

CS 676 Assignment 4

Ying Kit Hui (Graduate student)

April 4, 2022

Listings

1	Q1a.m	1
2	Q1b.m	6
3	Q2a.m	12
4	Q2b.m	17
5	CNRall.m	22
6	Q3a.m	25
7	callpricer_CNR.m	27
8	myfun.m	29
9	Q3d.m	30
10	Q6.m	36
11	sigmalvf_a.m	41
12	sigmalvf_b.m	41
13	eulerlvf.m	41
14	dVaRCVaR.m	42
15	pricer_CNR.m	42

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

```
1 %% Q1a
2
3 clearvars
4 close all
5 % parameters initialization
6 S0 = 76;
7 T = 1;
8 Deltatau = T/25;
9 alpha = 15;
10 r = 0.03;
11 K = S0;
12
13 % check S
14 S = [ 0:0.1*S0:0.4*S0, ...
15     0.45*S0:0.05*S0:0.8*S0, 0.82*S0 :0.02*S0 :0.9 *S0, ...
16     0.91*S0 :0.01*S0 :1.1*S0, 1.12*S0 :0.02*S0 :1.2*S0, ...
17     1.25*S0 :0.05*S0 :1.6*S0, 1.7*S0 :0.1*S0 :2*S0, ...
18     2.2*S0, 2.4*S0, 2.8*S0, 3.6*S0, 5*S0, 7.5*S0, 10*S0];
19 S = S';
20 Scell = {};
21 Scell{1} = S;
22
23 Vold = max(K-S,0).^3;
24 figure(1)
```

```

25 plot(S, Vold)
26 title('payoff graph, power put')
27 xlabel('asset value S')
28 ylabel('payoff')
29
30 Deltataulist = zeros(5,1);
31 Deltataulist(1) = Deltatau;
32 % 4 refinements
33 for i = 1:4
34     Sold = Scell{i};
35     Snew = [Sold;(Sold(1:end-1) + Sold(2:end)) / 2 ];
36     Snew = sort(Snew);
37     Scell{i+1} = Snew;
38     Deltataulist(i+1) = Deltatau/(2^i);
39 end
40
41 % make convergence table
42 Fimplicittable = zeros(length(Deltataulist),5);
43 CNTable = zeros(length(Deltataulist),5);
44 CNRantable = zeros(length(Deltataulist),5);
45
46 for i = 1:5
47     % obtain option value at t=0, S = S0 using the corresponding grid and
48     % time step
49     [V0FL, V0CN,V0CNRan] = powerputpricer(Scell{i}, Deltataulist(i), S0, T, K
50         , r);
51
52     % nodes number
53     Fimplicittable(i,1) = size(Scell{i},1);
54     CNTable(i,1) = size(Scell{i},1);
55     CNRantable(i,1) = size(Scell{i},1);
56
57     % timesteps number
58     Fimplicittable(i,2) = 25 * 2^(i-1);
59     CNTable(i,2) = 25 * 2^(i-1);
59     CNRantable(i,2) = 25 * 2^(i-1);
60
61     % option Value
62     Fimplicittable(i,3) = V0FL;
63     CNTable(i,3) = V0CN;
64     CNRantable(i,3) = V0CNRan;
65
66     % change in option value
67     if i > 1
68         Fimplicittable(i,4) = Fimplicittable(i,3) - Fimplicittable(i-1,3);
69         CNTable(i,4) = CNTable(i,3) - CNTable(i-1,3);
70         CNRantable(i,4) = CNRantable(i,3) - CNRantable(i-1,3);
71     end
72
73     % Ratio
74     if i > 2
75         Fimplicittable(i,5) = Fimplicittable(i-1,4)/Fimplicittable(i,4);
76         CNTable(i,5) = CNTable(i-1,4)/CNTable(i,4);
77         CNRantable(i,5) = CNRantable(i-1,4)/CNRantable(i,4);
78     end
79 end
80
81 Fimplicittable = array2table(Fimplicittable, 'VariableNames', ...
82     {'Nodes','timesteps','Value','Change','Ratio'})
83 CNTable = array2table(CNTable, 'VariableNames', ...
84     {'Nodes','timesteps','Value','Change','Ratio'})
85 CNRantable = array2table(CNRantable, 'VariableNames', ...
86     {'Nodes','timesteps','Value','Change','Ratio'})
87
88 %% Q1a Discussion

```

```

89 % Note that fully implicit method has convergence rate O(\Delta tau, (\Delta
90 S)^2)
91 % and hence it does not satisfy (4) in our question. Thus we don't expect
92 % the ratio for the convergence table of fully implicit method to be 4. In
93 % fact, we observe that such ratio is around 3, reflecting that the
94 % convergence rate is slightly better than first order (since we have
95 % O((\Delta S)^2)) but not as good as second order (since we have O(\Delta
96 % tau)).
97 %
98 % Note that Crank-Nicolson method has convergence rate O((\Delta tau)^2,
99 % (\Delta S)^2). Hence, we expect the ratio for the convergence table of CN
100 % method to be 4. We observe that such ratio is indeed roughly 4,
101 % consistent with the theory regarding the rate of convergence.
102 % Note that generally non-smooth payoff will cause oscillations in
103 % solution, resulting in slow convergence of CN method. However, in our
104 % case, the payoff function of the power put is quite smooth since we are
105 % taking cubic power. The smoothness of the payoff can also be seen in
106 % figure 1. At least, the payoff is continuous and the fact that K = S0 and
107 % we have a node at S0 = K means that no smoothing is required and we
108 % expect the usual second order convergence.
109 %
110 % For CN-Rannacher, the purpose of it is to do some smoothing so that
111 % second order convergence is attained even for nonsmooth payoff. But in
112 % our case, the payoff is quite smooth. But it does not prohibit
113 % CN-Rannacher to achieve second order convergence, as demonstrated in the
114 % Ratio column of its convergence table being around 4.
115
116 function [V0fullyimplicit, V0CN,V0CNRan] = powerputpricer(S, Deltatau, S0, T,
117 K, r)
118 % output option value at t =0, S =S0, using fully implicit, Crank Nicolson,
119 % and CN-Rannacher
120
121 %%% Fully implicit
122 %%% only compute initial option value
123 %%% can easily modified to obtain options values at each grid points
124 m = length(S);
125 Vold = zeros(m,1);
126 Vnew = Vold;
127 N = T / Deltatau;
128 % initialize payoff
129 Vold = max(K-S,0).^3;
130
131 % preprocessing step for simple BS equation
132 % so that we have positive coefficient discretization
133 % these will also be used for CN and CN-Rannacher
134 Smid = S(2:end-1);
135 Sleft = S(1:end-2);
136 Sright = S(3:end);
137 sigma = sigmalvf(Smid);
138 alphacentral = sigma.^2 .* Smid .^2 ./ ((Smid - Sleft).* (Sright - Sleft)) -
139 ...
140     r* Smid ./ (Sright - Sleft);
141 betacentral = sigma.^2 .* Smid .^2 ./ ((Sright - Smid).* (Sright - Sleft)) +
142 ...
143     r* Smid ./ (Sright - Smid);
144
145 idx1 = alphacentral <0;
146 idx2 = betacentral <0;
147 idx = idx1 | idx2;
148
149 alphavec = alphacentral;
150 betavec = betacentral;

```

```

151 alphavec(idx) = alphaups(idx);
152 betavec(idx) = betaups(idx);
153
154 % build the matrix M by spdiags
155 Bin = Deltatau* [[[ -alphavec; 0]; 0], [[r; (alphavec + betavec + r)]; 0], [[0; 0]; -betavec]];
156 M = spdiags(Bin, -1:1, m, m);
157 % spy(M) % see the nonzero entries of the matrix M
158 % Note that our M is time independent
159 I = speye(m);
160 [L, U, P] = lu(M + I);
161 % increase performance since the LU factorization is only computed once and
162 % being reused in each linear system solving
163 % to solve (M+I)x = b, y = L \ (P*b), x = U \ y
164
165 for n = 1 : N
166     y = L \ (P * Vold);
167     Vnew = U \ y;
168     Vold = Vnew;
169 end
170 S0id = S0 == S;
171 V0fullyimplicit = Vnew(S0id);
172
173
174 %%% Crank-Nicolson
175 %%% only compute initial option value
176 %%% can easily modified to obtain options values at each grid points
177 Vold = zeros(m, 1);
178 Vnew = Vold;
179 % initialize payoff
180 Vold = max(K - S, 0).^3;
181
182 Bin = Deltatau / 2 * [[[ -alphavec; 0]; 0], [[r; (alphavec + betavec + r)]; 0], [[0; 0]; -betavec]];
183 Mhat = spdiags(Bin, -1:1, m, m);
184 % Note that our Mhat is time independent
185 I = speye(m);
186 [L1, U1, P1] = lu(Mhat + I);
187
188
189 for n = 1 : N
190     y = L1 \ (P1 * (I - Mhat) * Vold);
191     Vnew = U1 \ y;
192     Vold = Vnew;
193 end
194 V0CN = Vnew(S0id);
195
196 %%% CN-Rannacher
197 Vold = zeros(m, 1);
198 Vnew = Vold;
199 % initialize payoff
200 Vold = max(K - S, 0).^3;
201
202 % two steps of fully implicit
203 for n = 1 : 2
204     y = L \ (P * Vold);
205     Vnew = U \ y;
206     Vold = Vnew;
207 end
208
209 % use CN after that
210 for n = 1 : N - 2
211     y = L1 \ (P1 * (I - Mhat) * Vold);
212     Vnew = U1 \ y;
213     Vold = Vnew;
214 end

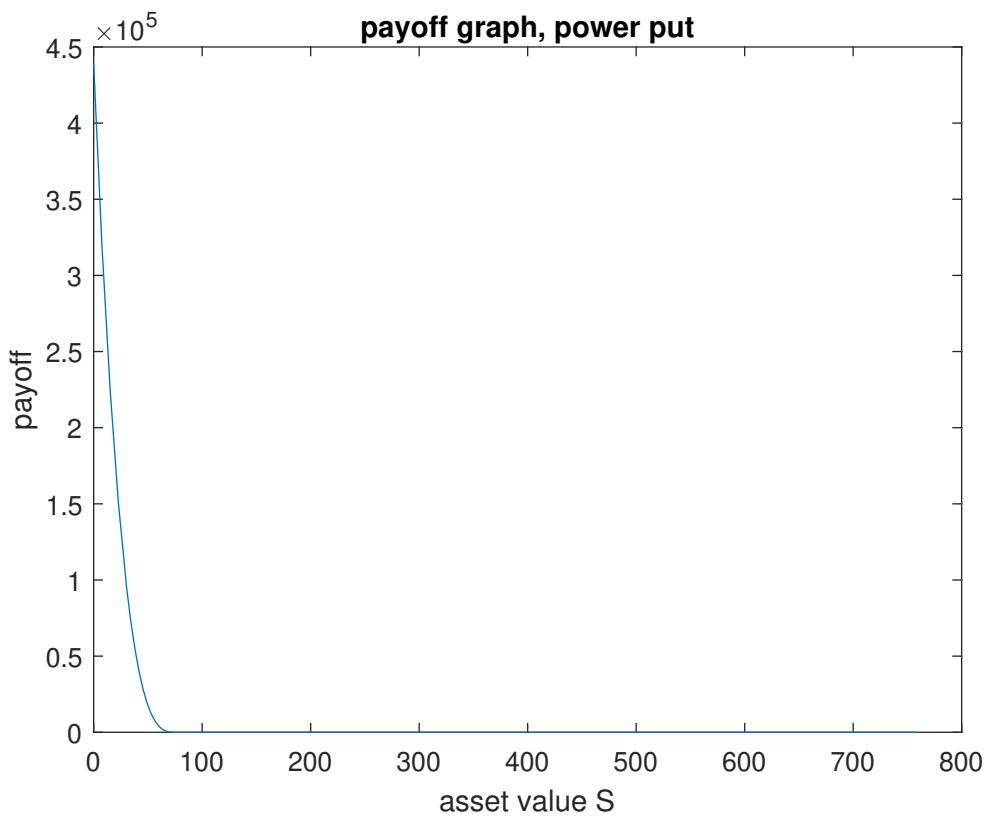
```

```

215 V0CNRan = Vnew(S0id);
216
217 end
218
219 function out = sigmalvf(S)
220 % note that the local volatility function actually is independent of t
221 % vectorized output, assume column vector input
222 alpha = 15;
223 S0 = 76;
224 out = alpha./ (max(S0 + (S-S0)./(2*S0), 10^(-8)));
225

```

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

```

1 %% Q1b
2 close all
3 % parameters initialization
4 S0 = 76;
5 T = 1;
6 Deltatau = T/25;
7 alpha = 15;
8 r = 0.03;
9 K = S0;
10
11 % check S
12 S = [ 0:0.1*S0:0.4*S0, ...
13     0.45*S0:0.05*S0:0.8*S0, 0.82*S0 :0.02*S0 :0.9 *S0, ...
14     0.91*S0 :0.01*S0 :1.1*S0, 1.12*S0 :0.02*S0 :1.2*S0, ...
15     1.25*S0 :0.05*S0 :1.6*S0, 1.7*S0 :0.1*S0 :2*S0, ...
16     2.2*S0, 2.4*S0, 2.8*S0, 3.6*S0, 5*S0, 7.5*S0, 10*S0];
17 S = S';
18 Scell = {};
19 Scell{1} = S;
20
21 Deltataulist = zeros(5,1);
22 Deltataulist(1) = Deltatau;
23 % 4 refinements
24 for i = 1:4
25     Sold = Scell{i};
26     Snew = [Sold;(Sold(1:end-1) + Sold(2:end)) / 2 ];
27     Snew = sort(Snew);

