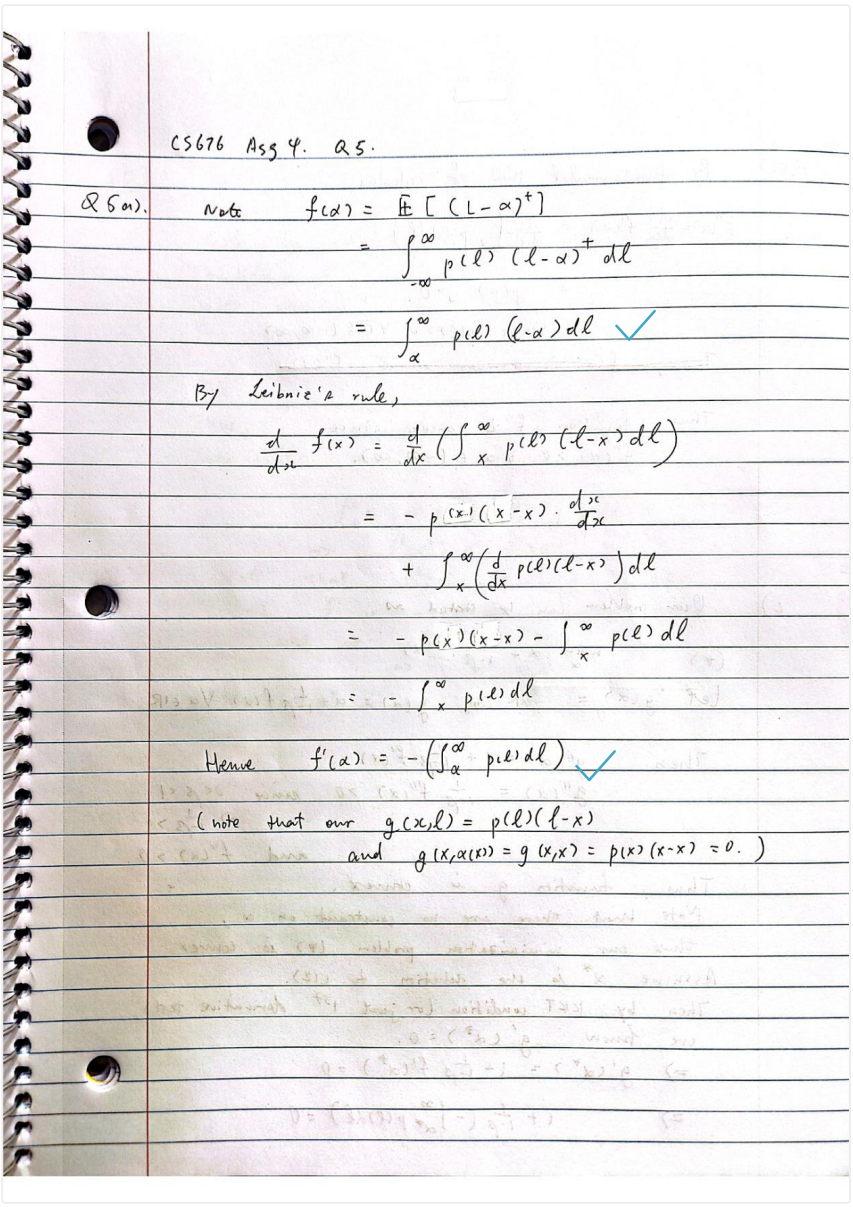
$\Leftrightarrow$ $c^T x^* - c^0 = 0$

($\because a_i^T x^* \in \mathbb{R}$) $\frac{2}{M} \sum_{i=1}^{M} (x^*)^T (a_i a_i^T) - \frac{2}{M} \sum_{i=1}^{M} b_i \cdot a_i^T + v^* c^T = 0$

$\Leftarrow$ $c^T x^* - c^0 = 0$

(take transpose) $\left(\frac{2}{M} \sum_{i=1}^{M} a_i \cdot a_i^T \right) x^* + c v^* = \frac{2}{M} \sum_{i=1}^{M} b_i \cdot a_i$

$\Leftrightarrow$
$$\begin{bmatrix} c^T & 0 \\ \frac{2}{M} \sum_{i=1}^{M} a_i a_i^T & c \end{bmatrix} \begin{bmatrix} x^* \\ v^* \end{bmatrix} = \begin{bmatrix} c^0 \\ \frac{2}{M} \sum_{i=1}^{M} b_i a_i \end{bmatrix}$$



CS676 Ass 4. Q5.

