Assignment 2 - Dupire formula and calibration of interest rate models

Table of Contents

Written by Peili Guo (peili.guo.7645@student.uu.se) and Francisco José Peralta Alguacil (franciscojose.peraltaalguacil.0481@student.uu.se) This report is for Computational Finance: Calibration and Estimation Assignment 2.

In this project we will use the Dupire formula in two different versions (the first one using the derivatives of C and the second one with implied volatilities) for calibrating the volatility σ of a call option having observations for different strike prices K and maturity times T.

In the second part, we will proceed to calibrate the interest rate from STIBOR and swap rates using the Vasicek model for some market data. This calibration will be performed using Nelder-Mead simplex method (fminsearch) and BFGS quasi-Newton (fminunc).

Part 1

```
clc
clear
close all

load('LocVol.mat');

%
% Analytical solution. First of all, we will plot the analytical
 solution
% for the calibration problem in order to compare it with the results
 that
% we will obtain troughout the project. The plot is shown at the end
 of
% this part, together with the numerical results obtained.
%

figure(4)
[X,Y]=meshgrid(K,T);
SigmaAnalyt = 0.15+0.15*(0.5+2.*Y).*((X./100-1.2).^2)./(((X.^2)./
(100.^2))+1.44);
surf(X,Y,SigmaAnalyt);
zlim([0 0.35])
xlabel("K")
ylabel("T")
zlabel("sigma")
title("Analytical solution")
caxis([0.00 0.3])
```

```
colormap(jet)
shading interp
colorbar

%
% Numerical solution for interior points. Here we calibrate the
% volatility for the interior points of the given data. We do that
 because
% of the fact that we will encounter some problems when adding
 boundary
% points, as we will see later on.
%

Tint = T(2:length(T)-1);
Kint = K(2:length(K)-1);
for i=2:size(C,1)-1
   for j=2:size(C,2)-1
           dCdTint(i-1,j-1) = (C(i+1,j)-C(i-1,j))/(T(i+1)-T(i-1));
           dCdKint(i-1,j-1) = (C(i,j+1)-C(i,j-1))/(K(j+1)-K(j-1));
   end
end

deltaK = 1;
for i=2:size(C,1)-1
   for j=2:size(C,2)-1
           dCdK2int(i-1,j-1) = (C(i,j+1)-2*C(i,j)+C(i,j-1))/
(deltaK)^2;
   end
end

for i=1:size(dCdKint,1)
   for j=1:size(dCdKint,2)
       SigmaNumInt(i,j) =
 sqrt(2*(dCdTint(i,j)+(r-0)*Kint(j)*dCdKint(i,j)+0*C(i,j))/
(Kint(j)*Kint(j)*dCdK2int(i,j)));
   end
end

figure(1)
[XnumInt,YnumInt]=meshgrid(Kint,Tint);
surf(XnumInt,YnumInt,SigmaNumInt);
zlim([0 0.35])
xlabel("K")
ylabel("T")
zlabel("sigma")
title("Numerical solution for interior points")

caxis([0.0 0.3])
colormap(jet)
shading interp
colorbar

%
% Numerical solution. Now we proceed to compute the value of
```