```

```

28     Scell{i+1} = Snew;
29     Deltataulist(i+1) = Deltatau/(2^i);
30 end
31
32 % USE the finest grid for CN-Rannacher
33 S = Scell{end};
34 Deltatau = Deltataulist(end);
35
36
37 % preprocessing step for simple BS equation
38 % so that we have positive coefficient discretization
39 % these will also be used for CN and CN-Rannacher
40 m = length(S);
41 N = T / Deltatau;
42
43 Smid = S(2:end-1);
44 Sleff = S(1:end-2);
45 Sright = S(3:end);
46 sigma = sigmalvf(Smid);
47 alphacentral = sigma.^2 .* Smid.^2 ./ ((Smid - Sleff).* (Sright - Sleff)) - ...
48     r* Smid ./(Sright - Sleff);
49 betacentral = sigma.^2 .* Smid.^2 ./ ((Sright - Smid).* (Sright - Sleff)) + ...
50     r* Smid ./(Sright - Sleff);
51
52 alphaups = sigma.^2 .* Smid.^2 ./ ((Smid - Sleff).* (Sright - Sleff));
53 betaups = sigma.^2 .* Smid.^2 ./ ((Sright - Smid).* (Sright - Sleff)) + ...
54     r* Smid ./(Sright - Smid);
55
56 idx1 = alphacentral <0;
57 idx2 = betacentral <0;
58 idx = idx1 | idx2;
59
60 alphavec = alphacentral;
61 betavec = betacentral;
62 alphavec(idx) = alphaups(idx);
63 betavec(idx) = betaups(idx);
64
65 % build the matrix M by spdiags
66 Bin = Deltatau* [[[ -alphavec; 0]; 0], [[r; (alphavec + betavec +r)]; 0], [[0; 0]; -betavec
67     ]];
68 M = spdiags(Bin, -1:1, m, m);
69 % spy(M) % see the nonzero entries of the matrix M
70 % Note that our M is time independent
71 I = speye(m);
72 [L,U,P] = lu(M+I);
73 % increase performance since the LU factorization is only computed once and
74 % being reused in each linear system solving
75 % to solve (M+I)x = b, y = L\ (P*b), x = U\y
76
77 Bin = Deltatau/2* [[[ -alphavec; 0]; 0], [[r; (alphavec + betavec +r)]; 0], [[0; 0]; -betavec
78     ]];
79 Mhat = spdiags(Bin, -1:1, m, m);
80 % Note that our Mhat is time independent
81 I = speye(m);
82 [L1,U1,P1] = lu(Mhat+I);
83
84 %% CN-Rannacher
85 Vold = zeros(m,1);
86 Vnew = Vold;
87 % initialize payoff
88 Vold = max(K-S,0).^3;
89
90 % two steps of fully implicit
91 for n = 1 : 2
92     y = L\ (P*Vold);
93     Vnew = U\y;
94     Vold = Vnew;
95 end
96
97 % use CN after that
98 Vbeforehalf = zeros(m,1);
99 Vafterhalf = zeros(m,1);

```

```

98 Vathalf = zeros(m,1);
99 Vbefore0 = zeros(m,1);
100 for n = 1 : N -2
101     y = L1\ (P1*(I-Mhat)*Vold);
102     Vnew = U1\y;
103     Vold = Vnew;
104     if n == N/2 -3 % save the data at the timestep just after t= T/2,
105         % after in terms of t
106         Vbeforehalf = Vold;
107     end
108     if n == N/2 -1 % save the data at the timestep just before t = T/2,
109         % before in terms of t
110         Vafterhalf = Vold;
111     end
112     if n == N/2 - 2 % save the data at the t= T/2
113         Vathalf = Vold;
114     end
115     if n == N-2 -1 % save the data at the timestep just before t = 0
116         Vbefore0 = Vold;
117     end
118
119 end
120
121 % index for S in the range = [0.5 S0, 1.5 S0]
122 idx = (S <= 1.5*S0) & (S >= 0.5 * S0);
123 idx = find(idx);
124 index1 = idx(1);
125 index2 = idx(end);
126
127 Vat0 = Vold; % the data at t = 0
128
129 % Central difference for delta and gamma at t = T/2
130 deltaxright = S(index1 + 1: index2 + 1) - S(index1:index2);
131 deltaxleft = S(index1:index2) - S(index1 - 1:index2-1);
132 deltafrighthalf = Vathalf(index1 + 1: index2 + 1) - Vathalf(index1:index2);
133 deltafleighthalf = Vathalf(index1:index2) - Vathalf(index1 - 1:index2-1);
134 gammaaathalf = (deltafrighthalf ./ deltaxright - deltafleighthalf./deltaxleft) ...
135     ./ ((deltaxright + deltaxleft)/2);
136 % not sure about this, I think this might improve the order of delta a bit
137 deltaaathalf = (Vathalf(index1+1: index2+1)-Vathalf(index1-1:index2-1))...
138     ./ (deltaxright + deltaxleft) - gammaaathalf /2 .* (deltaxright-deltaxleft);
139
140 % Central difference for delta and gamma at t = 0
141
142 deltafrighth0 = Vat0(index1 + 1: index2 + 1) - Vat0(index1:index2);
143 deltafleighth0 = Vat0(index1:index2) - Vat0(index1 - 1:index2-1);
144 gammaaat0 = (deltafrighth0 ./ deltaxright - deltafleighth0./deltaxleft) ...
145     ./ ((deltaxright + deltaxleft)/2);
146 % not sure about this, I think this might improve the order of delta a bit
147 deltaaat0 = (Vat0(index1+1: index2+1)-Vat0(index1-1:index2-1))...
148     ./ (deltaxright + deltaxleft) - gammaaat0 /2 .* (deltaxright-deltaxleft);
149
150 figure(1)
151 plot(S(idx), Vathalf(idx))
152 hold on
153 plot(S(idx), Vat0(idx))
154 title('Option value')
155 xlabel('asset value S')
156 legend('t= T/2', 't=0')
157 hold off
158
159 figure(2)
160 plot(S(idx), deltaaathalf)
161 hold on
162 plot(S(idx), deltaaat0)
163 title('Delta')
164 xlabel('asset value S')
165 legend('t= T/2', 't=0')
166 hold off
167
168 figure(3)
169 plot(S(idx), gammaaathalf)

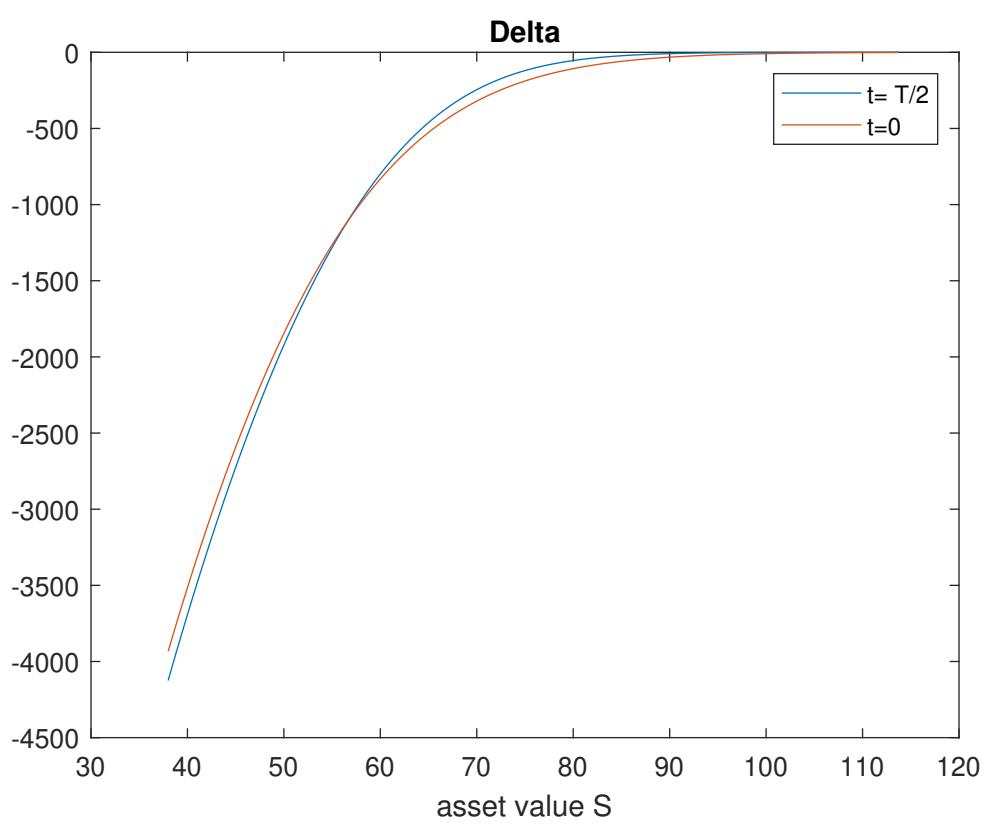
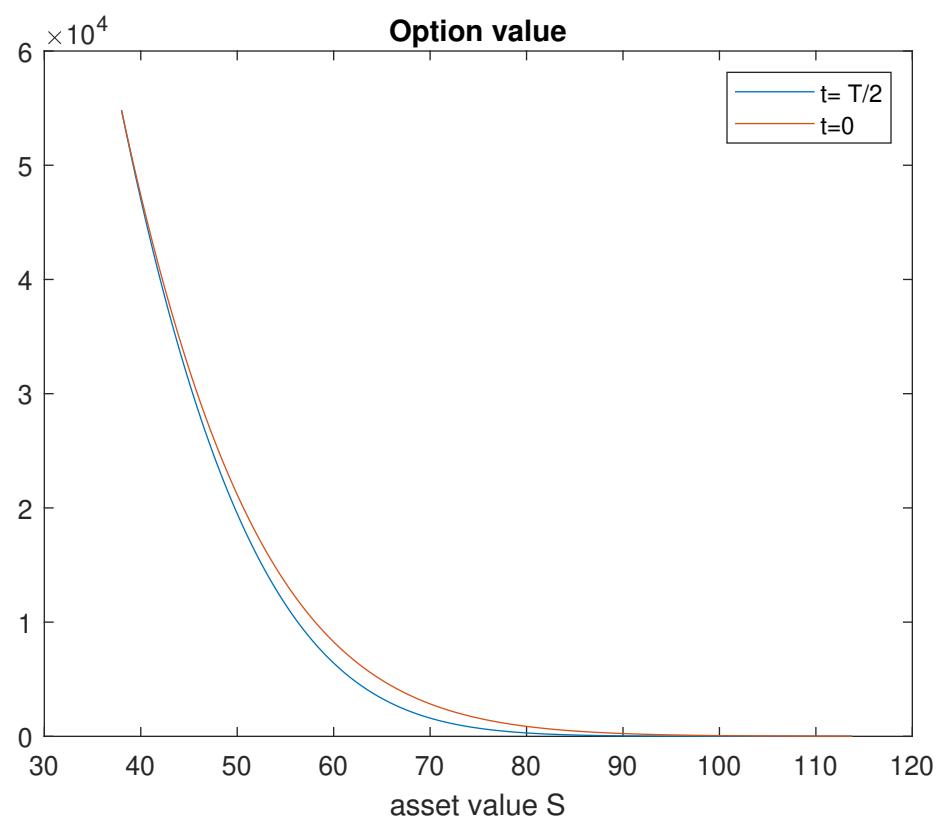
```

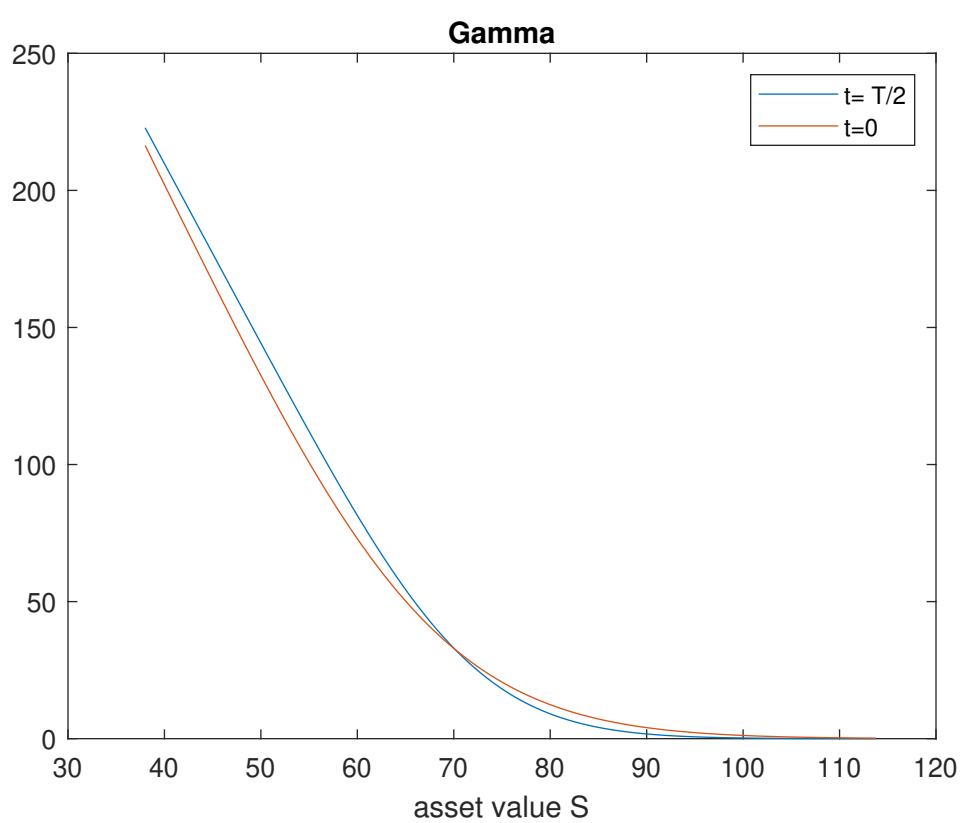
```

170 hold on
171 plot(S(idx), gammaat0)
172 title('Gamma')
173 xlabel('asset value S')
174 legend('t= T/2', 't=0')
175 hold off
176
177 save Q1b.mat
178
179 %% Discussion
180 % For figure(1) option value plot, we observe that the option values at t =
181 % 0 are larger than that at t = T/2, for the same asset value S. This makes
182 % sense since the Greek Theta is generally negative for long call or long
183 % put, reflecting the fact that there are time values in options and as
184 % time goes, this time value decays. Note particularly that the gap between
185 % two curves is the most significant (relatively speaking) when S is near
186 % the strike, this is because at the far left or the far right, the time
187 % value of the option is almost negligible. On the far right, the value of
188 % the option is almost 0, and we expect 0 payoff. On the far left, it might
189 % best to exercise right now. (not sure)
190 %
191 % For figure(2) delta plot, we see a more complicated relationship. First
192 % of all, the delta are all nonpositive. This is because as S increases,
193 % our expected payoff will decrease, since we are holding power put. Note
194 % that two curves intersect, let's denote the intersection by x* (around
195 % 56). Also, we consider absolute value of delta in the following
196 % comparison of size. So we say size of Delta 1 is larger than that of
197 % Delta 2, we actually mean that the absolute value of Delta 1 is larger
198 % than absolute value of Delta 2. Before x*, the size delta for t = 0 is
199 % less than that of t = T/2. This makes sense since at T/2, any increase in
200 % asset value at region that already has low asset value (i.e. high payoff)
201 % will very likely result in a decrease of final payoff, whereas at t = 0,
202 % that phenomenon is less severe. (not sure about this interpretation)
203 %
204 % For figure(3) gamma plot, we see a more complicated relationship. First
205 % of all, the gamma are all nonpositive, indicating that the option value
206 % is convex and delta is increasing function of S. Note
207 % that two curves intersect, let's denote the intersection by x** (around
208 % 70). Before x**, option value for t = T/2 is more convex. After x**,
209 % option value for t = 0 is more convex.
210 %
211 % I am not sure how can I intuitively or financially interpret these plots.
212
213 function out = sigmalvf(S)
214 % note that the local volatility function actually is independent of t
215 % vectorized output, assume column vector input
216 alpha = 15;
217 S0 = 76;
218 out = alpha./ (max(S0 + (S - S0). / (2*S0), 10^(-8)));
219 end