Q5(a). Note $f(\alpha) = \mathbb{E}[(L-\alpha)^+]$

$$= \int_{-\infty}^{\infty} p(l)\,(l-\alpha)^+\,dl$$

$$= \int_{\alpha}^{\infty} p(l)\,(l-\alpha)\,dl \quad \checkmark$$

By Leibniz's rule,

$$\frac{d}{dx} f(x) = \frac{d}{dx}\left(\int_x^{\infty} p(l)\,(l-x)\,dl\right)$$

$$= -p(x)(x-x)\cdot\frac{dx}{dx} + \int_x^{\infty}\left(\frac{d}{dx}\,p(l)(l-x)\right)dl$$

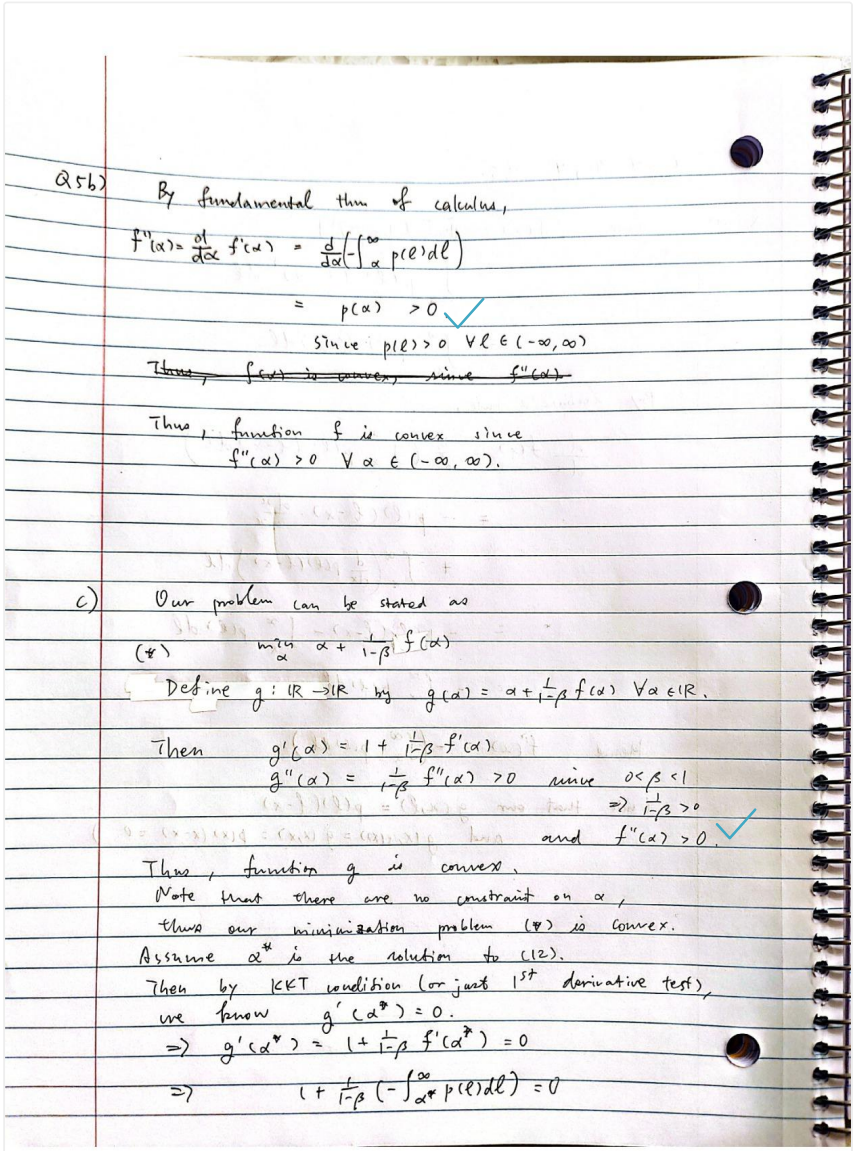
$$= -p(x)(x-x) - \int_x^{\infty} p(l)\,dl$$

$$= -\int_x^{\infty} p(l)\,dl$$

Hence $f'(\alpha) = -\left(\int_{\alpha}^{\infty} p(l)\,dl\right)$ ✓

(note that our $g(x,l) = p(l)(l-x)$

and $g(x,\alpha(x)) = g(x,x) = p(x)(x-x) = 0.$)



