FIN 503: Advanced derivatives **(Due: 27/10/20)**

Assignment #5

Professor: Elena Perazzi *Assistant:* Marc-Aurèle Divernois

Students: Hien Lê, Francesco Maizza, Anita Mezzetti

The goal of this exercise is to build the implied volatility surface of STOXX50E the index using the Andersean-Huge algorithm.

Initialisation
We have been given data consisting of Market Implied Volatilities corresponding to a set of strikes (one strike for each line) and a set of maturities (one column for each maturity). In particular, we have 161 strikes, varying from 0.4 to 2, and 7 maturities:

$$\texttt{lenk} = 161 \qquad \text{and} \qquad \texttt{lent} = 7.$$

We define `lenk` as global variable.

After importing the data, we set the current spot price S0 to 2272.70. We found this value on the *Volatility Interpolation* paper. Then, we extract from the data the vector of strikes `K`, the vector of maturities `T` and the matrix `vol` containing the corresponding Market Implied Volatilities. We also create the vector `dK`, in which each element i corresponds to $k(i+1)-k(i)$. Similarly, we build `dT`. We also set r and q to zero.

We have that

$$\sigma(S_t, t) = \frac{\widetilde{\sigma}(S_t, t)}{S_t}$$

For $t = T$, $S_t = K$. Hence,

$$\widetilde{\sigma}(K, T) = \sigma(K, T) \cdot K.$$

This is true for each (K,T) in `(K,T)`. So, we can find the matrix corresponding to the **observed** $\widetilde{\sigma}$, called `vol_tilde`. It is observed because it is calculated using the observed volatilieties.

Observed Call Prices
Now, we find the call prices using the Black Scholes formula. The Black Scholes formula for a call price is implemented in a separated function called `CallBS`. First, we initialise the matrix `call_obs` sized `(lenk,lent)` to zero. Then we fill this matrix element by element using `CallBS`, checking that the corresponding `vol` is positive. If `vol` is not positive, we leave the call price zero. The resulting call prices are saved in the matrix `call_obs`.

Implied Volatilities
For this part we need to keep in mind that the Andersean-Huge algorithm uses the Dupire forward PDE for option prices to recursively generate option prices for expiration $T + dT$ given the prices for expiration T. We remind that the general Dupire forward equation is

$$\frac{dC}{dT} + (r - q)K\frac{dC}{dK} + qC - \frac{1}{2}\widetilde{\sigma}^2(K, T)\frac{d^2C}{dK^2} = 0.$$

In our case,

$$\frac{dC}{dT} - \frac{1}{2}\widetilde{\sigma}^2(K,T)\frac{d^2C}{dK^2} = 0. \tag{0.1}$$

To solve this equation, let us discretise the space of T and K. In other words, we solve the problem for each point (K_i, T_j). Thanks to that, we can pass from derivatives to finite increments. Equation 0.1 becomes:

$$\frac{C_{i,j+1} - C_{i,j}}{dT(j)} - \frac{1}{2}\widetilde{\sigma}^2_{i,j+1}\frac{C_{i+1,j+1} - 2C_{i,j+1} + C_{i-1,j+1}}{dK(i)^2} = 0. \tag{0.2}$$

We can rearrange terms in order to find $C_{i,j+1}$ from $C_{i,j}$ for each i, using $\widetilde{\sigma}$. Working with vectors, $C(T_j)$ is the vector containing all the prices at time T_j for all the 161 strikes. We can find $C(T_j + 1)$ from $C(T_j)$. In particular, we need to define a matrix, called A_j, such that

$$A_{j+1}C(T_j + 1) = C(T_j). \tag{0.3}$$

The structure of this matrix can be found from Equation 0.2 rearranging terms. We create a function, called `build_A`, which is in charge of creating this matrix. Specifically, in this function, whose parameters are $\widetilde{\sigma}$, dT and dK, we create the vector $z(j+1) = (z_{i,j+1})_{i=1:lenk-1}$:

$$z_{i,j+1} = \frac{1}{2}\frac{dT(j)}{dK(i)^2}\widetilde{\sigma}^2_{i,j+1}. \tag{0.4}$$

From these $z_{i,j+1}$, we build the matrix A as explained is slide 14, using the MATLAB function `diag`. Thanks to this function, we can find $C(T_j + 1)$ as

$$C(T_j + 1) = A^{-1}_{j+1}C(T_j). \tag{0.5}$$

There is one problem in this solution: we do not know the theoretical $\widetilde{\sigma}$ (we only have the observed one, used to find the observed call prices with BS). Therefore, we need to estimate it. To do so, let us assume we have M call prices from the market for the expiration T_{j+1} and assume $\widetilde{\sigma}$ depends on M parameters. We need to fit these parameters so that the option prices generated by the previous description, $C^{\text{model}}(T_j+1) = A^{-1}_{j+1}C^{\text{model}}(T_j)$, for the observed strikes match the observed prices, C^{observed}, saved in `call_obs`. In other words, we need to minimise the sum

$$\sum_{n=1}^{j}(C^{\text{model}}(K_n, T_j + 1) - C^{\text{observed}}(K_n, T_j + 1))^2 \tag{0.6}$$