```

1.2.1 result





2 Q2

2.1 Q2a

Listing 3: Q2a.m

```
1 %% Q2a, variable timestepping and constant timestepping
2
3 clearvars
4 close all
5 %% parameters initialization
6 S0 = 76;
7 T = 1;
8 Deltatau = T/25; % initial time step
9 dnorm = 0.1;
10 alpha = 15;
11 r = 0.03;
12 K = S0;
13
14 S = [ 0:0.1*S0:0.4*S0, ...
15     0.45*S0:0.05*S0:0.8*S0, 0.82*S0 :0.02*S0 :0.9 *S0, ...
16     0.91*S0 :0.01*S0 :1.1*S0, 1.12*S0 :0.02*S0 :1.2*S0, ...
17     1.25*S0 :0.05*S0 :1.6*S0, 1.7*S0 :0.1*S0 :2*S0, ...
18     2.2*S0, 2.4*S0, 2.8*S0, 3.6*S0, 5*S0, 7.5*S0, 10*S0];
19 S = S';
20
21 %% Now do refinements
22 Scell = {};
23 Scell{1} = S;
24 % list of initial timestep
25 Deltataulist = zeros(5,1);
26 Deltataulist(1) = Deltatau;
27 dnormlist = zeros(5,1);
28 dnormlist(1) = dnorm;
29 % 4 refinements
30 for i = 1:4
31     Sold = Scell{i};
32     Snew = [Sold;(Sold(1:end-1) + Sold(2:end)) / 2 ];
33     Snew = sort(Snew);
34     Scell{i+1} = Snew;
35     Deltataulist(i+1) = Deltatau/(4^i);
36     dnormlist(i+1) = dnorm/(2^i);
37 end
38
39 % make convergence table
40 CNRvartable = zeros(length(Deltataulist),7);
41 CNRconstanttable = zeros(length(Deltataulist),7);
42 vartimetotal = 0;
43 contimetotal = 0;
44 for i = 1:5
45     % obtain option value at t=0, S = S0 using the corresponding grid and
46     % time step
47     Deltatau = Deltataulist(i);
48     S = Scell{i};
49     dnorm = dnormlist(i);
50
51
52     vartime = tic;
53     [V0var, steptotalvar, countervar] = CNR(S, Deltatau, S0, T, K, r,dnorm,1)
54         ;
55     vartime = toc(vartime);
56     vartimetotal = vartimetotal + vartime;
57
58     contime = tic;
59     [V0con, steptotalcon, countercon] = CNR(S, Deltatau, S0, T, K, r,dnorm,0)
60         ;
61     contime = toc(contime);
```

```

60     contimetotal = contimetotal + contime;
61
62     % nodes number
63     CNRvartable(i,1) = size(Scell{i},1);
64     CNRconstanttable(i,1) = size(Scell{i},1);
65
66     % timesteps number
67     CNRvartable(i,2) = T/ Deltatau;
68     CNRconstanttable(i,2) = T/ Deltatau;
69
70     % option Value
71     CNRvartable(i,3) = V0var;
72     CNRconstanttable(i,3) = V0con;
73     % change in option value
74     if i > 1
75         CNRvartable(i,4) = CNRvartable(i,3) - CNRvartable(i-1,3);
76         CNRconstanttable(i,4) = CNRconstanttable(i,3) - CNRconstanttable(i
77             -1,3);
78     end
79
80     % Ratio
81     if i > 2
82         CNRvartable(i,5) = CNRvartable(i-1,4)/CNRvartable(i,4);
83         CNRconstanttable(i,5) = CNRconstanttable(i-1,4)/CNRconstanttable(i,4)
84             ;
85     end
86
87     % counter
88     CNRvartable(i,6) = countervar;
89     CNRconstanttable(i,6) = countercon;
90
91     % total iterative steps
92     CNRvartable(i,7) = steptotalvar;
93     CNRconstanttable(i,7) = steptotalcon;
94
95 end
96
97 CNRvartable = array2table(CNRvartable, 'VariableNames', ...
98     {'Nodes','initial timestep','Value','Change','Ratio','# timestepping','
99         iterative steps' })
100 vartimetotal
101
102 CNRconstanttable = array2table(CNRconstanttable, 'VariableNames', ...
103     {'Nodes','timesteps','Value','Change','Ratio','# timestepping','iterative
104         steps' })
105 contimetotal
106
107 %% Discussion
108 % For both methods, we refine the grids as in Q1, doubling the number of
109 % intervals.
110 %
111 % For CN-Rannacher with variable timestepping, at each refinement, we
112 % reduce the initial timestep by a factor of 4, and reduce dnorm by a
113 % factor of 2.
114 %
115 % For CN-Rannacher with constant timestepping, at each refinement, we reduce
116 % the constant timestepping by a factor of 4.
117 %
118 % We see that CN-Rannacher with variable timestepping does give us
119 % quardartic convergence, since the ratio in CNRvartable is around 4.
120 % This is expected.
121 %
122 % Note that for the convergence table of CN-Rannacher with constant
123 % timestepping, at each refinement we are dividing the constant timestep by
124 % 4. This might explain why we also observe quardartic convergence in such
125 % case. I suspect that the (convergence) error might only depend slightly

```

```

122 % on \Delta S, so that the number of nodes does not greatly affect the
123 % convergence rate. If this is true, this explains why we obtain quardartic
124 % convergence. Another reason for that might be the powerput payoff is
125 % quite smooth so that quardartic convergence can still be achieved.
126 %
127 % Note that if we change the refinement to timestep /2, still observe the
128 % quardartic convergence... (WHY???) Perhaps it is because the powerput
129 % payoff is quite smooth. I have experimented my codes on other examples in
130 % the lecture slides and obtain similar results.
131 %
132 % Lastly, in the setting we consider here, the runtime of CNR-variable
133 % timestepping is less than that of CNR-constant timestepping.
134 %
135 function [V0, steptotal, counter] = CNR(S, Deltatau, S0, T, K, r,dnorm,opt)
136 %% Description
137 %% Inputs:
138 % opt = 1 for variable time, else for constant timestepping
139 % when opt = 1, Deltatau is the initial timestep
140 % when opt is else, Deltatau is the constant timestep
141 %% Outputs:
142 % V0, option value at S0 at t = 0
143 % steptotal, the total number of steps used for iterative solvers
144 % counter, the total time of choosing variable timestep; counter is just
145 % T/Deltatau when constant timestepping
146 %
147 %% preprocessing step for simple BS equation
148 % so that we have positive coefficient discretization
149 % these will also be used for CN and CN-Rannacher
150 Smid = S(2:end-1);
151 Sleft = S(1:end-2);
152 Sright = S(3:end);
153 sigma = sigmalvf(Smid);
154 alphacentral = sigma.^2 .* Smid .^2 ./ ((Smid - Sleft).* (Sright - Sleft)) -
...
155 r* Smid ./(Sright - Sleft);
156 betacentral = sigma.^2 .* Smid .^2 ./ ((Sright - Smid).* (Sright - Sleft)) +
...
157 r* Smid ./(Sright - Sleft);
158
159 alphaups = sigma.^2 .* Smid .^2 ./ ((Smid - Sleft).* (Sright - Sleft));
160 betaups = sigma.^2 .* Smid .^2 ./ ((Sright - Smid).* (Sright - Sleft)) +
...
161 r* Smid ./(Sright - Smid);
162
163 idx1 = alphacentral <0;
164 idx2 = betacentral <0;
165 idx = idx1 | idx2;
166
167 alphavec = alphacentral;
168 betavec = betacentral;
169 alphavec(idx) = alphaups(idx);
170 betavec(idx) = betaups(idx);
171 %% Some common data needed for constant and variable timestepping
172 m = length(S);
173 Vold = zeros(m,1);
174 Vnew = Vold;
175 N = T / Deltatau;
176 % initialize payoff
177 Vold = max(K-S,0).^3;
178
179 % penalty method data, tolerance and Large (penalty coefficient)
180 tol = 10^(-6); % as suggested by piazza
181 Large = 1/tol;
182
183 % Vstar, payoff at the gridpoints, for American option payoff constraints
184 Vstar = max(K-S,0).^3;
185

```

```

186 Bin = [[[−alphavec;0];0], [[r; (alphavec + betavec +r)];0],[[0;0];−betavec]];
187 % generic bin, used repeatedly below
188
189 steptotal = 0;
190
191 %% opt = 1, Use variable timestep
192 %%% also obtain number of steps to reach tol
193 %%% also need to update the Mhat and M
194 if opt == 1
195     dtold = Deltatau;
196     dtnew = 0;
197     currenttau = 0;
198     D = 1; % p.32 in lec 18, D = 1 fine for dollars
199     counter = 0;
200     % CN-Rannacher, variable time
201     while currenttau < T
202         counter = counter +1;
203
204         if currenttau + dtold >= T
205             dtold = T − currenttau;
206             currenttau = T;
207         else
208             currenttau = currenttau + dtold;
209         end
210
211         if counter <= 2 % first two steps are fully implicit
212             % build M, using correct \Delta tau
213             M = spdiags(dtold*Bin, −1:1, m, m);
214             [Vnew,step]= iterativesolver(Vold, Vstar, tol, Large, M,m,0);
215         else
216             % build Mhat, using correct \Delta tau
217             Mhat = spdiags((dtold/2) *Bin, −1:1, m, m);
218             [Vnew,step]= iterativesolver(Vold, Vstar, tol, Large, Mhat,m,1);
219         end
220
221         steptotal = steptotal + step;
222
223         % choose timestep at the end
224         % assume Vold is still storing previous data, not current data
225         inter = max(D,max(abs(Vnew), abs(Vold)));
226         maxrelchange = max(abs(Vnew − Vold)./inter);
227         dtnew = (dnorm/maxrelchange) * dtold;
228         dtold = dtnew;
229         % update V
230         Vold = Vnew;
231     end
232
233 S0id = S0 == S;
234 V0 = Vnew(S0id);
235 else
236     %% opt is not 1, use constant timestepping
237     Mhat = spdiags((Deltatau/2)*Bin, −1:1, m, m);
238     M = spdiags(Deltatau*Bin, −1:1, m, m);
239     % two steps of Fully Implicit
240     for i = 1:2
241         [Vnew,step]= iterativesolver(Vold, Vstar, tol, Large, M,m,0);
242         steptotal = steptotal + step;
243         Vold = Vnew;
244     end
245
246     for i = 1: N−2
247         [Vnew,step]= iterativesolver(Vold, Vstar, tol, Large, Mhat,m,1);
248         steptotal = steptotal + step;
249         Vold = Vnew;
250     end
251     S0id = S0 == S;