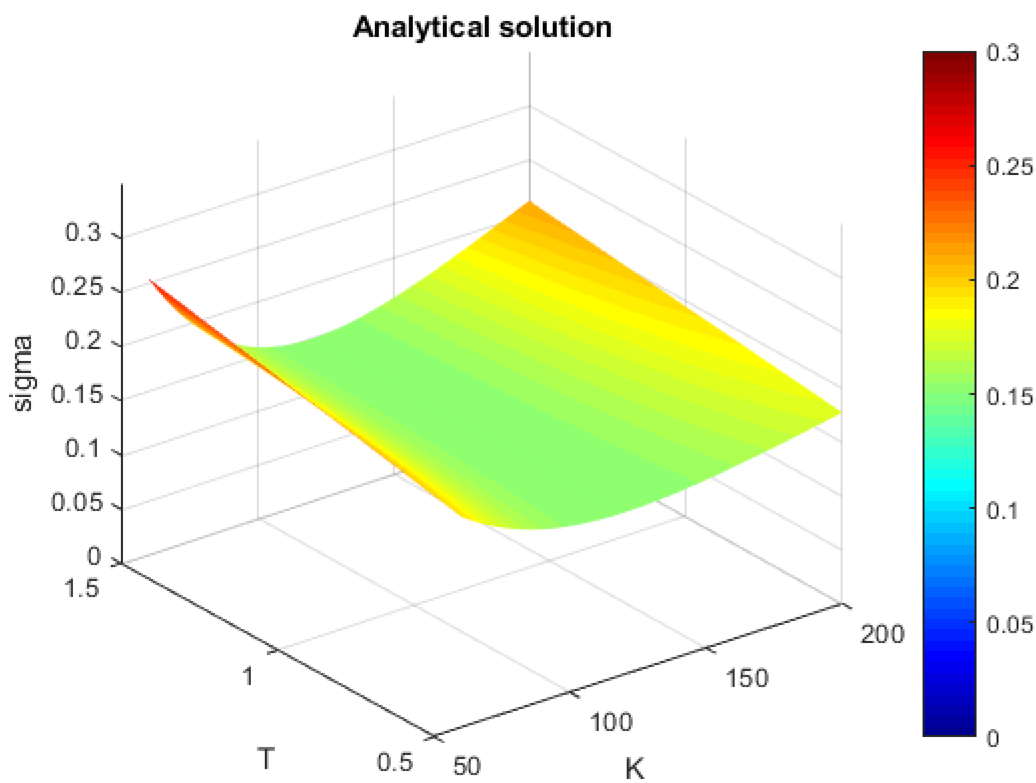
```
% $\sigma$ for the whole region (including the boundary points).
 However,
% as can be seen in the plot at the end of the section, we found out
 the
% problem that the values corresponding to the boundary of T=0.5
 somehow
% diverge towards negative values (in fact, a couple of points before
% computing the square root were negative, and an absolute value that
% should not be there was added in order to be able to at least obtain
 some
% results for the rest of the points).
%

for i=1:size(C,1) %i for T
   for j=1:size(C,2) %j for K
       if i==1 && j==1 %left bottom corner
           dCdT(i,j) = (-C(i+2,j)+4*C(i+1,j)-3*C(i,j))/(2*(T(i+1)-
T(i))); %taylor expansion fw with 2 order error
           dCdK(i,j) = (-C(i,j+2)+4*C(i,j+1)-3*C(i,j))/(2*(K(j+1)-
K(j)));
       elseif i==1 && j<size(C,2) && j>1 %bottom boundary
           dCdT(i,j) = (-C(i+2,j)+4*C(i+1,j)-3*C(i,j))/(2*(T(i+1)-
T(i)));
           dCdK(i,j) = (C(i,j+1)-C(i,j-1))/(K(j+1)-K(j-1));
       elseif i==1 && j==size(C,2) %right bottom corner
           dCdT(i,j) = (-C(i+2,j)+4*C(i+1,j)-3*C(i,j))/(2*(T(i+1)-
T(i)));
           dCdK(i,j) = (3*C(i,j)-4*C(i,j-1)+C(i,j-2))/(2*(K(j)-
K(j-1)));
       elseif i>1 && i<size(C,1) && j==1 %left side boundary
           dCdT(i,j) = (C(i+1,j)-C(i-1,j))/(T(i+1)-T(i-1));
           dCdK(i,j) = (-C(i,j+2)+4*C(i,j+1)-3*C(i,j))/(2*(K(j+1)-
K(j)));
       elseif i>1 && i<size(C,1) && j==size(C,2) %right side boundary
           dCdT(i,j) = (C(i+1,j)-C(i-1,j))/(T(i+1)-T(i-1));
           dCdK(i,j) = (3*C(i,j)-4*C(i,j-1)+C(i,j-2))/(2*(K(j)-
K(j-1)));
       elseif i==size(C,1) && j==1 %left up corner
           dCdT(i,j) = (3*C(i,j)-4*C(i-1,j)+C(i-2,j))/(2*(T(i)-
T(i-1)));
           dCdK(i,j) = (-C(i,j+2)+4*C(i,j+1)-3*C(i,j))/(2*(K(j+1)-
K(j)));
       elseif i==size(C,1) && j>1 && j<size(C,2) %upper boundary
           dCdT(i,j) = (3*C(i,j)-4*C(i-1,j)+C(i-2,j))/(2*(T(i)-
T(i-1)));
           dCdK(i,j) = (C(i,j+1)-C(i,j-1))/(K(j+1)-K(j-1));
       elseif i==size(C,1) && j==size(C,2) %right up corner
           dCdT(i,j) = (3*C(i,j)-4*C(i-1,j)+C(i-2,j))/(2*(T(i)-
T(i-1)));
           dCdK(i,j) = (-3*C(i,j)-4*C(i,j-1)+C(i,j-2))/(2*(K(j)-
K(j-1)));
       else
           dCdT(i,j) = (C(i+1,j)-C(i-1,j))/(T(i+1)-T(i-1));
           dCdK(i,j) = (C(i,j+1)-C(i,j-1))/(K(j+1)-K(j-1));
```