Q5b) By fundamental thm of calculus,

$f'(\alpha) = \frac{d}{d\alpha} f(\alpha) = \frac{d}{d\alpha}\left(-\int_\alpha^\infty p(\ell)\,d\ell\right)$

$= p(\alpha) > 0$ ✓

since $p(\ell) > 0 \quad \forall \ell \in (-\infty, \infty)$

~~Thus, $f(\alpha)$ is convex, since $f''(\alpha)$~~

Thus, function f is convex since

$f''(\alpha) > 0 \quad \forall \alpha \in (-\infty, \infty)$.

c) Our problem can be stated as

$(*) \qquad \min_\alpha \; \alpha + \frac{1}{1-\beta} f(\alpha)$

Define $g : \mathbb{R} \to \mathbb{R}$ by $g(\alpha) = \alpha + \frac{1}{1-\beta} f(\alpha) \quad \forall \alpha \in \mathbb{R}$.

Then $g'(\alpha) = 1 + \frac{1}{1-\beta} f'(\alpha)$

$g''(\alpha) = \frac{1}{1-\beta} f''(\alpha) > 0$ since $0 < \beta < 1$

$\Rightarrow \frac{1}{1-\beta} > 0$

and $f''(\alpha) > 0$. ✓

Thus, function g is convex.

Note that there are no constraint on α,

thus our minimization problem $(*)$ is convex.

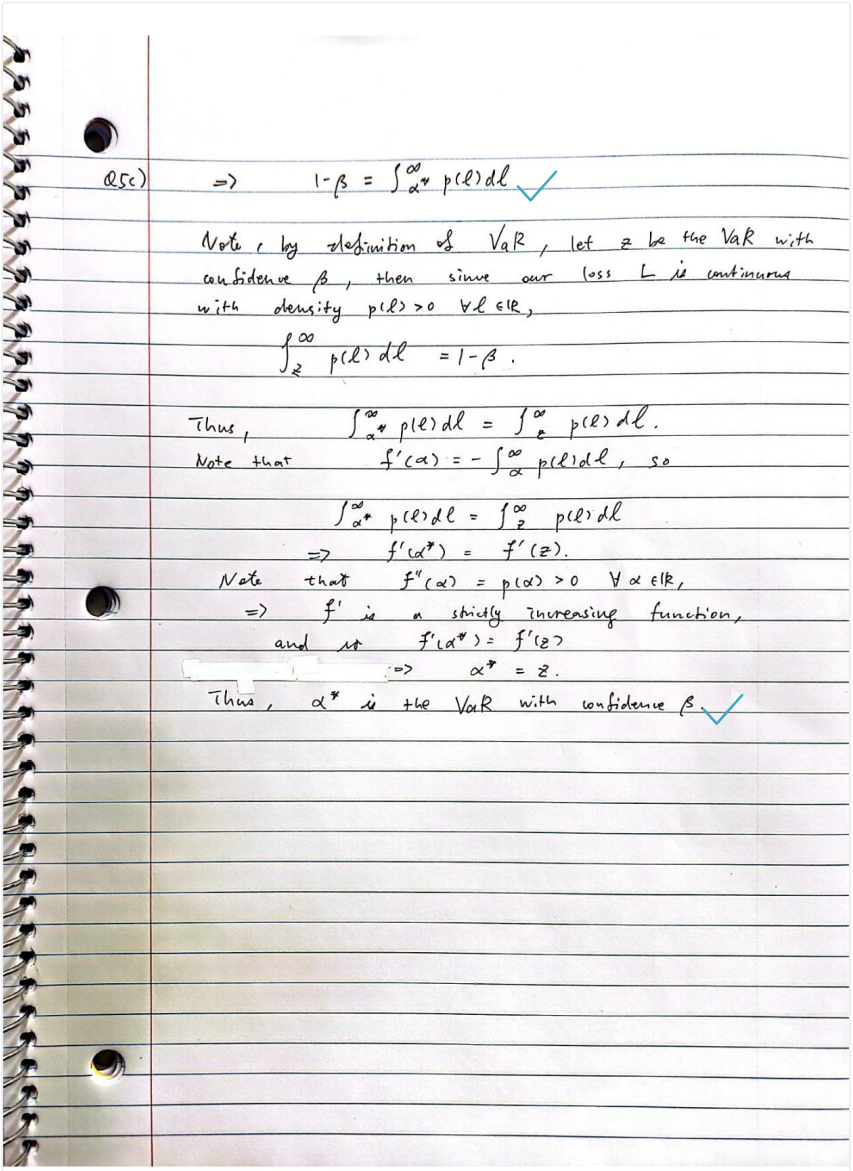
Assume α^* is the solution to (12).