```

```

252     V0 = Vnew(S0id);
253     counter = N;
254 end
255
256 end
257
258 function [Vnew,step]= iterativesolver(Vold, Vstar, tol, Large, genM,m,opt)
259 % opt = 1 for CN, else fully implicit
260 % genM= Mhat when opt = 1 (i.e. CN), o/w genM = M (i.e. fully implicit)
261
262 error = 10000; % set the error to be some large number
263 step = 0;
264 Viterold = Vold;
265 I = speye(m);
266 while error >= tol
267     % Vold is V^n
268     % Viterold is (V^{n+1})^k, Viternew will be (V^{n+1})^{(k+1)}
269     largediagonal = zeros(m,1);
270     largeidx = Viterold < Vstar;
271     largediagonal(largeidx) = Large;
272     P = spdiags(largediagonal, 0,m,m);
273     if opt == 1 % CN
274         y = (I - genM)*Vold + P* Vstar;
275         Viternew = (I + genM + P) \ y;
276     else % fully implicit
277         y = Vold + P* Vstar;
278         Viternew = (I + genM + P) \ y;
279     end
280
281     inter = abs(Viternew - Viterold)./max(1,abs(Viternew));
282     error = max(inter);
283     step = step + 1;
284     if error < tol
285         Vnew = Viternew;
286         break;
287     end
288     Viterold = Viternew;
289 end
290
291 % why
292 % largediagonal = zeros(m,1);
293 % largeidx = Vnew < Vstar;
294 % largediagonal(largeidx) = Large;
295 % P = spdiags(largediagonal, 0,m,m);
296 % sum(P,'all')
297 end
298
299 function out = sigmalvf(S)
300 % note that the local volatility function actually is independent of t
301 % vectorized output, assume column vector input
302 alpha = 15;
303 S0 = 76;
304 out = alpha./ (max(S0 + (S-S0)./(2*S0), 10^{-8})));
305 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

```

1 %% Q2b
2 % use CNRall, a version of CNR in Q2a that outputs the whole options value
3 % associated to the grid at t = 0
4
5 clearvars
6 close all
7 %% parameters initialization
8 S0 = 76;
9 T = 1;
10 Deltatau = T/25; % initial time step
11 dnorm = 0.1;
12 alpha = 15;
13 r = 0.03;
14 K = S0;
15
16 S = [ 0:0.1*S0:0.4*S0, ...
17     0.45*S0:0.05*S0:0.8*S0, 0.82*S0 :0.02*S0 :0.9 *S0, ...
18     0.91*S0 :0.01*S0 :1.1*S0, 1.12*S0 :0.02*S0 :1.2*S0, ...
19     1.25*S0 :0.05*S0 :1.6*S0, 1.7*S0 :0.1*S0 :2*S0, ...
20     2.2*S0, 2.4*S0, 2.8*S0, 3.6*S0, 5*S0, 7.5*S0, 10*S0];
21 S = S';
22
23 %% Now do refinements
24 Scell = {};
25 Scell{1} = S;
26 % list of initial timestep
27 Deltataulist = zeros(5,1);
28 Deltataulist(1) = Deltatau;
29 dnormlist = zeros(5,1);
30 dnormlist(1) = dnorm;
31 % 4 refinements
32 for i = 1:4
33     Sold = Scell{i};
34     Snew = [Sold;(Sold(1:end-1) + Sold(2:end)) / 2 ];
35     Snew = sort(Snew);
36     Scell{i+1} = Snew;
37     Deltataulist(i+1) = Deltatau/(4^i);
38     dnormlist(i+1) = dnorm/(2^i);
39 end
40 % use finest grid

```

```

41 S = Scell{end};
42 Deltatau = Deltataulist(end);
43 dnorm = dnormlist(end);
44
45 %% load data for european, obtain data for American, plot graphs
46
47 filename = 'Q1b.mat';
48 myVars = {'Vat0','deltaat0','gammaat0'};
49
50 % load data from Q1b
51 Euro = load(filename, myVars{:});
52
53 [V0, steptotal, counter] = CNRall(S, Deltatau, S0, T, K, r,dnorm,1);
54
55 Vat0 = V0;
56
57 % Central difference for delta and gamma at t = 0
58
59 idx = (S <= 1.5*S0) & (S >= 0.5 * S0);
60 idx = find(idx);
61 index1 = idx(1);
62 index2 = idx(end);
63
64 deltaxright = S(index1 + 1: index2 + 1) - S(index1:index2);
65 deltaxleft = S(index1:index2) - S(index1 - 1:index2-1);
66
67 deltafright0 = Vat0(index1 + 1: index2 + 1) - Vat0(index1:index2);
68 deltafleft0 = Vat0(index1:index2) - Vat0(index1 - 1:index2-1);
69 gammaat0 = (deltafright0 ./ deltaxright - deltafleft0./deltaxleft) ...
70 ./ ((deltaxright + deltaxleft)/2);
71 % not sure about this, I think this might improve the order of delta a bit
72 deltaat0 = (Vat0(index1+1: index2+1)-Vat0(index1-1:index2-1))...
73 ./ (deltaxright + deltaxleft) - gammaat0 /2 .* (deltaxright-deltaxleft);
74
75 figure(1)
76 plot(S(idx), Euro.Vat0(idx))
77 hold on
78 plot(S(idx),Vat0(idx))
79 title('Option value')
80 xlabel('asset value S')
81 legend('Euro','American')
82 hold off
83
84 figure(2)
85 plot(S(idx), Euro.deltaat0)
86 hold on
87 plot(S(idx), deltaat0)
88 title('Delta')
89 xlabel('asset value S')
90 legend('Euro','American')
91 hold off
92
93 figure(3)
94 plot(S(idx), Euro.gammaat0)
95 hold on
96 plot(S(idx), gammaat0)
97 title('Gamma')
98 xlabel('asset value S')
99 legend('Euro','American')
100 hold off
101
102 figure(4)
103 plot(S(idx), (Vat0(idx)- Euro.Vat0(idx)) )
104 title('American - European')
105 xlabel('asset value S')
106
107 figure(5)
108 plot(S(idx), (Vat0(idx)- Euro.Vat0(idx))./Euro.Vat0(idx) )
109 title('American - European, relative change')
110 xlabel('asset value S')
111
112 %% Discussion

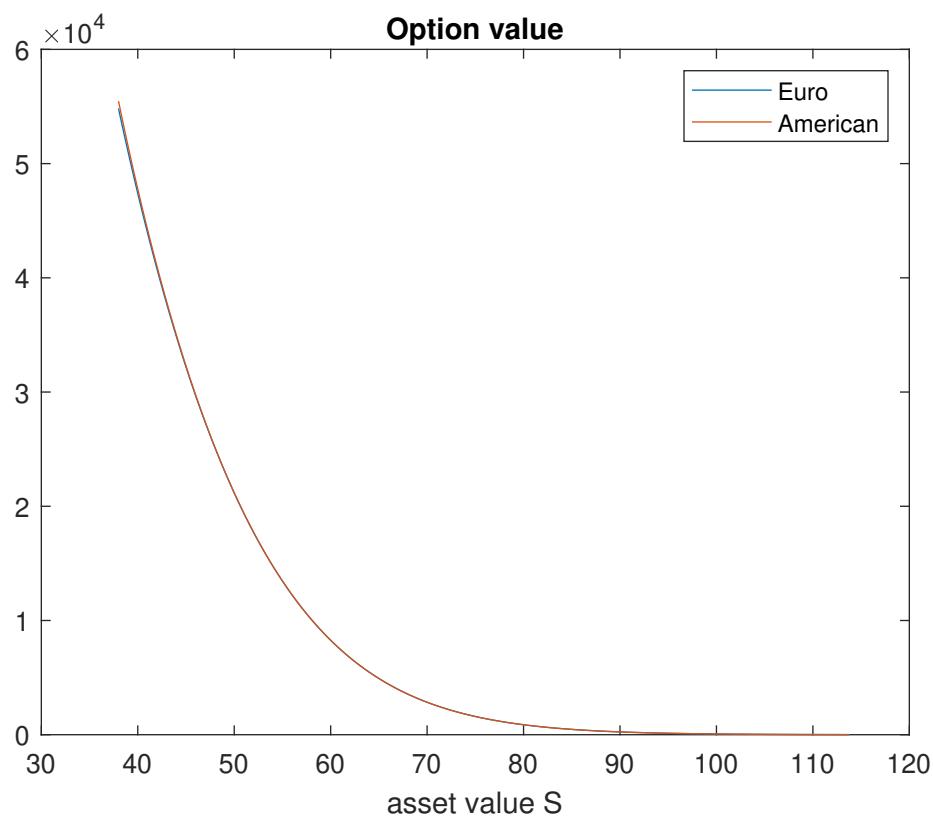
```

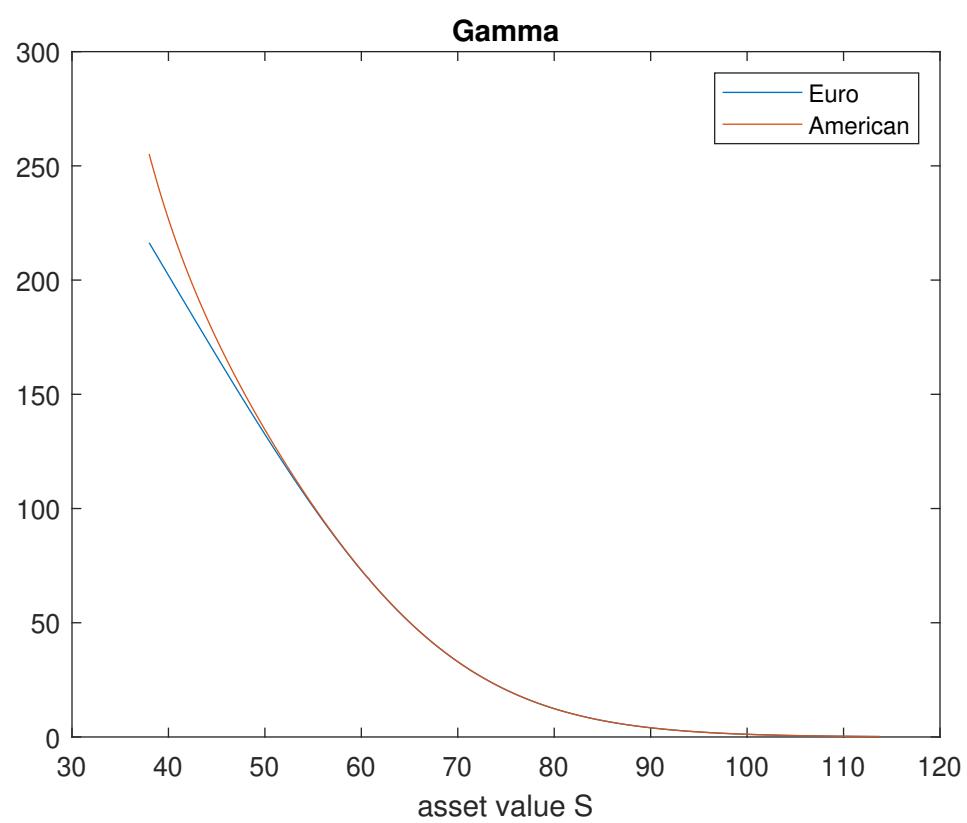
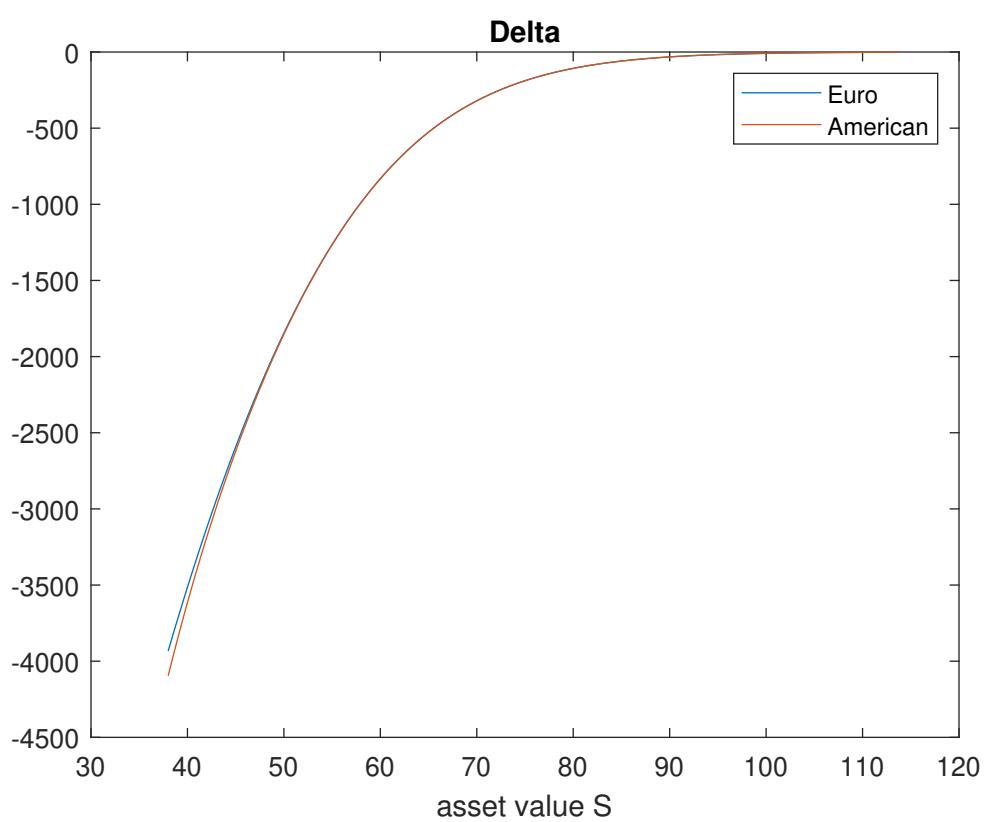
```