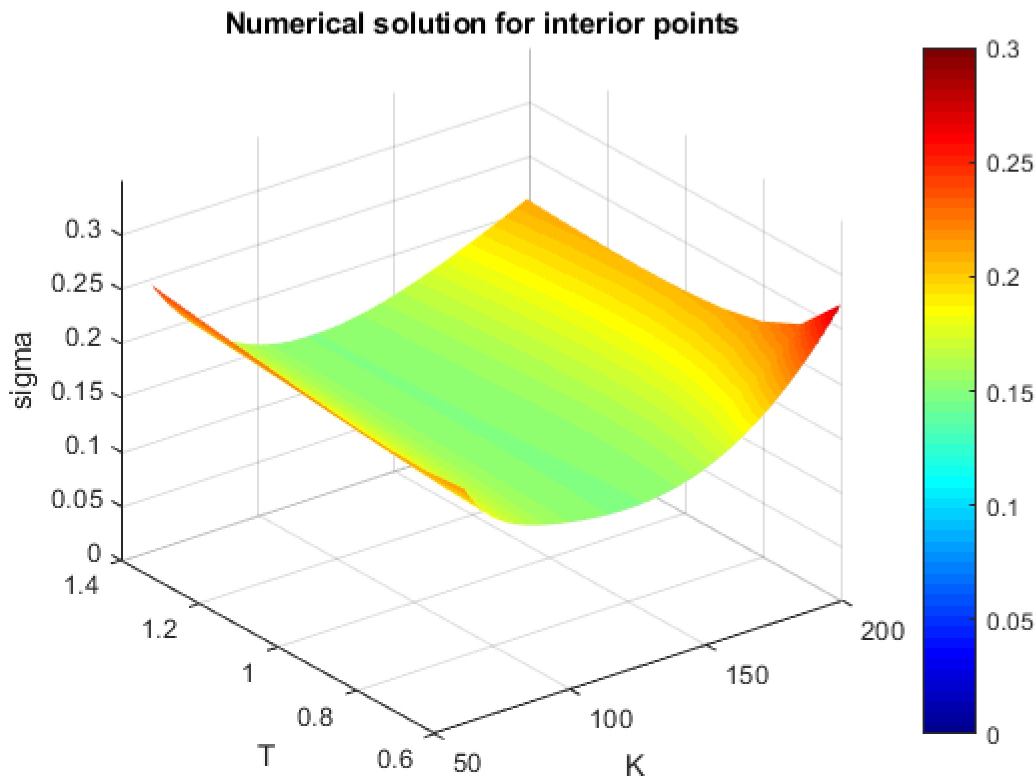
```
        end
    end
end

deltaK = 1;
for i=1:size(C,1)
    for j=1:size(C,2)
        if j==1
            dCdK2(i,j) = (2*C(i,j)-5*C(i,j+1)+4*C(i,j+2)-C(i,j+3))/
((deltaK)^2);
        elseif j==size(C,2)
            dCdK2(i,j) = (2*C(i,j)-5*C(i,j-1)+4*C(i,j-2)-C(i,j-3))/
((deltaK)^2);
        else
            dCdK2(i,j) = (C(i,j+1)-2*C(i,j)+C(i,j-1))/((deltaK)^2);
        end
    end
end

for i=1:size(dCdK,1)
    for j=1:size(dCdK,2)
        SigmaNum(i,j) =
 sqrt(abs(2*(dCdT(i,j)+(r-0)*K(j)*dCdK(i,j)+0*C(i,j))/
(K(j)*K(j)*dCdK2(i,j))));
    end
end

figure(2)
[Xnum1,Ynum1]=meshgrid(K,T);
surf(Xnum1,Ynum1,SigmaNum)
zlim([0 0.35])
xlabel("K")
ylabel("T")
zlabel("sigma")
title("Numerical solution")

caxis([0.0 0.3]);
colormap(jet)
shading interp
colorbar

%
% Numerical solution with implied volatilities. As we have mentioned
% before, the obtained volatilities were quite innacurate for boundary
% points and for larger values of trike price K, due to the fact
% that the values of C are so small there. So, the version of the
 Dupier
% formula with implied volatilities (that we will compute using the
 program
% that we developed in the previous assignment) will is used in this
 part.
% The resulting plot is shown at the end of the part 1 section.
%
```