Therefore, we need to create the function to minimise and pass it to the MATLAB method `fminsearchcon` [1], which computes constrained optimisation.
We name the function `optimization_function` and it returns the "object" to minimise. Inside it, we find the interpolated volatilities needed to find the $z_{i,j+1}$ and, consequently, matrix A. The function in charge of this is called `volatility_interpolation` and its parameters are the positive observed call prices and the parameters that must be chosen in order to minimize Equation 0.6. This function uses the MATLAB `interp1` method to find the `interpolated_vol`, that we will use to build A using `build_A`. At this point. we can invert A, use A^-1 in Equation 0.5 and return the function to minimise.

As anticipated, we can finally proceed to use `fminsearchcon` and find the right parameters. To the optimisation MATLAB method, we pass also the upper and lower bound. We use again the

[1] We add this function to the folder but we found it already implemented. We have also already used it for other courses (Optimization Methods and Computational Finance).

`volatility_interpolation` method with this parameters and we have optimised $\tilde{\sigma}$, saved in `esimated_vol`. Following the same steps as above, using A, we find the call prices. We collect them in a function called `result_C`.

We can use `result_C` to find the implied volatility thanks to the `implied_volBS` method. Remind that the implied volatility is the volatility we have to insert in the Black Scholes model in order to reproduce the observed price. Therefore, it is the volatility for which the difference between the two prices is zero. See slide 8 of Lecture 2 for more details. We save the implied volatilities in the matrix `IV`.

Implied volatility surface
Running the `ass5`, which follows the previous described steps, for the implied volatility matrix `IV` we get the surface in Figure 1.