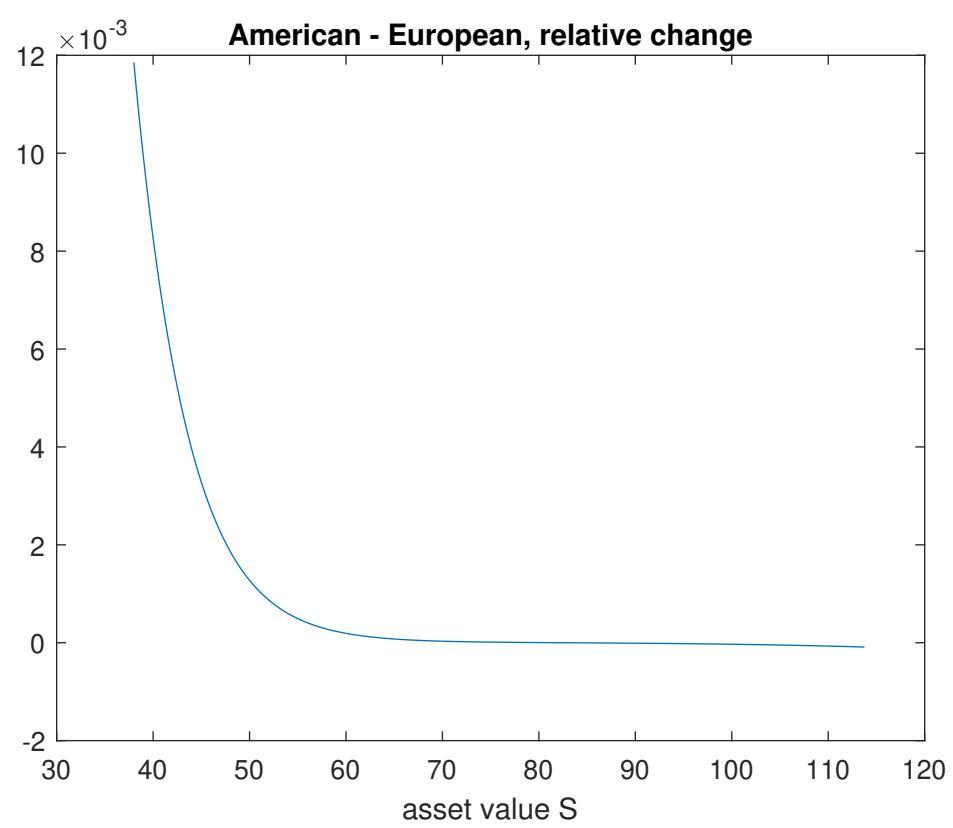
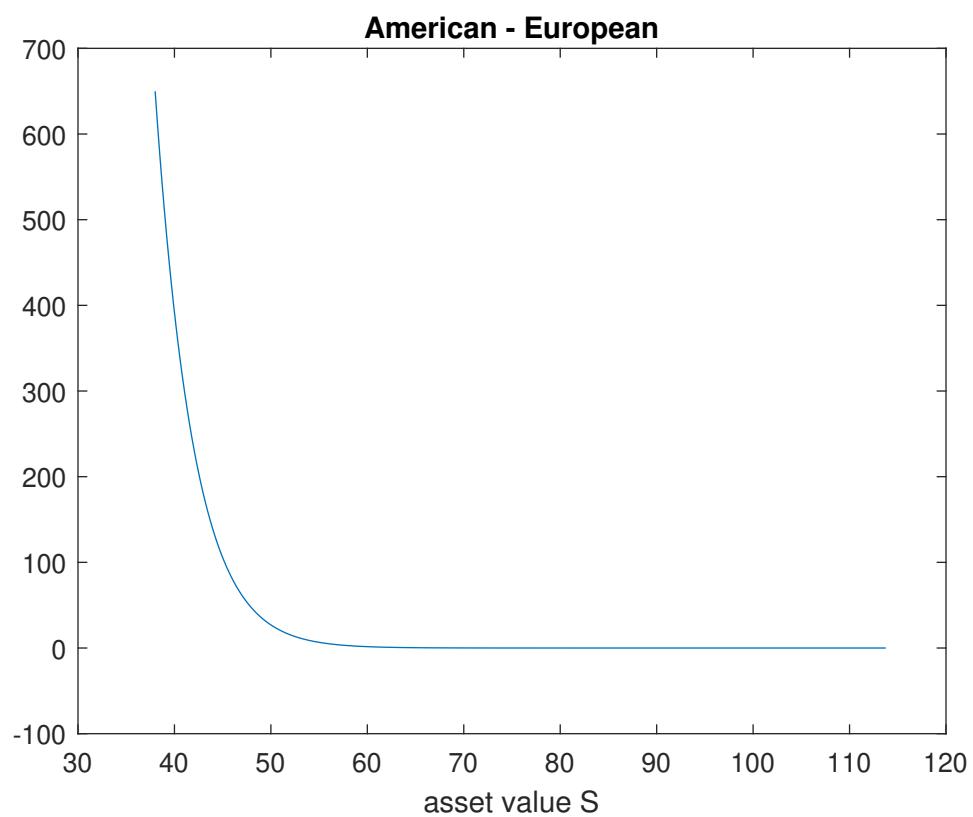
113 % Generally, we see that the European options value is smaller than or
114 % equal to that of American options. This is because American options can
115 % exercise early and hence its value must be >= European options. However,
116 % such difference is very small in our case. This is because r = 0.03 is
117 % quite small and the interval [0.5 *S0, 1.5*S0] that we consider is not
118 % very at the money for the power put options, so that the early exercise
119 % value is not very large. But note that the as S decreases (and so payoff
120 % increases), such early exercise value is larger and a larger difference
121 % is thus shown on the left of the option value plot.
122 %
123 % Note that if we set r = 0.5, then a much bigger difference will be seen.
124 % If we set r = 0, then no difference will be seen.

```

2.2.1 result







2.2.2 functions used

Listing 5: CNRall.m

```

1 function [V0, steptotal, counter] = CNRall(S, Deltatau, S0, T, K, r,dnorm,opt)
2 %% Description
3 % Inputs:
4 % opt = 1 for variable time, else for constant timestepping
5 % when opt = 1, Deltatau is the initial timestep
6 % when opt is else, Deltatau is the constant timestep
7 % Outputs:
8 % V0, option value at S (all the grid points) at t = 0
9 % steptotal, the total number of steps used for iterative solvers
10 % counter, the total time of choosing variable timestep; counter is just
11 % T/Deltatau when constant timestepping
12
13 %% preprocessing step for simple BS equation
14 % so that we have positive coefficient discretization
15 % these will also be used for CN and CN-Rannacher
16 Smid = S(2:end-1);
17 Sleft = S(1:end-2);
18 Sright = S(3:end);
19 sigma = sigmalvf(Smid);
20 alphacentral = sigma.^2 .* Smid .^2 ./ ((Smid - Sleft).* (Sright - Sleft)) - ...
21     r* Smid ./(Sright - Sleft);
22 betacentral = sigma.^2 .* Smid .^2 ./ ((Sright - Smid).* (Sright - Sleft)) + ...
23     r* Smid ./(Sright - Sleft);
24
25 alphaups = sigma.^2 .* Smid .^2 ./ ((Smid - Sleft).* (Sright - Sleft));
26 betaups = sigma.^2 .* Smid .^2 ./ ((Sright - Smid).* (Sright - Sleft)) + ...
27     r* Smid ./(Sright - Smid);
28
29 idx1 = alphacentral <0;
30 idx2 = betacentral <0;
31 idx = idx1 | idx2;
32
33 alphavec = alphacentral;
34 betavec = betacentral;
35 alphavec(idx) = alphaups(idx);
36 betavec(idx) = betaups(idx);
37 %% Some common data needed for constant and variable timestepping
38 m = length(S);
39 Vold = zeros(m,1);
40 Vnew = Vold;
41 N = T / Deltatau;
42 % initialize payoff
43 Vold = max(K-S,0).^3;
44
45 % penalty method data, tolerance and Large (penalty coefficient)
46 tol = 10^(-6); % as suggested by piazza
47 Large = 1/tol;
48
49 % Vstar, payoff at the gridpoints, for American option payoff constraints
50 Vstar = max(K-S,0).^3;
51
52 Bin = [[[ -alphavec; 0]; 0], [[r; (alphavec + betavec + r)]; 0], [[0; 0]; -betavec]];
53 % generic bin, used repeatedly below
54
55 steptotal = 0;
56
57 %% opt = 1, Use variable timestep
58 %%% also obtain number of steps to reach tol
59 %%% also need to update the Mhat and M
60 if opt == 1
61     dtold = Deltatau;
62     dtnew = 0;
63     currenttau = 0;
64     D = 1; % p.32 in lec 18, D = 1 fine for dollars
65     counter = 0;
66     % CN-Rannacher, variable time
67     while currenttau < T
68         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     % choose 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(largedialogal, 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

```

```

141      y = Vold + P* Vstar;
142      Viternew = (I + genM + P) \ y;
143  end
144
145  inter = abs(Viternew - Viterold)./max(1,abs(Viternew));
146  error = max(inter);
147  step = step + 1;
148  if error < tol
149    Vnew = Viternew;
150    break;
151  end
152  Viterold = Viternew;
153 end
154
155 end
156
157 function out = sigmalvf(S)
158 % note that the local volatility function actually is independent of t
159 % vectorized output, assume column vector input
160 alpha = 15;
161 S0 = 76;
162 out = alpha./ (max(S0 + (S- S0)./(2*S0), 10^(-8)));
163

```

3 Q3

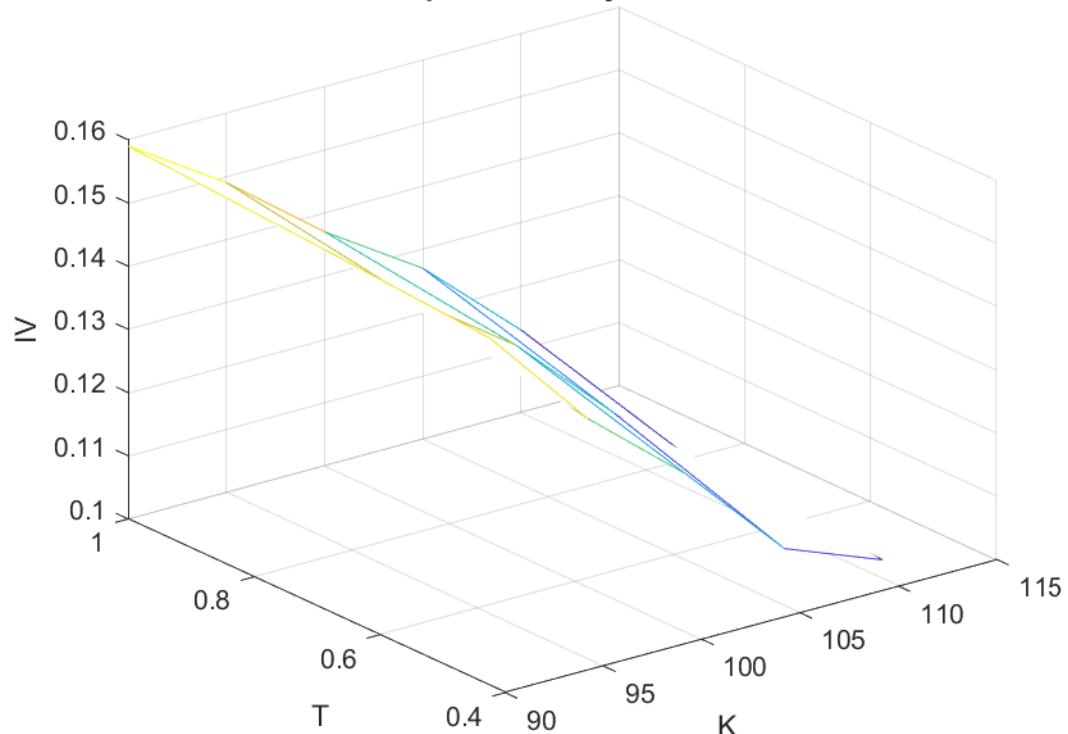
3.1 Q3a

Listing 6: Q3a.m

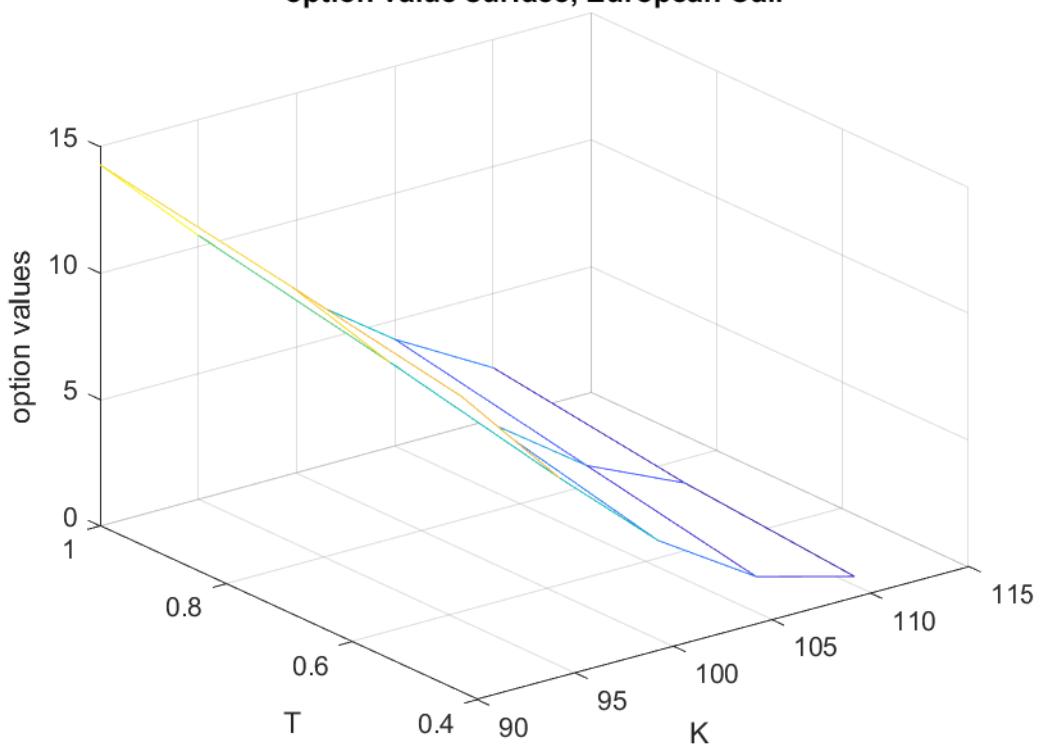
```
1 %% Q3a
2
3 clear vars
4 close all
5
6 S0 =100;
7 r = 0.03;
8
9 Klist = [0.9, 0.95, 1, 1.05, 1.1]* S0;
10 Tlist = [0.425, 0.695, 1];
11
12 % IV table
13 table2 = zeros(3,5);
14 table2(1,:) = [0.155, 0.138, 0.125, 0.109, 0.103];
15 table2(2,:) = [0.157, 0.144, 0.133, 0.118, 0.104];
16 table2(3,:) = [0.159, 0.149, 0.137, 0.127, 0.113];
17
18 [X,Y] = meshgrid(Klist, Tlist);
19
20 figure(1)
21 mesh(X,Y, table2)
22 xlabel('K')
23 ylabel('T')
24 zlabel('IV')
25 title('implied volatility surface')
26
27 optiontable = zeros(3,5);
28
29 for j = 1:5
30     for i = 1:3
31         [Call , put] = blsprice(S0, Klist(j), r, Tlist(i), table2(i,j));
32         optiontable(i,j) = Call;
33     end
34 end
35
36 figure(2)
37 mesh(X,Y, optiontable)
38 xlabel('K')
39 ylabel('T')
40 zlabel('option values')
41 title('option value surface, European Call')
42
43 %% Q3a Discussion
44 % We see that implied volatility surface and option value surface have
45 % similar shape. Maybe option value surface is a little bit smoother?
```

3.1.1 result

implied volatility surface



option value surface, European Call



3.2 Q3b

Listing 7: callpricer_CNR.m