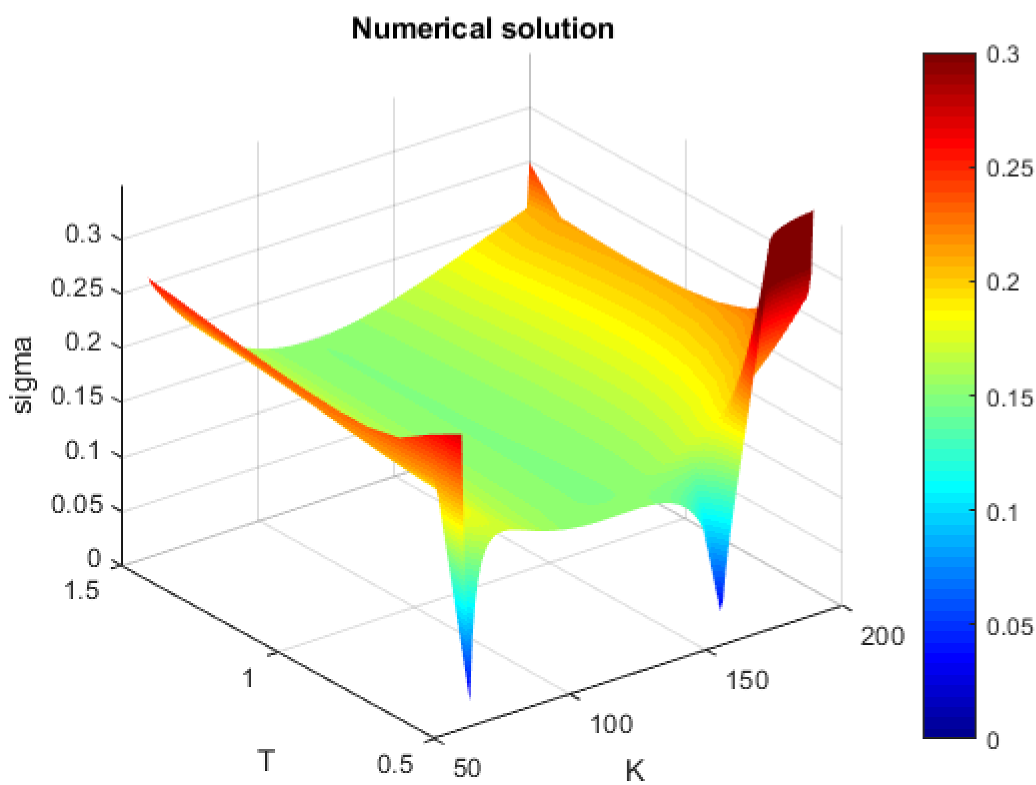
```
for i=1:size(C,1)
    for j=1:size(C,2)
        Timp = T(i);
        t = 0;

        myfun = @(sigmaImp,Cimp,Kimp,S0)
 normcdf(1/(sigmaImp*sqrt((Timp-t)))*(log(S0/
Kimp)+(r-0+0.5*sigmaImp*sigmaImp)*(Timp-t)))*S0*exp(-0*(Timp-
t))-normcdf(1/(sigmaImp*sqrt((Timp-t)))*(log(S0/
Kimp)+(r-0-0.5*sigmaImp*sigmaImp)*(Timp-t)))*Kimp*exp(-r*(Timp-t))-
Cimp;
        Kimp = K(j);
        Cimp = C(i,j);

        fun = @(sigmaImp) myfun(sigmaImp,Cimp,Kimp,S0);

        sigmaImp(i,j) = fzero(fun,0);
        d1(i,j) = 1/(sigmaImp(i,j)*sqrt(Timp))*(log(S0/
Kimp)+(r-0+0.5*sigmaImp(i,j)*sigmaImp(i,j))*Timp);
        d2(i,j) = 1/(sigmaImp(i,j)*sqrt(Timp))*(log(S0/
Kimp)+(r-0-0.5*sigmaImp(i,j)*sigmaImp(i,j))*Timp);
    end
end

for i=1:size(sigmaImp,1)
   for j=1:size(sigmaImp,2)
       if i==1 && j==1
           dsdT(i,j) = (sigmaImp(i+1,j)-sigmaImp(i,j))/(T(i+1)-T(i));
           dsdK(i,j) = (sigmaImp(i,j+1)-sigmaImp(i,j))/(K(j+1)-K(j));
       elseif i==1 && j<size(sigmaImp,2) && j>1
           dsdT(i,j) = (sigmaImp(i+1,j)-sigmaImp(i,j))/(T(i+1)-T(i));
           dsdK(i,j) = (sigmaImp(i,j+1)-sigmaImp(i,j-1))/(K(j+1)-
K(j-1));
       elseif i==1 && j==size(sigmaImp,2)
           dsdT(i,j) = (sigmaImp(i+1,j)-sigmaImp(i,j))/(T(i+1)-T(i));
           dsdK(i,j) = (sigmaImp(i,j)-sigmaImp(i,j-1))/(K(j)-K(j-1));
       elseif i>1 && i<size(sigmaImp,1) && j==1
           dsdT(i,j) = (sigmaImp(i+1,j)-sigmaImp(i-1,j))/(T(i+1)-
T(i-1));
           dsdK(i,j) = (sigmaImp(i,j+1)-sigmaImp(i,j))/(K(j+1)-K(j));
       elseif i>1 && i<size(sigmaImp,1) && j==size(sigmaImp,2)
           dsdT(i,j) = (sigmaImp(i+1,j)-sigmaImp(i-1,j))/(T(i+1)-
T(i-1));
           dsdK(i,j) = (sigmaImp(i,j)-sigmaImp(i,j-1))/(K(j)-K(j-1));
       elseif i==size(sigmaImp,1) && j==1
           dsdT(i,j) = (sigmaImp(i,j)-sigmaImp(i-1,j))/(T(i)-T(i-1));
           dsdK(i,j) = (sigmaImp(i,j+1)-sigmaImp(i,j))/(K(j+1)-K(j));
       elseif i==size(sigmaImp,1) && j>1 && j<size(sigmaImp,2)
           dsdT(i,j) = (sigmaImp(i,j)-sigmaImp(i-1,j))/(T(i)-T(i-1));
           dsdK(i,j) = (sigmaImp(i,j+1)-sigmaImp(i,j-1))/(K(j+1)-
K(j-1));
       elseif i==size(sigmaImp,1) && j==size(sigmaImp,2)
           dsdT(i,j) = (sigmaImp(i,j)-sigmaImp(i-1,j))/(T(i)-T(i-1));
           dsdK(i,j) = (sigmaImp(i,j)-sigmaImp(i,j-1))/(K(j)-K(j-1));
```