Then by KKT condition (or just 1st derivative test),

we know $g'(\alpha^*) = 0$.

$\Rightarrow g'(\alpha^*) = 1 + \frac{1}{1-\beta} f'(\alpha^*) = 0$

$\Rightarrow 1 + \frac{1}{1-\beta}\left(-\int_{\alpha^*}^\infty p(\ell)\,d\ell\right) = 0$



Q5c) $\Rightarrow \quad 1-\beta = \int_{\alpha^*}^{\infty} p(l)\,dl$ ✓

Note, by definition of VaR, let z be the VaR with confidence β, then since our loss L is continuous with density $p(l) > 0 \quad \forall l \in \mathbb{R}$,

$$\int_z^{\infty} p(l)\,dl = 1-\beta.$$

Thus, $\int_{\alpha^*}^{\infty} p(l)\,dl = \int_z^{\infty} p(l)\,dl.$

Note that $f'(\alpha) = -\int_{\alpha}^{\infty} p(l)\,dl$, so

$$\int_{\alpha^*}^{\infty} p(l)\,dl = \int_z^{\infty} p(l)\,dl$$

$$\Rightarrow \quad f'(\alpha^*) = f'(z).$$

Note that $f''(\alpha) = p(\alpha) > 0 \quad \forall \alpha \in \mathbb{R}$,

$\Rightarrow$ f' is a strictly increasing function,

and as $f'(\alpha^*) = f'(z)$

$\Rightarrow \quad \alpha^* = z.$

Thus, α^* is the VaR with confidence β. ✓

6 Q6

Note that part a and part b are presented in the same file. Also, I have written other functions as well. They will be presented after the main script.

6.1 main script

Listing 10: Q6.m

```
%% Q6a
clearvars
close all

rng('default')

SA0scalar = 20;
SB0scalar = 25;
r = 0.03;
m = 10000;

SA0 = ones(m,1) * SA0scalar;
SB0 = ones(m,1) * SB0scalar;

T = 1/12; % one month
N = 10;

SA = eulerlvf(SA0, T, N, 1, r);
SA = sort(SA); % sort the SA so that the order matches options
% max(SA) -20
% min(SA)- 20
SB = eulerlvf(SB0, T, N, 0, r);
SB = sort(SB);
% max(SB) - 25
% min(SB) -25

SAreturn = (SA - SA0scalar) ./ SA0scalar;
SBreturn = (SB - SB0scalar) ./ SB0scalar;

% make grid, refine the grid 4 times
SAgrid = gridrefine(SA0scalar, SA,4);
SBgrid = gridrefine(SB0scalar, SB,4);

% note that the probability of obtaining duplicate elements are 0
% in fact our choice of random seed make sure that there is no duplicate in
% SAgrid, SBgrid

% save all the rate of return on all the instruments
return_A = zeros(m,5);
return_B = zeros(m,5);
% first column stores the stock
return_A(:,1) = SAreturn;
return_B(:,1) = SBreturn;

Tlist = [0.5,0.5,0.5, 2/3]';
Klist = [SA0scalar, 1.1*SA0scalar, 1.2*SA0scalar, SA0scalar]';
[~,idxA] = intersect(SAgrid, SA);
N = 100;
idscalar = SAgrid == SA0scalar;
[V1old,V2old,V3old,Vppold] = pricer_CNR(SAgrid, N, 1, Tlist, Klist, r);
price_old = [V1old(idscalar),V2old(idscalar),V3old(idscalar),Vppold(idscalar)
    ];
[V1,V2,V3,Vpp] = pricer_CNR(SAgrid, N, 1, Tlist- 1/12, Klist, r);
price_new = [V1(idxA),V2(idxA),V3(idxA),Vpp(idxA)];
return_A(:,2:5) = (price_new - price_old) ./ price_old;
% for stock A, 2nd to 4th columns are call, last column is powerput