```

1 %% Q3b, only CNR
2
3 function V0CNRan = callpricer_CNR(S, Deltatau, x, T, K, r)
4 % output option value at t =0, for all gridpoints, using fully implicit,
5 % Crank Nicolson, and CN-Rannacher
6 %
7 % input: x is the set of parameters. x =(w_1,w_2,u_1,u_2,v_1,v_2)', used in
8 % local volatility
9
10 m = length(S);
11 Vold = zeros(m,1);
12 Vnew = Vold;
13 N = T / Deltatau;
14 %% Boundary condition concern:
15 % initialize payoff, call option payoff
16 % Note that we adopt just the payoff as the boundary condition. By p.9 in
17 % Lec16, it is suggested to use S_m for call option. But I think the
18 % difference is small and adopt S_m - K instead, since S_m is quite large.
19 Vold = max(S-K,0);
20
21 %% preprocessing step for simple BS equation
22 % so that we have positive coefficient discretization
23 % these will also be used for CN and CN-Rannacher
24 Smid = S(2:end-1);
25 Sleft = S(1:end-2);
26 Sright = S(3:end);
27 sigma = sigmalvf(Smid,x);
28 alphacentral = sigma.^2 .* Smid.^2 ./ ((Smid - Sleft).* (Sright - Sleft)) -
...
29     r* Smid ./ (Sright - Sleft);
30 betacentral = sigma.^2 .* Smid.^2 ./ ((Sright - Smid).* (Sright - Sleft)) +
...
31     r* Smid ./ (Sright - Sleft);
32
33 alphaups = sigma.^2 .* Smid.^2 ./ ((Smid - Sleft).* (Sright - Sleft));
34 betaups = sigma.^2 .* Smid.^2 ./ ((Sright - Smid).* (Sright - Sleft)) +
...
35     r* Smid ./ (Sright - Smid);
36
37 idx1 = alphacentral <0;
38 idx2 = betacentral <0;
39 idx = idx1 | idx2;
40
41 alphavec = alphacentral;
42 betavec = betacentral;
43 alphavec(idx) = alphaups(idx);
44 betavec(idx) = betaups(idx);
45
46 %% Fully implicit
47
48 % build the matrix M by spdiags
49 Bin = Deltatau* [[[ -alphavec; 0]; 0], [[r; (alphavec + betavec + r)]; 0], [[0; 0]; -
betavec]];
50 M = spdiags(Bin, -1:1, m, m);
51 % spy(M) % see the nonzero entries of the matrix M
52 % Note that our M is time independent
53 I = speye(m);
54 [L,U,P] = lu(M+I);
55 % increase performance since the LU factorization is only computed once and
56 % being reused in each linear system solving
57 % to solve (M+I)x = b, y = L\ (P*b), x = U\y
58
59 % for n = 1 : N
60 %     y = L\ (P*Vold);

```

```

61 %     Vnew = U\y;
62 %     Vold = Vnew;
63 % end
64 % V0fullyimplicit = Vnew;
65
66
67 %% Crank-Nicolson
68 %%% only compute initial option value
69 %%% can easily modified to obtain options values at each grid points
70 % Vold = zeros(m,1);
71 % Vnew = Vold;
72 % % initialize payoff, call option payoff
73 % Vold = max(S-K,0);
74
75 Bin = Deltatau/2* [[[-alphavec;0];0], [[r; (alphavec + betavec +r)
    ];0],[[0;0];-betavec]];
76 Mhat = spdiags(Bin, -1:1, m, m);
77 % Note that our Mhat is time independent
78 I = speye(m);
79 [L1,U1,P1] = lu(Mhat+I);
80
81
82 % for n = 1 : N
83 %     y = L1\ (P1*(I-Mhat)*Vold);
84 %     Vnew = U1\y;
85 %     Vold = Vnew;
86 % ends
87 % V0CN = Vnew;
88
89 %% CN-Rannacher
90 Vold = zeros(m,1);
91 Vnew = Vold;
92 % initialize payoff, call option payoff
93 Vold = max(S-K,0);
94
95 % two steps of fully implicit
96 for n = 1 : 2
97     y = L\ (P*Vold);
98     Vnew = U\y;
99     Vold = Vnew;
100 end
101
102 % use CN after that
103 for n = 1 : N-2
104     y = L1\ (P1*(I-Mhat)*Vold);
105     Vnew = U1\y;
106     Vold = Vnew;
107 end
108 V0CNRan = Vnew;
109
110 end
111
112 function out = sigmalvf(S,x)
113 %% local volatility by simple neural network, (9) in asg4
114 %
115 % note that the local volatility function actually is independent of t
116 % vectorized output, assume column vector input
117 %
118 % inputs: S, gridpoints of S
119 %           x, set of parameters. x =(w_1,w_2,u_1,u_2,v_1,v_2)'
120 w1 = x(1);
121 w2 = x(2);
122 u1 = x(3);
123 u2 = x(4);
124 v1 = x(5);
125 v2 = x(6);

```

```

126 | y1 = 1./(u1 + v1 * S);
127 | y2 = 1./(u2 + v2 *S);
128 | out = 1./(1 + exp(w1 * y1 + w2* y2));
129 | end

```

3.3 Q3c

Listing 8: myfun.m

```

1 %% Q3c
2
3 function [F,J] = myfun(x)
4 % marketvalue: V_0^mkt, a column vector
5
6 % parameters and data preparation
7 S0 =100;
8 r = 0.03;
9 Klist = [0.9, 0.95, 1, 1.05, 1.1]* S0;
10 Tlist = [0.425, 0.695, 1];
11 N = 100; % what should be the correct number of steps?
12 % can change to N = 100
13
14 S = [ 0:0.1*S0:0.4*S0, ...
15     0.45*S0:0.05*S0:0.8*S0, 0.82*S0 :0.02*S0 :0.9 *S0, ...
16     0.91*S0 :0.01*S0 :1.1*S0, 1.12*S0 :0.02*S0 :1.2*S0, ...
17     1.25*S0 :0.05*S0 :1.6*S0, 1.7*S0 :0.1*S0 :2*S0, ...
18     2.2*S0, 2.4*S0, 2.8*S0, 3.6*S0, 5*S0, 7.5*S0, 10*S0];
19 S = S';
20
21 % can do the refinements following
22 Sold = S;
23
24 for i = 1:2
25     Snew = [Sold;(Sold(1:end-1) + Sold(2:end)) / 2 ];
26     Snew = sort(Snew);
27     Sold = Snew;
28 end
29
30 S = Snew;
31
32
33 % IV table
34 table2 = zeros(3,5);
35 table2(1,:) = [0.155, 0.138, 0.125, 0.109, 0.103];
36 table2(2,:) = [0.157, 0.144, 0.133, 0.118, 0.104];
37 table2(3,:) = [0.159, 0.149, 0.137, 0.127, 0.113];
38
39 market = zeros(3,5);
40
41 for j = 1:5
42     for i = 1:3
43         [Call , ~] = blsprice(S0, Klist(j), r, Tlist(i), table2(i,j));
44         market(i,j) = Call;
45     end
46 end
47
48 market = reshape(market, numel(market), []);
49
50 F = zeros(length(Tlist),length(Klist));
51 % Objective function values at x, stored as in table 2
52
53 S0id = S == S0;
54
55 for i = 1:length(Tlist)
56     T = Tlist(i);
57     Deltatau = T/N;

```

```

58     for j = 1 : length(Klist)
59         K = Klist(j);
60         V0CNRan = callpricer_CNR(S, Deltatau, x, T, K, r);
61         % only consider CNR, note that other methods can be obtained in
62         % callpricer.m
63         value = V0CNRan(S0id);
64         F(i,j) = value;
65     end
66 end
67
68 % Note that F now only contains value of options depending on x, not
69 % subtracted by the market values yet
70
71 % reshape F, column by column
72 F = reshape(F, numel(F), []);
73
74
75 % Note that we don't need to consider the constant market values when doing
76 % Jacobian. Hence we will subtract F by marketvalue later.
77 if nargout > 1 % Two output arguments
78     change = sqrt(eps);
79     Fnew = zeros(length(Tlist),length(Klist));
80     J = zeros(numel(Fnew), length(x)); % Jacobian of the function evaluated
81     at x
82
83     for k = 1 : length(x)
84         Fnew = zeros(length(Tlist),length(Klist));
85         % calculate the k-th column of J
86         dir = zeros(length(x), 1); % direction of derivative
87         dir(k) = 1;
88         for i = 1:length(Tlist)
89             T = Tlist(i);
90             Deltatau = T/N;
91             for j = 1 : length(Klist)
92                 K = Klist(j);
93                 V0CNRan = callpricer_CNR(S, Deltatau, x+ change*dir, T, K, r)
94                 ;
95                 % only consider CNR, note that other methods can be obtained
96                 % in
97                 % callpricer.m
98                 Fnew(i,j) = V0CNRan(S0id);
99             end
100        end
101    end
102
103    Fnew = reshape(Fnew, numel(Fnew), []);
104    J(:,k) = (Fnew - F) / change;
105
106 end

```

3.4 Q3d

Listing 9: Q3d.m

```

1 %% Q3d
2
3 clear vars
4 close all
5
6 %% Data preparation
7 S0 = 100;
8 r = 0.03;
9

```

```

10 Klist = [0.9, 0.95, 1, 1.05, 1.1]* S0;
11 Tlist = [0.425, 0.695, 1];
12
13 % IV table
14 table2 = zeros(3,5);
15 table2(1,:) = [0.155, 0.138, 0.125, 0.109, 0.103];
16 table2(2,:) = [0.157, 0.144, 0.133, 0.118, 0.104];
17 table2(3,:) = [0.159, 0.149, 0.137, 0.127, 0.113];
18
19 market = zeros(3,5);
20
21 for j = 1:5
22     for i = 1:3
23         [Call , put] = blsprice(S0, Klist(j), r, Tlist(i), table2(i,j));
24         market(i,j) = Call;
25     end
26 end
27
28 market = reshape(market, numel(market), []);
29
30 %% Levenberg–Marquardt method
31 options = optimset('Jacobian', 'on', 'Algorithm', 'levenberg-marquardt',...
32 'Display', 'iter', 'MaxIter', 50);
33
34 % initial x
35 x0 = [0,0,1,1,0,0]';
36
37 % xopt: optimal parameters xopt
38 % cal_error: calibration error
39 [xopt, cal_error] = lsqnonlin(@myfun, x0, [],[], options)
40
41 % for more information on the iterative display, see
42 % https://www.mathworks.com/help/optim/ug/iterative-display.html#f92387
43
44 %% IV from true model and the IV from the optimal parameters
45 Vfromxopt = myfun(xopt) + market;
46 Vfromxopt = reshape(Vfromxopt, length(Tlist), length(Klist))
47
48 [X,Y] = meshgrid(Klist, Tlist);
49 IVfromxopt = blsimpv(S0, X, r, Y, Vfromxopt)
50
51 % IVtrue = blsimpv(S0, X,r,Y, reshape(market, 3,5))
52 % note that table 2 are the implied volatilties from the true model
53 table2
54
55 figure(1)
56 mesh(X,Y, table2,'FaceColor','blue')
57 xlabel('K')
58 ylabel('T')
59 zlabel('IV')
60 title('implied volatility surface')
61 hold on
62 mesh(X,Y, IVfromxopt,'FaceColor','red')
63 legend('true','from xopt')
64
65 relerrorIV = max(abs(IVfromxopt - table2)./table2,[],'all')
66
67 %% Discussion
68 % We see that in general, our implied volatilties calibration does a fairly
69 % good job, with the relative error being around 12%. That is, the implied
70 % volatilties from the option values computed by model as given by xopt,
71 % are roughly 88% accurate, when compared to the true model, i.e. the
72 % implied volatilities in table 2. Based on the simplicity of our model
73 % (simple neural network), this is already quite good.
74 %
75 % Some thoughts on the formulation of the minimization problem. Note that

```