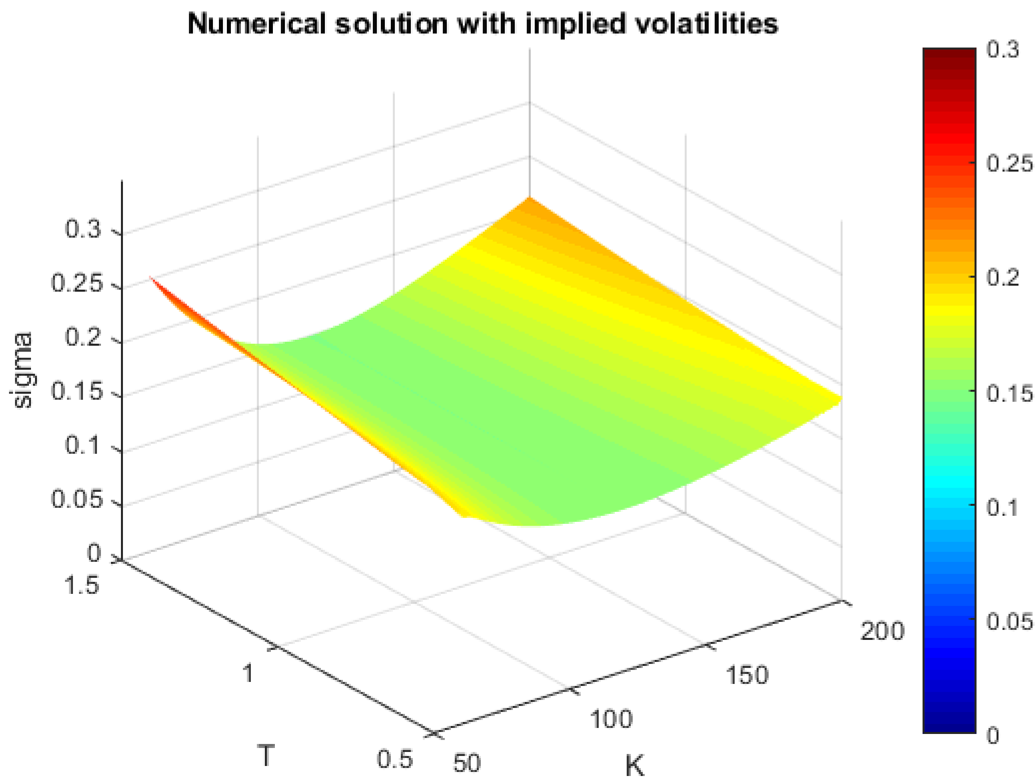
```
        else
            dsdT(i,j) = (sigmaImp(i+1,j)-sigmaImp(i-1,j))/(T(i+1)-
T(i-1));
            dsdK(i,j) = (sigmaImp(i,j+1)-sigmaImp(i,j-1))/(K(j+1)-
K(j-1));
        end
    end
end

deltaK = 1;
for i=1:size(sigmaImp,1)
    for j=1:size(sigmaImp,2)
        if j==1
            dsdK2(i,j) = (2*sigmaImp(i,j)-5*sigmaImp(i,j
+1)+4*sigmaImp(i,j+2)-sigmaImp(i,j+3))/(deltaK)^2;
        elseif j==size(C,2)
            dsdK2(i,j) =
 (2*sigmaImp(i,j)-5*sigmaImp(i,j-1)+4*sigmaImp(i,j-2)-
sigmaImp(i,j-3))/(deltaK)^2;
        else
            dsdK2(i,j) = (sigmaImp(i,j
+1)-2*sigmaImp(i,j)+sigmaImp(i,j-1))/(deltaK)^2;
        end
    end
end

for i=1:size(dsdK,1)
    for j=1:size(dsdK,2)
        SigmaNumImp(i,j) =
 sqrt((sigmaImp(i,j)^2+2*sigmaImp(i,j)*T(i)*(dsdT(i,j)+r*K(j)*dsdK(i,j)))/
(1+2*d1(i,j)*K(j)*sqrt(T(i))*dsdK(i,j)+K(j)*K(j)*T(i)*(d1(i,j)*d2(i,j)*dsdK(i,j)*d
    end
end

figure(3)
[Xnum2,Ynum2]=meshgrid(K,T);
surf(Xnum2,Ynum2,SigmaNumImp);
zlim([0 0.35])
xlabel("K")
ylabel("T")
zlabel("sigma")
title("Numerical solution with implied volatilities")

caxis([0.00 0.3])
colormap(jet)
shading interp
colorbar

%
% As it can be seen from the following plots, we can consider all of
 our
% results to be quite accurate for the interior points, where they are
% really close to the analytical solution (in fact the calibration
 using
```

```
% implied volatilities gave accurate results for all points (including
% those with large strike K). However, as mentioned before, the second
 plot
% shows some weird results that sould not be correct at all. We do not
 know
% the reason of them because we have checked the computed derivatives
 and
% they don't give any extrange results, so we think that it might be a
% problem with the data or with the formula that makes the numerical
 errors
% to explode around those points.
%
```