```

```
Tlist = [1,1,1,1]';
Klist = [0.8*SB0scalar, 0.9*SB0scalar, SA0scalar, SA0scalar]';
[~,idxB] = intersect(SBgrid, SB);
N = 100;
idscalar = SBgrid == SB0scalar;
[V1old,V2old,V3old,Vppold] = pricer_CNR(SBgrid, N, 0, Tlist, Klist, r);
price_old = [V1old(idscalar),V2old(idscalar),V3old(idscalar),Vppold(idscalar)
    ];
[V1,V2,V3,Vpp] = pricer_CNR(SBgrid, N, 0, Tlist- 1/12, Klist, r);
price_new = [V1(idxB),V2(idxB),V3(idxB),Vpp(idxB)];
return_B(:,2:5) = (price_new - price_old) ./ price_old;
% for stock B, 2nd to 4th columns are put, last column is powerput


% obtain equally weighted portfolio
eq_weg_portfolio = 1/10 * (sum(return_A, 2) + sum(return_B,2));
% histogram
figure(1)
histogram(eq_weg_portfolio,50)
title('histogram of equal allocation')
xlabel('rate of return')
ylabel('frequency')

% Note that the histogram is not normal, very skewed

%% Q6b
allresult = [return_A, return_B]; % 10000 by 10 matrix

loss = -allresult; % so that we consider a min opt problem

betalist = [0.8, 0.85, 0.9,0.95];
rholist = zeros(4,1);
M = 10000;
n =10;

avgloss = mean(loss)';
f = [avgloss;zeros(M+1,1)]; % so that we consider a min opt problem
varlength= n + M + 1; % our variable is listed as [x,y,alpha]'
Aeq = zeros(1, varlength);
Aeq(1,1:n) =1;
beq = 1;

A = zeros(n + 1 + M + M,varlength);
b = zeros(n + 1 + M + M,1);
% first n columns for x_i \geq 0
A(1:n, 1:n) = -eye(n); % need to add negative sign since A x \leq b
b(1:n) = 0;

% next column for CVar constraint, we will update it in the for loop below

% then constraints on the variables y that is larger L^T_i x - alpha
A(n+2:n+1+M,n+1:n+M) = -eye(M);
A(n+2:n+1+M, end) = -1;
A(n+2:n+1+M, 1:n) = loss;
b(n+2:n+1+M) = 0;

% then nonnegative constraint on y
A(n+1+M+1:end, n+1:n+M) = -eye(M);
b(n+1+M+1:end) = 0;

allvaroptlist = zeros(varlength, 4);

comparetable = zeros(5,5);
alphalist = zeros(4,1 );