```

76 % the objective function is the 1/2 * sum of square of difference between
77 % option values from our model and option values from true model (or
78 % so-called market prices). What if we directly incorporate the difference
79 % between implied volatilities in the objective function? For instance, we
80 % can define F to be the sum of the square of the difference between the
81 % true implied volatilities and the implied volatilities from our model.
82 % 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

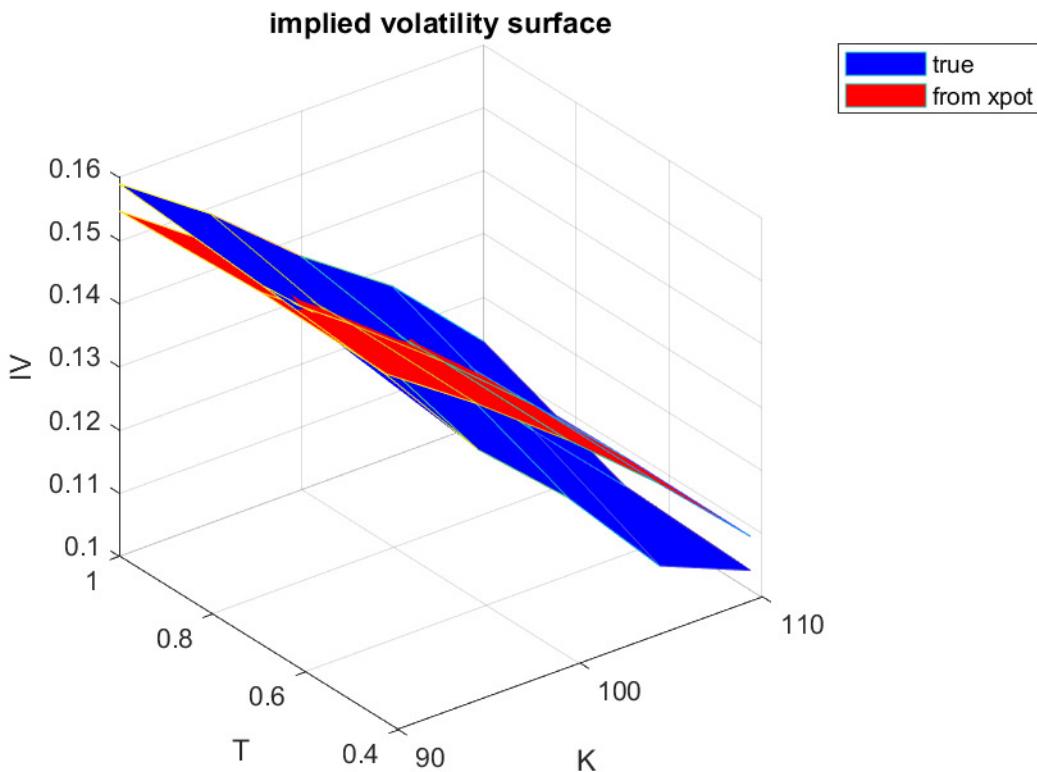
Vfromxopt =
11.711623569997183 7.565274404568785 4.136163782120330 1.720735068697083 0.440370920715182
12.927035435653890 8.947034335766618 5.525697661161873 2.853969561159865 1.085357244629239
14.176673782379037 10.288162733514328 6.849099907922974 3.996240518166629 1.872033627894845

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

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

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

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 hedging error will be
(quadratic)

$$i\text{-th error} = (\text{payoff of lookback option} - \Pi_T)^2$$

$$= (\max(H - S_T^i, 0) - \Pi_T)^2$$

$$= [\max(H - S_T^i, 0) - (x_1 \cdot S_T^i + x_2 \cdot \max(S_T^i - K, 0))]^2$$

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

$\mathbb{E}[(\text{payoff} - \Pi_T)^2]$ by MC, we take

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

$$= \frac{1}{M} \sum_{i=1}^M [\max(H_i - S_T^i, 0) - (x_1 \cdot S_T^i + x_2 \cdot \max(S_T^i - K, 0))]^2$$

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

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

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

Rmk: Note that we could have just used $H = \max_{\sigma \in \mathbb{R}^M} S_\sigma$ 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

$$c_1 x_1 + c_2 x_2 = C^0 \Rightarrow C^T x = C^0.$$

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 $\text{randn}(M, 1)$ 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 $\text{randn}(M, 1)$, the resulting random variables are independent among themselves and also independent of all previously generated random variables.

Lastly, we also use $\text{ones}(M, 1)$ to mean it generates a M -by-1 row vector with each entry being 1. Also, we use $\text{zeros}(m, n)$ 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

```

 $\Delta t = T/N$ 
 $S_0 = \text{ones}(M, 1) \cdot c_1$ 
 $H = S_0$ 
 $A = \text{zeros}(M, 2)$ 
 $b = \text{zeros}(M, 1)$ 
 $c = [c_1, c_2]^T$ 
for  $j = 1, \dots, N$  do
     $\phi = \text{randn}(M, 1)$  ▷  $M$  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

```

Q 4c 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 $\forall x, y \in \mathbb{R}^n, \theta \in [0, 1]$, we have

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

(since g is increasing
and f is convex)

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

(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 (11) is a convex problem.

Moreover, Euclidean norm is differentiable.
square of

Thus, we can apply KKT conditions.

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

$$(1) \quad g_1(x^*) = c^T x^* - c^0 = 0$$

$$(2) \quad \nabla_x L(x^*, v^*) = \nabla F(x^*) + v^* \nabla g_1(x^*) = 0$$

where $g_1(x) = c^T x - c^0$, $F(x) = \frac{1}{M} \|Ax - b\|_2^2$,

$$L(x, v) = F(x) + v^* g_1(x),$$

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

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

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

Q4c) Also, $\nabla g_1(x) = c^T$. 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

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

which then become

$$\begin{aligned} c^T x^* - c^0 &= 0 \\ \left(\frac{1}{M} \sum_{i=1}^M 2(s_T^i x^* + \max(s_T^i - k, 0)) \right) &= 0 \\ &\quad + v^* c^T = 0 \end{aligned}$$

(*) looks better.

(*) can be simplified to.

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

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

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

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

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

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

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

5 Q5

CS676 Asg 4. Q5.

& 6(a). Note $f(\alpha) = \mathbb{E}[(L-\alpha)^+]$

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

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

By Leibniz's rule,

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

$$= -p(x)(x-x) \cdot \frac{d}{dx}$$

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

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

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

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,

$$f''(\alpha) = \frac{d}{d\alpha} f'(\alpha) = \frac{d}{d\alpha} \left(- \int_{\alpha}^{\infty} p(l) dl \right)$$
$$= p(\alpha) > 0.$$

Since $p(l) > 0 \forall l \in (-\infty, \infty)$

Thus, $f(\alpha)$ is convex, since $f''(\alpha) > 0$.

Thus, function f is convex since

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

c)

Our problem can be stated as

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

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

Then

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

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

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

$$\text{and } f''(\alpha) > 0.$$

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

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

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

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

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

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

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

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

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

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

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

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

```
1 %% Q6a
2 clearvars
3 close all
4
5 rng('default')
6
7 SA0scalar = 20;
8 SB0scalar = 25;
9 r = 0.03;
10 m = 10000;
11
12 SA0 = ones(m,1) * SA0scalar;
13 SB0 = ones(m,1) * SB0scalar;
14
15 T = 1/12; % one month
16 N = 10;
17
18 SA = eulerlvf(SA0, T, N, 1, r);
19 SA = sort(SA); % sort the SA so that the order matches options
20 % max(SA) -> 20
21 % min(SA) -> 20
22 SB = eulerlvf(SB0, T, N, 0, r);
23 SB = sort(SB);
24 % max(SB) -> 25
25 % min(SB) -> 25
26
27 SAreturn = (SA - SA0scalar) ./ SA0scalar;
28 SBreturn = (SB - SB0scalar) ./ SB0scalar;
29
30 % make grid, refine the grid 4 times
31 SAgrid = gridrefine(SA0scalar, SA, 4);
32 SBgrid = gridrefine(SB0scalar, SB, 4);
33
34 % note that the probability of obtaining duplicate elements are 0
35 % in fact our choice of random seed make sure that there is no duplicate in
36 % SAgrid, SBgrid
37
38 % save all the rate of return on all the instruments
39 return_A = zeros(m,5);
40 return_B = zeros(m,5);
41 % first column stores the stock
42 return_A(:,1) = SAreturn;
43 return_B(:,1) = SBreturn;
44
45 Tlist = [0.5, 0.5, 0.5, 2/3]';
46 Klist = [SA0scalar, 1.1*SA0scalar, 1.2*SA0scalar, SA0scalar]';
47 [~, idxA] = intersect(SAgrid, SA);
48 N = 100;
49 idscalar = SAgrid == SA0scalar;
50 [V1old, V2old, V3old, Vppold] = pricer_CNR(SAgrid, N, 1, Tlist, Klist, r);
51 price_old = [V1old(idscalar), V2old(idscalar), V3old(idscalar), Vppold(idscalar)
    ];
52 [V1, V2, V3, Vpp] = pricer_CNR(SAgrid, N, 1, Tlist - 1/12, Klist, r);
53 price_new = [V1(idxA), V2(idxA), V3(idxA), Vpp(idxA)];
54 return_A(:,2:5) = (price_new - price_old) ./ price_old;
55 % for stock A, 2nd to 4th columns are call, last column is powerput
56
```

```

57 Tlist = [1,1,1,1]';
58 Klist = [0.8*SB0scalar, 0.9*SB0scalar, SA0scalar, SA0scalar]';
59 [~,idxB] = intersect(SBgrid, SB);
60 N = 100;
61 idscalar = SBgrid == SB0scalar;
62 [V1old,V2old,V3old,Vppold] = pricer_CNR(SBgrid, N, 0, Tlist, Klist, r);
63 price_old = [V1old(idscalar),V2old(idscalar),V3old(idscalar),Vppold(idscalar)
    ];
64 [V1,V2,V3,Vpp] = pricer_CNR(SBgrid, N, 0, Tlist- 1/12, Klist, r);
65 price_new = [V1(idxB),V2(idxB),V3(idxB),Vpp(idxB)];
66 return_B(:,2:5) = (price_new - price_old) ./ price_old;
67 % for stock B, 2nd to 4th columns are put, last column is powerput
68
69
70 % obtain equally weighted portfolio
71 eq_weg_portfolio = 1/10 * (sum(return_A, 2) + sum(return_B,2));
72 % histogram
73 figure(1)
74 histogram(eq_weg_portfolio,50)
75 title('histogram of equal allocation')
76 xlabel('rate of return')
77 ylabel('frequency')
78
79 % Note that the histogram is not normal, very skewed
80
81 %% Q6b
82 allresult = [return_A, return_B]; % 10000 by 10 matrix
83
84 loss = -allresult; % so that we consider a min opt problem
85
86 betalist = [0.8, 0.85, 0.9,0.95];
87 rholist = zeros(4,1);
88 M = 10000;
89 n =10;
90
91 avgloss = mean(loss)';
92 f = [avgloss;zeros(M+1,1)]; % so that we consider a min opt problem
93 varlength= n + M + 1; % our variable is listed as [x,y,alpha]'
94 Aeq = zeros(1, varlength);
95 Aeq(1,1:n) =1;
96 beq = 1;
97
98 A = zeros(n + 1 + M + M,varlength);
99 b = zeros(n + 1 + M + M,1);
100 % first n columns for x_i \geq 0
101 A(1:n, 1:n) = -eye(n); % need to add negative sign since A x \leq b
102 b(1:n) = 0;
103
104 % next column for CVar constraint, we will update it in the for loop below
105
106 % then constraints on the variables y that is larger L^T_i x - alpha
107 A(n+2:n+1+M,n+1:n+M) = -eye(M);
108 A(n+2:n+1+M, end) = -1;
109 A(n+2:n+1+M, 1:n) = loss;
110 b(n+2:n+1+M) = 0;
111
112 % then nonnegative constraint on y
113 A(n+1+M+1:end, n+1:n+M) = -eye(M);
114 b(n+1+M+1:end) = 0;
115
116 allvaroptlist = zeros(varlength, 4);
117
118 comparetable = zeros(5,5);
119 alphalist = zeros(4,1 );
120
121