Numerical solution for interior points
sigma
0.3
0.25
0.2
0.15
0.1
0.05
0
T
1.4
1.2
1
0.8
0.6
K
50
100
150
200
0.3
0.25
0.2
0.15
0.1
0.05
0


Numerical solution
sigma
0.3
0.25
0.2
0.15
0.1
0.05
0
T
1.5
1
0.5
K
50
100
150
200
0.3
0.25
0.2
0.15
0.1
0.05
0



Part 2

In this part, we will calibrate the simply-compounded interest rates and swap rates using the Vasicek model.

In this part we calibrate the parameters θ = {k, ϕ, σ,r0} by calling the functions fminsearch and fminunc.

```
clear all;


x0 = [0.9572 0.4854 0.8003 0.1419];
%x0 = [1 1 1 1];
T1 = [1 7 30 60 90 180]; %in days
T1 = T1/360; % in years
Lmarket = [0.0300 0.0302 0.0307 0.0314 0.0320 0.0335];

T2 = [1 2 3 4 5 6 7 8 9 10]; %in years swap maturities time
Smarket = [0.0351 0.0368 0.0375 0.0379 0.0381 0.0383 0.0384 0.0385
 0.0385 0.0386];
swaptime = 0.25;
fun = @(x)cirmodel(x,T1,T2,Lmarket,Smarket,swaptime);
x1 = fminsearch(fun,x0); %from fminsearch
x2 = fminunc(fun, x0); %from fminunc

for i = 1:1:size(T1,2)
    tau(i) = T1(i) - 0;
    B(i) = (1 - exp(-x1(2)* tau(i)))/x1(2);
```

```
    A(i) = (x1(1) - (x1(3)*x1(3)/(2*x1(2)*x1(2)))) * (B(i) - tau(i)) -
 (x1(3)*x1(3)*B(i)*B(i)/(4*x1(2)));
    Z(i) = exp(A(i) - B(i)*x1(4));
    L(i) = (1 - Z(i))/(tau(i)*Z(i));
    tau2(i) = T1(i) - 0;
    B2(i) = (1 - exp(-x2(2)* tau(i)))/x2(2);
    A2(i) = (x2(1) - (x2(3)*x2(3)/(2*x2(2)*x2(2)))) * (B(i) - tau(i))
 - (x2(3)*x2(3)*B(i)*B(i)/(4*x2(2)));
    Z2(i) = exp(A(i) - B(i)*x2(4));
    L2(i) = (1 - Z(i))/(tau(i)*Z(i));
end

for j = 1:size(T2,2)
    tau(j) = T2(j) - 0;
    B(j) = (1 - exp(-x1(2)* tau(j)))/x1(2);
    A(j) = (x1(1) - (x1(3)*x1(3)/(2*x1(2)*x1(2)))) * (B(j) - tau(j)) -
 (x1(3)*x1(3)*B(j)*B(j)/(4*x1(2)));
    Z(j) = exp(A(j) - B(j)*x1(4));
    n = 1;
    for k = swaptime:swaptime:T2(j)
        Bj(n) = (1 - exp(-x1(2)*k))/x1(2);
        Aj(n) = (x1(1) - (x1(3)*x1(3)/(2*x1(2)*x1(2)))) * (Bj(n) - k)
 - (x1(3)*x1(3)*Bj(n)*Bj(n)/(4*x1(2)));
        Zj(n) = exp(Aj(n) - Bj(n)*x1(4));
        n = n + 1;
    end

    S(j) = (1 - Z(j))/(swaptime * sum(Zj));

    tau2(j) = T2(j) - 0;
    B2(j) = (1 - exp(-x2(2)* tau(j)))/x2(2);
    A2(j) = (x2(1) - (x2(3)*x2(3)/(2*x2(2)*x2(2)))) * (B(j) - tau(j))