```

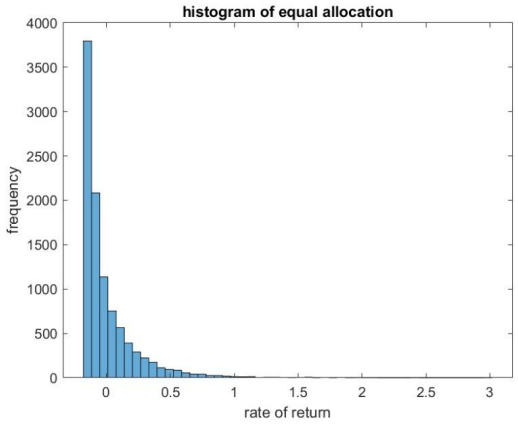
37

non-negativity constraint on y should be handled in the ub and lb parameters of *linprog*. This is probably why your results look off.

```
for i = 1:4
    beta =betalist(i); % for solving min opt problem
    [~,cvar] = dVaRCVaR(eq_weg_portfolio, beta);
    rholist(i) = -cvar;  % consider cvar in terms of loss
    A(n+1, end) = 1;
    A(n+1,n+1:n+M) = 1/(M*(1-beta));
    b(n+1) = rholist(i);
    % solve min problem
    allvar = linprog(f,A,b,Aeq,beq);
    allvaroptlist(:,i) = allvar;
    x = allvar(1:10);
    alpha = allvar(end);

    alphalist(i) = alpha; % var approximation

    optportfolio = allresult * x;

%     figure(i+2)
%     histogram(optportfolio,50)

    optmean = mean(optportfolio);

    comparetable(i+1,1) = optmean;

    for j = 1:4 % for table,
        beta = betalist(j);
        [~,cvar] = dVaRCVaR(optportfolio, beta);
        comparetable(i+1,j+1) = -cvar; % consider cvar in terms of loss
    end
    % note that we can also kind of recover the CVaR by using
    % alpha + 1 / (M*(1-beta)) * sum(y), but we can only recover one CVaR
    % for the particular beta we used in minimization problem
end

comparetable(1,1) = mean(eq_weg_portfolio);
comparetable(1,2:end) = rholist;

format short

comparetable = array2table(comparetable,'VariableNames',...
    {'mean','80% CVaR','85% CVaR','90% CVaR','95% CVaR'},...
    'RowNames',{'equal allocation','optimal wrt 80%', ...
    'optimal wrt 85%','optimal wrt 90%','optimal wrt 95%'})

%% Discussion
% We see that the optimal portfolios (with respect to different beta), have
% very similar mean rate of return, around 0.00363. Note that the rate of
% return we are considering here is for holding the portfolio for one
% month. So the risk-free rate that we should compare to is exp(r/12)-1,
% which is around 0.002503127605795. We observe that the equal allocation
% portfolio has mean rate return 0.00269571202274778, which is very close
% to the risk-free rate. We observe that the mean rate return for optimal
% portfolios are siginificantly better than equal allocation and risk free
% rate.
%
% For optimal portfolio corresponding to beta and rho_beta (i.e CVaR of
% equal allocation portfolio with confidence level beta), the CVaR with
% confidence level beta is very close to rho_beta. This is true because the
% optimization procedure will kind of try to make use of all the given
% budget (i.e. rho_beta) of the CVaR of the optimal portfolio. Note that
% the second line constraint is an upper bound of the CVaR_beta of the
% optimal portfolio. To maximize the return, the optimal portfolio will try
% its best to reach the upper bound, making more rooms for increasing the
% return and hence resulting in a CVaR_beta that is close to rho.
%
% Suppose beta_1 < beta_2. (for instance beta_1 = 0.8, beta_2 = 0.9). We
```

```
% observe that optimal portfolio wrt beta_1 has mean slight lower than that
% wrt beta_2. Moreover, with respect to any beta that we consider, the
% CVaR_beta of the former portfolio are always less than the latter
% portfolio. I suspect that it is true because we actually put more
% constraints on the former portfolio since a lower beta_1 means that we
% have to consider more scenarios when calculating the CVaR. Thus, we put
% constraints on more scenarios and resulting in a slightly "safer"
% portfolio than the latter portfolio in the sense that all CVaR calculated
% are smaller. But it is done at the cost of having a lower mean rate of
% return. (in short, constraint on 80% CVaR impose constraints on worse 20%
% scenarios, while constraint on 95% CVaR only impose constraints on worse
% 5% scenarios. A limit on 80% CVaR will often also limit 95% CVaR to a
% satisfiable degree while the converse is not necessarily the case.
%
% Another possible reason for the above phenomenon might be that the risk
% budget obtained from taking CVaR with respect to different beta on the
% equal allocation portfolio acutally result in a rather uneven
% distribution of risk budget that allows more risk tolearance when we take
% large beta. (i think the last paragraph is more convincing)

function S = gridrefine(S0, Snew,n)
% S0, initial stock value
% Snew, all new stock prices that we want to add
% n, number of refinements on (2) in asg 4
S = [ 0:0.1*S0:0.4*S0, ...
    0.45*S0:0.05*S0:0.8*S0, 0.82*S0 :0.02*S0 :0.9 *S0,...
    0.91*S0 :0.01*S0 :1.1*S0, 1.12*S0 :0.02*S0 :1.2*S0, ...
    1.25*S0 :0.05*S0 :1.6*S0, 1.7*S0 :0.1*S0 :2*S0,...
    2.2*S0, 2.4*S0, 2.8*S0, 3.6*S0, 5*S0, 7.5*S0, 10*S0];
S = S';

for i = 1: n
    S = [S;(S(1:end-1) + S(2:end)) / 2 ];
    S = sort(S);
end

S = [S; Snew];
S = sort(S);

end
```