```

```

122 for i = 1:4
123     beta = betalist(i); % for solving min opt problem
124     [~,cvar] = dVaRCVaR(eq_weg_portfolio, beta);
125     rhoList(i) = -cvar; % consider cvar in terms of loss
126     A(n+1, end) = 1;
127     A(n+1,n+1:n+M) = 1/(M*(1-beta));
128     b(n+1) = rhoList(i);
129     % solve min problem
130     allvar = linprog(f,A,b,Aeq,beq);
131     allvarOptList(:,i) = allvar;
132     x = allvar(1:10);
133     alpha = allvar(end);
134
135     alphalist(i) = alpha; % var approximation
136
137     optPortfolio = allresult * x;
138
139 %     figure(i+2)
140 %     histogram(optPortfolio,50)
141
142     optmean = mean(optPortfolio);
143
144     compareTable(i+1,1) = optmean;
145
146     for j = 1:4 % for table,
147         beta = betalist(j);
148         [~,cvar] = dVaRCVaR(optPortfolio, beta);
149         compareTable(i+1,j+1) = -cvar; % consider cvar in terms of loss
150     end
151     % note that we can also kind of recover the CVaR by using
152     % alpha + 1 / (M*(1-beta)) * sum(y), but we can only recover one CVaR
153     % for the particular beta we used in minimization problem
154 end
155
156 compareTable(1,1) = mean(eq_weg_portfolio);
157 compareTable(1,2:end) = rhoList;
158
159 format short
160
161 compareTable = array2table(compareTable,'VariableNames',...
162     {'mean','80% CVaR','85% CVaR','90% CVaR','95% CVaR'},...
163     'RowNames',{'equal allocation','optimal wrt 80%',...
164     'optimal wrt 85%','optimal wrt 90%','optimal wrt 95%'})
165
166 %% Discussion
167 % We see that the optimal portfolios (with respect to different beta), have
168 % very similar mean rate of return, around 0.00363. Note that the rate of
169 % return we are considering here is for holding the portfolio for one
170 % month. So the risk-free rate that we should compare to is  $\exp(r/12)-1$ ,
171 % which is around 0.002503127605795. We observe that the equal allocation
172 % portfolio has mean rate return 0.00269571202274778, which is very close
173 % to the risk-free rate. We observe that the mean rate return for optimal
174 % portfolios are significantly better than equal allocation and risk free
175 % rate.
176 %
177 % For optimal portfolio corresponding to beta and rho_beta (i.e CVaR of
178 % equal allocation portfolio with confidence level beta), the CVaR with
179 % confidence level beta is very close to rho_beta. This is true because the
180 % optimization procedure will kind of try to make use of all the given
181 % budget (i.e. rho_beta) of the CVaR of the optimal portfolio. Note that
182 % the second line constraint is an upper bound of the CVaR_beta of the
183 % optimal portfolio. To maximize the return, the optimal portfolio will try
184 % its best to reach the upper bound, making more rooms for increasing the
185 % return and hence resulting in a CVaR_beta that is close to rho.
186 %
187 % Suppose beta_1 < beta_2. (for instance beta_1 = 0.8, beta_2 = 0.9). We

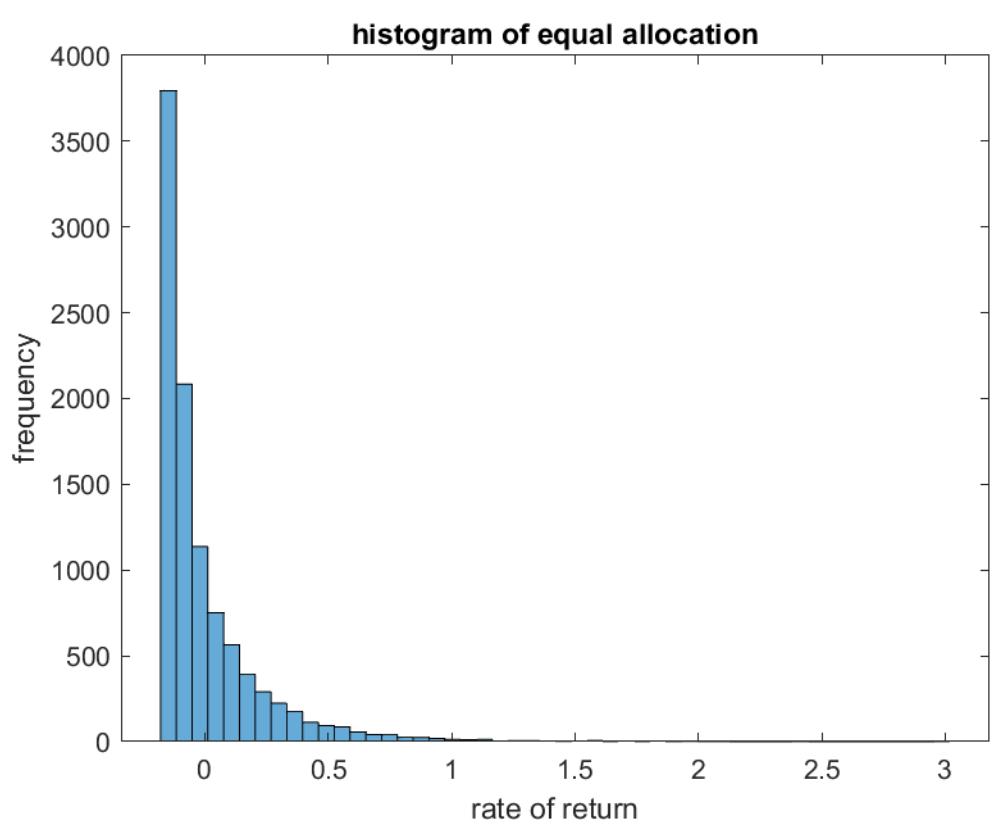
```

```

188 % observe that optimal portfolio wrt beta_1 has mean slight lower than that
189 % wrt beta_2. Moreover, with respect to any beta that we consider, the
190 % CVaR_beta of the former portfolio are always less than the latter
191 % portfolio. I suspect that it is true because we actually put more
192 % constraints on the former portfolio since a lower beta_1 means that we
193 % have to consider more scenarios when calculating the CVaR. Thus, we put
194 % constraints on more scenarios and resulting in a slightly "safer"
195 % portfolio than the latter portfolio in the sense that all CVaR calculated
196 % are smaller. But it is done at the cost of having a lower mean rate of
197 % return. (in short, constraint on 80% CVaR impose constraints on worse 20%
198 % scenarios, while constraint on 95% CVaR only impose constraints on worse
199 % 5% scenarios. A limit on 80% CVaR will often also limit 95% CVaR to a
200 % satisfiable degree while the converse is not necessarily the case.
201 %
202 % Another possible reason for the above phenomenon might be that the risk
203 % budget obtained from taking CVaR with respect to different beta on the
204 % equal allocation portfolio acutally result in a rather uneven
205 % distribution of risk budget that allows more risk tolerance when we take
206 % large beta. (i think the last paragraph is more convincing)
207
208 function S = gridrefine(S0, Snew,n)
209 % S0, initial stock value
210 % Snew, all new stock prices that we want to add
211 % n, number of refinements on (2) in asg 4
212 S = [ 0:0.1*S0:0.4*S0, ...
213     0.45*S0:0.05*S0:0.8*S0, 0.82*S0 :0.02*S0 :0.9 *S0, ...
214     0.91*S0 :0.01*S0 :1.1*S0, 1.12*S0 :0.02*S0 :1.2*S0, ...
215     1.25*S0 :0.05*S0 :1.6*S0, 1.7*S0 :0.1*S0 :2*S0, ...
216     2.2*S0, 2.4*S0, 2.8*S0, 3.6*S0, 5*S0, 7.5*S0, 10*S0];
217 S = S';
218
219 for i = 1: n
220     S = [S;(S(1:end-1) + S(2:end)) / 2 ];
221     S = sort(S);
222 end
223
224 S = [S; Snew];
225 S = sort(S);
226
227 end

```

6.2 result



```
Optimal solution found.
```

```
comparetable =
```

```
5x5 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	0.14982	0.15055	0.151
optimal wrt 90%	0.0036374	0.14972	0.15051	0.15124	0.15171
optimal wrt 95%	0.0036388	0.14988	0.15074	0.15148	0.15192

6.3 functions scripts

Listing 11: sigmalvf_a.m

```
1 function sigma_a = sigmalvf_a(S)
2 %% local volatility by \alpha ./ sqrt(S)
3 %
4 % note that the local volatility function actually is independent of t
5 % vectorized output, assume column vector input
6 %
7 % inputs: S, stock A price
8 alpha = 0.85;
9 sigma_a = alpha./ sqrt(S);
10 end
```

Listing 12: sigmalvf_b.m

```
1 function sigma_b = sigmalvf_b(S)
2 %% local volatility by \alpha ./ sqrt(S)
3 %
4 % note that the local volatility function actually is independent of t
5 % vectorized output, assume column vector input
6 %
7 % inputs: S, stock B price
8 alpha = 0.9;
9 sigma_b = alpha./ sqrt(S);
10 end
```

Listing 13: eulerlvf.m

```
1 % Q6a
2 function Snew = eulerlvf(Sold, T, N, opt, r)
3 % Sold is the stock price at time 0
4 % Snew will be the stock price at time T, generated by Euler method
5 %
6 % opt = 1 for stock A, else for stock B
7 % N: number of timesteps taken, integer, so dt = T / N
8 %
9 % assume column vector as input
10 m= length(Sold);
11 Snew = zeros(m,1);
12 dt = T/N;
13 sqrt_dt = sqrt(dt);
14
15 if opt == 1
16     for i = 1: N
```

```

17     phi = randn(m,1);
18     Snew = Sold + Sold .*(r * dt + sigmalvf_a(Sold)* sqrt_dt .*phi);
19     Sold = Snew;
20 end
21 else
22 for i = 1: N
23     phi = randn(m,1);
24     Snew = Sold + Sold .*(r * dt + sigmalvf_b(Sold)* sqrt_dt .*phi);
25     Sold = Snew;
26 end
27 end
28
29 end

```

Listing 14: dVaRCVaR.m

```

1 %% ask, the input is profit or loss?
2 % after reading lecture carefully, i think it is profit
3
4 function [var,cvar] = dVaRCVaR(PL, beta)
5 % PL: discrete P&L distribution with M independent samples
6 % assume PL is a column vector
7 % assume PL is profit, i.e. positive value for profit
8 % beta could be 95%
9
10 M = length(PL);
11 PL_sorted = sort(PL); % sorts in ascending order.
12
13 idx = floor((1-beta)*M);
14 var = PL_sorted(idx);
15
16 cvar = mean(PL_sorted(1:idx));
17
18 end

```

Listing 15: pricer_CNR.m

```

1 %% Q6a, only CNR, for different stock, both powerput and standard call and put
2
3 function [V1,V2,V3,Vpp] = pricer_CNR(S, N, opt, Tlist, Klist, r)
4 % output option value at t =0, for all gridpoints, using CN-Rannacher
5 %
6 % if opt = 1, compute call option prices and compute powerput on stock A,
7 % the order of computation matches the order of Tlist, Klist
8 % V1, V2, V3 will be option values on the grid S for the 3 call options
9 %
10 % if opt is sth else, compute put option prices and compute powerput
11 % on stock B
12 % V1, V2, V3 will be option values on the grid S for the 3 put options
13 %
14 % in both cases, Vpp are the option values of the grid for the powerput
15
16 m = length(S);
17
18
19 %% preprocessing step for simple BS equation
20 % so that we have positive coefficient discretization
21 % these will also be used for CN and CN-Rannacher
22 Smid = S(2:end-1);
23 Sleft = S(1:end-2);
24 Sright = S(3:end);
25
26 if opt == 1
27     sigma = sigmalvf_a(Smid);
28 else
29     sigma = sigmalvf_b(Smid);
30 end
31
32 alphacentral = sigma.^2 .* Smid.^2 ./ ((Smid - Sleft).* (Sright - Sleft)) - ...
33     r* Smid ./ (Sright - Sleft);
34 betacentral = sigma.^2 .* Smid.^2 ./ ((Sright - Smid).* (Sright - Sleft)) + ...
35     r* Smid ./ (Sright - Sleft);

```

```

36
37 alphaups = sigma.^2 .* Smid.^2 ./ ((Smid - Sleft).* (Sright - Sleft));
38 betaups = sigma.^2 .* Smid.^2 ./ ((Sright - Smid).* (Sright - Sleft)) + ...
39     r* Smid ./ (Sright - Smid);
40
41 idx1 = alphacentral <0;
42 idx2 = betacentral <0;
43 idx = idx1 | idx2;
44
45 alphavec = alphacentral;
46 betavec = betacentral;
47 alphavec(idx) = alphaups(idx);
48 betavec(idx) = betaups(idx);
49
50 % Boundary condition concern:
51 % initialize payoff, call option payoff
52 % Note that we adopt just the payoff as the boundary condition. By p.9 in
53 % Lec16, it is suggested to use S_m for call option. But I think the
54 % difference is small and adopt S_m - K instead, since S_m is quite large.
55 % similar approach is taken for put and powerput
56
57 %% main for loop
58 result= zeros(m,4);
59
60 for i = 1:4
61     %% Fully implicit
62     T = Tlist(i);
63     Deltatau = T/N;
64     % build the matrix M by spdiags
65     Bin = Deltatau* [[[ -alphavec; 0]; 0], [[r; (alphavec + betavec + r)]; 0], [[0; 0]; ...
66         -betavec]];
67     M = spdiags(Bin, -1:1, m, m);
68     % spy(M) % see the nonzero entries of the matrix M
69     % Note that our M is time independent
70     I = speye(m);
71     [L,U,P] = lu(M+I);
72     % increase performance since the LU factorization is only computed once and
73     % being reused in each linear system solving
74     % to solve (M+I)x = b, y = L\ (P*b), x = U\y
75
76     %% Crank-Nicolson
77     Bin = Deltatau/2* [[[ -alphavec; 0]; 0], [[r; (alphavec + betavec + r)]; 0], [[0; 0]; ...
78         -betavec]];
79     Mhat = spdiags(Bin, -1:1, m, m);
80     % Note that our Mhat is time independent
81     I = speye(m);
82     [L1,U1,P1] = lu(Mhat+I);
83
84     %% CN-Rannacher
85     K = Klist(i);
86     Vold = zeros(m,1);
87     Vnew = Vold;
88     % initialize payoff, call option payoff
89     if opt == 1
90         if i < 4
91             Vold = max(S-K,0);
92         else
93             Vold = max(K-S,0).^3;
94         end
95     else
96         if i < 4
97             Vold = max(K-S,0);
98         else
99             Vold = max(K-S,0).^3;
100        end
101    end
102
103    % two steps of fully implicit
104    for n = 1 : 2
105        y = L\ (P*Vold);
106        Vnew = U\y;
107        Vold = Vnew;

```

```
106     end
107
108 % use CN after that
109 for n = 1 : N-2
110     y = L1\ (P1*(I-Mhat)*Vold);
111     Vnew = U1\y;
112     Vold = Vnew;
113 end
114 result(:,i) = Vnew;
115
116
117 V1 = result(:,1);
118 V2 = result(:,2);
119 V3 = result(:,3);
120 Vpp =result(:,4);
121
122
123 end
```