 - (x2(3)*x2(3)*B(j)*B(j)/(4*x2(2)));
    Z2(j) = exp(A(j) - B(j)*x2(4));
    n = 1;

    for k = swaptime:swaptime:T2(j)
        Bj2(n) = (1 - exp(-x2(2)*k))/x2(2);
        Aj2(n) = (x2(1) - (x2(3)*x2(3)/(2*x2(2)*x2(2)))) * (Bj(n) - k)
 - (x2(3)*x2(3)*Bj(n)*Bj(n)/(4*x2(2)));
        Zj2(n) = exp(Aj(n) - Bj(n)*x2(4));
        n = n + 1;
    end

    S2(j) = (1 - Z(j))/(swaptime * sum(Zj));
end

my_fig = figure('position', [0, 0, 700, 500]);

subplot(2,1,1);
plot(T1*360,L*100);
hold on
plot(T1*360,L2*100,'g:');
plot(T1*360, Lmarket*100,'rd','MarkerSize',10);
```

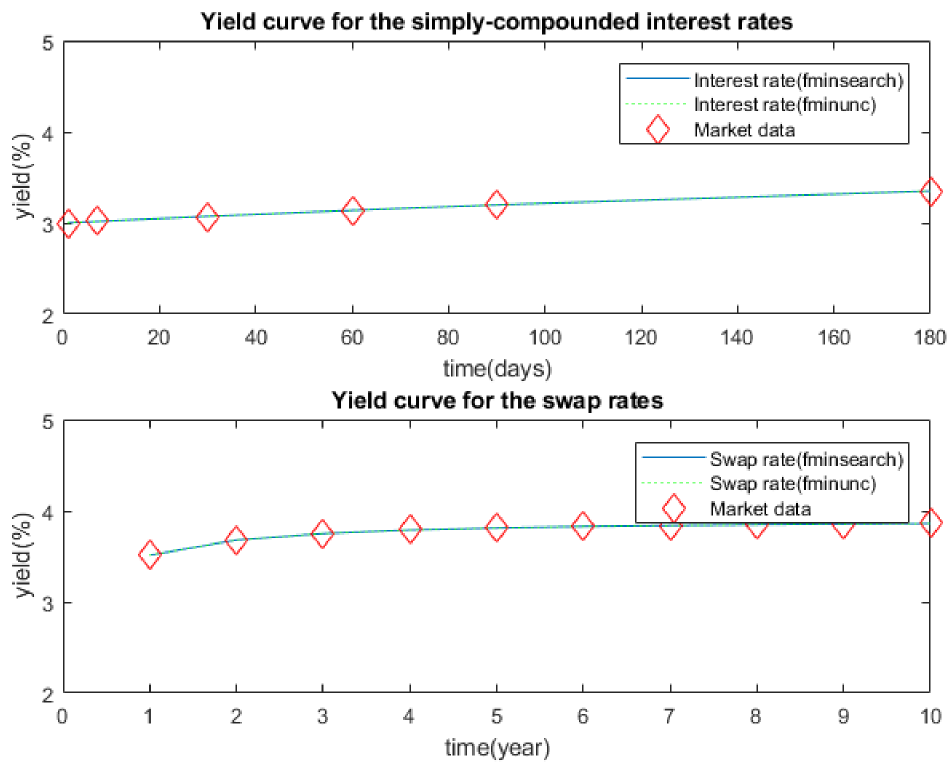
```
xlim([0 T1(end)*360]);
ylim([2 5]);
xlabel('time(days)');
ylabel('yield(%)');
legend('Interest rate(fminsearch)','Interest rate(fminunc)','Market
 data')
title('Yield curve for the simply-compounded interest rates');
hold off

subplot(2,1,2);
plot(T2,S*100);
hold on
plot(T2,S2*100,'g:');
plot(T2, Smarket*100,'rd','MarkerSize',10);
xlim([0 T2(end)]);
ylim([2 5]);
xlabel('time(year)');
ylabel('yield(%)');
legend('Swap rate(fminsearch)','Swap rate(fminunc)','Market data')
title('Yield curve for the swap rates');
hold off

%saveas(my_fig,'yield_plot2.png');

%
% The different initial guesses x0 will give different results when
 calling
% fminsearch and fminunc. But when calculating back the simply-
compouded
% interest rate and swap rates, they give the same result.
% This might be that the fminsearch and fminunc found local minimizers
 and
% the value are both close to 0.Tthe convergency rates are fast, but
% fminsearch use Nelder-Mead simplex method that does not require the
% minimizing problem to be differentiable and the error function can
 be
% discontinuous. While the fminunc function use the calssic Newton
 method
% assuming that it is differentiable. fminsearch tend to give more
 robust
% results while fminunc might get stucked at local minimum. Finally,
 we
% can confirm that the obtained results for both methods are very
 accurate
% as they match almost perfectly the provided market data this is
 shown in
% the plots at the end of the report.
%

dbtype('cirmodel.m');
```