6.2 result


histogram of equal allocation
frequency
rate of return
4000
3500
3000
2500
2000
1500
1000
500
0
0
0.5
1
1.5
2
2.5
3

```
Optimal solution found.

Optimal solution found.

Optimal solution found.

Optimal solution found.

comparetable =

  5×5 table

                         mean       80% CVaR    85% CVaR    90% CVaR    95% CVaR
                       ---------    --------    --------    --------    --------

    equal allocation    0.0026957    0.14824     0.14982     0.15123     0.15192
    optimal wrt 80%     0.0036289    0.14824     0.14919     0.14992     0.15039
    optimal wrt 85%     0.0036332      0.149     0.15056     0.15095     0.15121
    optimal wrt 90%     0.0036374    0.14972     0.15057     0.15104     0.15131
    optimal wrt 95%     0.0036388    0.14988     0.15072     0.15119     0.15145
```

The CVaR for the optimal allocation strategy should match the CVaR of the equal weight strategy. However, the mean return should be significantly higher than the equal weight strategy, and the return should change w.r.t. beta. -2

6.3 functions scripts

Listing 11: sigmalvf_

```
function sigma_a = sigmalvf_a(S)
%%  local volatility by \alpha ./ sqrt(S)
%
% note that the local volatility function actually
% vectorized output, assume column vector input
%
% inputs: S, stock A price
alpha = 0.85;
sigma_a = alpha./ sqrt(S);
end
```

Listing 12: sigmalvf_

```
function sigma_b = sigmalvf_b(S)
%%  local volatility by \alpha ./ sqrt(S)
%
% note that the local volatility function actually
% vectorized output, assume column vector input
%
% inputs: S, stock B price
alpha = 0.9;
sigma_b = alpha./ sqrt(S);
end
```

Listing 13: eulerlvf.m

```
% Q6a
function Snew = eulerlvf(Sold, T, N, opt, r)
% Sold is the stock price at time 0
% Snew will be the stock price at time T, generated by Euler method
%
% opt = 1 for stock A, else for stock B
% N: number of timesteps taken, integer, so dt = T / N
%
% assume column vector as input
m= length(Sold);
Snew = zeros(m,1);
dt = T/N;
sqrt_dt = sqrt(dt);

if opt == 1
    for i = 1: N
```

```
        phi = randn(m,1);
        Snew = Sold + Sold .*(r * dt + sigmalvf_a(Sold)* sqrt_dt .*phi);
        Sold = Snew;
    end
else
    for i = 1: N
        phi = randn(m,1);
        Snew = Sold + Sold .*(r * dt + sigmalvf_b(Sold)* sqrt_dt .*phi);
        Sold = Snew;
    end
end

end
```

Listing 14: dVaRCVaR.m

```
%% ask, the input is profit or loss?
% after reading lecture carefully, i think it is profit

function [var,cvar] = dVaRCVaR(PL, beta)
% PL: discrete P&L distribution with M independent samples
% assume PL is a column vector
% assume PL is profit, i.e. positive value for profit
% beta could be 95%

M = length(PL);
PL_sorted = sort(PL); % sorts in ascending order.

idx = floor((1-beta)*M);
var = PL_sorted(idx);

cvar = mean(PL_sorted(1:idx));

end
```

Listing 15: pricer_CNR.m

```
%% Q6a, only CNR, for different stock, both powerput and standard call and put

function [V1,V2,V3,Vpp] = pricer_CNR(S, N, opt, Tlist, Klist, r)
% output option value at t =0, for all gridpoints, using CN-Rannacher
%
% if opt = 1, compute call option prices and compute powerput on stock A,
% the order of computation matches the order of Tlist, Klist
% V1, V2, V3 will be option values on the grid S for the 3 call options
%
% if opt is sth else, compute put option prices and compute powerput
% on stock B
% V1, V2, V3 will be option values on the grid S for the 3 put options
%
% in both cases, Vpp are the option values of the grid for the powerput

m = length(S);


%% preprocessing step for simple BS equation
% so that we have positive coefficient discretization
% these will also be used for CN and CN-Rannacher
Smid = S(2:end-1);
Sleft = S(1:end-2);
Sright = S(3:end);

if opt == 1
    sigma = sigmalvf_a(Smid);
else
    sigma = sigmalvf_b(Smid);
end

alphacentral = sigma.^2 .* Smid .^2 ./ ((Smid - Sleft).* (Sright - Sleft)) - ...
    r* Smid ./(Sright - Sleft);
betacentral = sigma.^2 .* Smid .^2 ./ ((Sright - Smid).* (Sright - Sleft)) + ...
    r* Smid ./(Sright - Sleft);
```

```

alphaups = sigma.^2 .* Smid .^2 ./ ((Smid - Sleft).* (Sright - Sleft));
betaups = sigma.^2 .* Smid .^2 ./ ((Sright - Smid).* (Sright - Sleft)) + ...
    r* Smid ./(Sright - Smid);

idx1 = alphacentral <0;
idx2 = betacentral <0;
idx = idx1 | idx2;

alphavec = alphacentral;
betavec = betacentral;
alphavec(idx) = alphaups(idx);
betavec(idx) = betaups(idx);

%% Boundary condition concern:
% initialize payoff, call option payoff
% Note that we adopt just the payoff as the boundary condition. By p.9 in
% Lec16, it is suggested to use S_m for call option. But I think the
% difference is small and adopt S_m - K instead, since S_m is quite large.
% similar approach is taken for put and powerput

%% main for loop
result= zeros(m,4);

for i = 1:4
    %% Fully implicit
    T = Tlist(i);
    Deltatau = T/N;
    % build the matrix M by spdiags
    Bin = Deltatau* [[[-alphavec;0];0], [[r; (alphavec + betavec +r)];0],[[0;0];-
        betavec]];
    M = spdiags(Bin, -1:1, m, m);
    % spy(M) % see the nonzero entries of the matrix M
    % Note that our M is time independent
    I = speye(m);
    [L,U,P] = lu(M+I);
    % increase performance since the LU factorization is only computed once and
    % being reused in each linear system solving
    % to solve (M+I)x = b, y =L\ (P*b), x = U\y

    %% Crank-Nicolson
    Bin = Deltatau/2* [[[-alphavec;0];0], [[r; (alphavec + betavec +r)];0],[[0;0];-
        betavec]];
    Mhat = spdiags(Bin, -1:1, m, m);
    % Note that our Mhat is time independent
    I = speye(m);
    [L1,U1,P1] = lu(Mhat+I);

    %% CN-Rannacher
    K = Klist(i);
    Vold = zeros(m,1);
    Vnew = Vold;
    % initialize payoff, call option payoff
    if opt == 1
        if i < 4
            Vold = max(S-K,0);
        else
            Vold = max(K-S,0).^3;
        end
    else
        if i < 4
            Vold = max(K-S,0);
        else
            Vold =  max(K-S,0).^3;
        end
    end

    % two steps of fully implicit
    for n = 1 : 2
        y = L\(P*Vold);
        Vnew = U\y;
        Vold = Vnew;
```

```
    end

    % use CN after that
    for n = 1 : N-2
        y = L1\(P1*(I-Mhat)*Vold);
        Vnew = U1\y;
        Vold = Vnew;
    end
    result(:,i) = Vnew;
end

V1 = result(:,1);
V2 = result(:,2);
V3 = result(:,3);
Vpp =result(:,4);


end
```