Local minimum found.

Optimization completed because the size of the gradient is less than the default value of the optimality tolerance.

```
function y_out = cirmodel(x,T1,T2,Lmarket,Smarket,swaptime)
%here x is vector input,
%x(1) = phi
%x(2) = k
%x(3) = sigma (volatility)
%x(4) = r0(present interest rate)

%T1 = [1 7 30 60 90 180]; %in days
%T1 = T1/360;
%Lmarket = [0.0300 0.0302 0.0307 0.0314 0.0320 0.0335];%,market libor rate

%B = (1 - exp(-x(2)* tauti))/x(2);%need to get tauti

for i = 1:size(T1,2)
    tau(i) = T1(i) - 0;
    B(i) = (1 - exp(-x(2)* tau(i)))/x(2);
    A(i) = (x(1) - (x(3)*x(3)/(2*x(2)*x(2)))) * (B(i) - tau(i)) - (x(3)*x(3)*B(i)*B(i)/(4*x(2)));
    Z(i) = exp(A(i) - B(i)*x(4));
    L(i) = (1 - Z(i))/(tau(i)*Z(i));
    Ldiff2(i) = ((L(i) - Lmarket(i))/Lmarket(i))^2;
end

%T2 = [1 2 3 4 5 6 7 8 9 10];%in years swap maturities time
%Smarket = [0.0351 0.0368 0.0375 0.0379 0.0381 0.0383 0.0384 0.0385 0.0385 0.0386];

%swaptime = 3/12;

for j = 1:size(T2,2)

    tau(j) = T2(j) - 0;
    B(j) = (1 - exp(-x(2)* tau(j)))/x(2);
    A(j) = (x(1) - (x(3)*x(3)/(2*x(2)*x(2)))) * (B(j) - tau(j)) - (x(3)*x(3)*B(j)*B(j)/(4*x(2)));
    Z(j) = exp(A(j) - B(j)*x(4));
    n = 1;
    for k = swaptime:swaptime:T2(j)

        Bj(n) = (1 - exp(-x(2)*k))/x(2);
        Aj(n) = (x(1) - (x(3)*x(3)/(2*x(2)*x(2)))) * (Bj(n) - k) - (x(3)*x(3)*Bj(n)*Bj(n)/(4*x(2)));
        Zj(n) = exp(Aj(n) - Bj(n)*x(4));
        n = n + 1;

    end

```

```
        S(j) = (1 - Z(j))/(swaptime * sum(Zj));
        Sdiff2(j) = ((S(j) - Smarket(j))/Smarket(j))^2;
    end

    y_out = sum(Ldiff2) + sum(Sdiff2);
    end
```



Published with MATLAB® R2017b