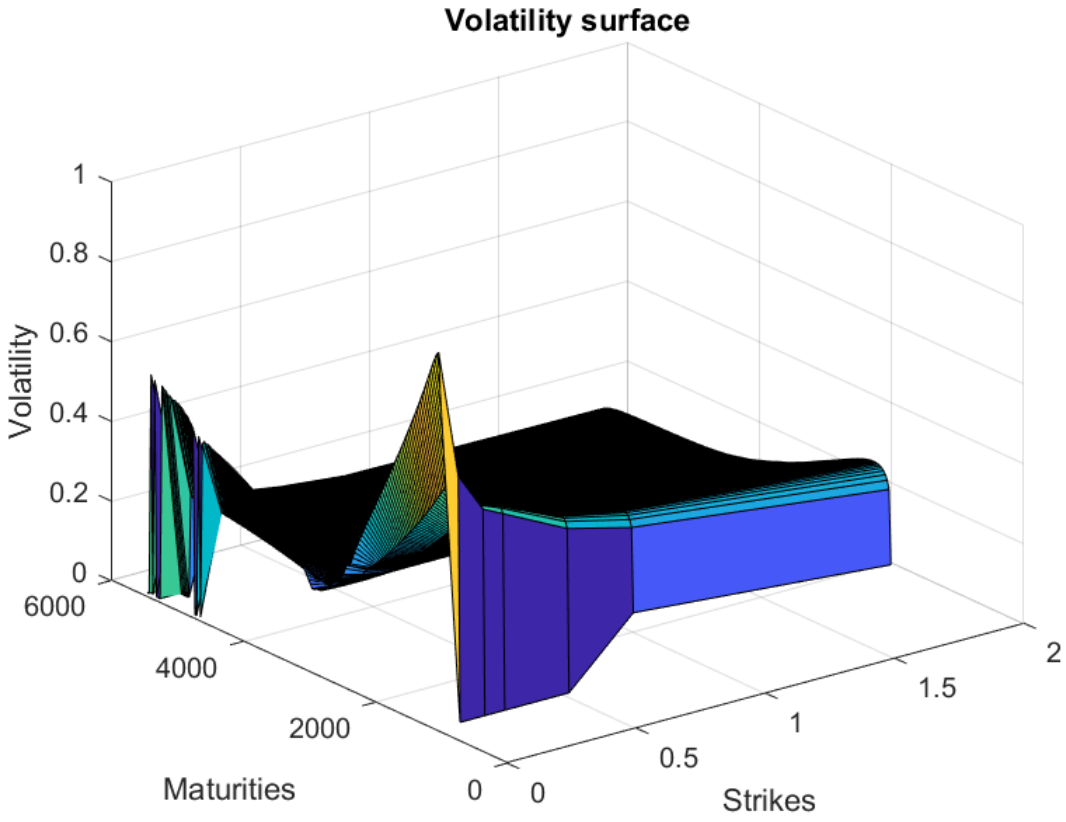

Figure 1: Implied volatility surface

Implied volatility for two other expiration
We need to find the implied volatility also for the expirations T = 1 and T = 1.5. The steps are the same for both. Fist we need to find what is, in the vector of observed maturities `T`, the last previous maturity T_j, in order to find dT. Then we extract the volatilities corresponding to T_j, from the already calculated `esimated_vol`, and we use them to find A and the estimated call prices. The resulting matrix, called `C_models` and sized 161x2, contains all the call prices corresponding to these two maturities for all the possible stock prices. Finally, we find the implied volatilities as above, calling the function `implied_volBS` and passing `C_models` to it. The results are saved in `IV_new_mat`.

```
% Advanced Derivatives - problem set 5
% Hien Le, Francesco Maizza, Anita Mezzetti

clc
clear all
close all

global lenk

%% Inizialization:

data = xlsread('SX5E_Impliedvols.xlsx');  %load data

S0 = 2772.70;                % current spot given by the paper

K = data(:,1) * S0;          % strikes from 40% to 200%
K = K(~isnan(K));            % delete NaN values

T = data(1,:);               % Maturity dates
T = T(~isnan(T));            % delete NaN values

lenk = length(K);            % number of strikes
lenT = length(T);            % number of maturity dates

dK = K(2)-K(1);              % vector dk(i) = k(i+1)-k(i)
dT = [T(1),diff(T)];         % vector dT(i) = T(i+1)-T(i)

% sempifications:
r = 0;                       % interst rate
q = 0;                       % dividend

vol = data(2:end,2:end);     % volatility data
positive_vol = vol > 0;      % true/false matrix: positive => 1, zero↙
=> 0

vol_tilde = diag(K) * vol;   % vol_tilde(K,T) = K * vol(K,T)

% Call Prices: algorithm to extrapolate call prices using the BS↙
formula:
call_obs = zeros(lenk, lenT); % initialise the observed call prices↙
matrix


for i = 1:lenk
    for j = 1:lenT
        if (positive_vol(i,j) > 0) % otherwhise is not a stochastic↙
process
            % find the call price for maturity T(j) and strike K(i):
            call_obs(i,j) = CallBS(S0, K(i), r, T(j), vol(i,j), q);
```

```
        end
    end
end

%% Implied Volatility for the expirations in the spreadsheet:

C0 = max(S0-K, 0);                  % initial call price for each K
result_C = zeros(lenk, lenT);       % resulting C after the AH↙
algorithm
esimated_vol = zeros(size(vol));    % estimated volatility through↙
interp.
C_actual = C0;

for j = 1:length(T)

    % find nonzero vol for all vol tilde of that maturity (one row for↙
each T):
    pos_vol_T = find((vol_tilde(:,j)>0));

    %first guess corresponding to the observed sigma volatilities
    para0 = nonzeros(vol_tilde(:,j));

    %display(length(para0))

    % sigma lower and upper bounds for the optimization method
    lb = zeros(1,length(para0));        % sigma > 0
    up = S0 * ones(1,length(para0));    % sigma < S0

    % Optimization
    [parameters,fval,~,~] = fminsearchcon (@(parameters)↙
optimization_function...
        (C_actual,call_obs(:,j),dT(j),dK,parameters),para0,lb,up);

    % Prediction (similar steps to optimization_function)
    pos_call_obs = find(call_obs(:,j)); % positions in which call_obs↙
not zero

    % Estimated Volatility
    interp_vol = volatility_interpolation (pos_call_obs, parameters);
    esimated_vol(:,j) = interp_vol;

    inv_A = pinv(build_A(interp_vol,dT(j),dK)); % A^(-1) =↙
psudoinverse of A
    % inv_A>=0 (showed by Nabben) imples that the discrete system is↙
stable
    if inv_A < 0
        fprintf("The discrete system is not stable")
    end
    C_next = inv_A * C_actual;
```

```
    C_actual = C_next;              % update the currenr call price
    result_C(:,j) = C_next;         % update result_C

end

%% Implied Volatility: (from the computed call prices)
sigma0 = 0.02;                  % initial value, pr 0.05
[IV, ~] = implied_volBS(S0, K, r, T, q, result_C, sigma0);

%% Volatility Surface Plot:

figure
surf(T, K, IV)
title('Volatility surface')
xlabel('Strikes')
ylabel('Maturities')
zlabel('Volatility')


%% Call Prices and Implied Volatility for T=1 and T=1.5
TT = [1, 1.5];
C_new_maturities = zeros(lenk, length(TT));  % resulting C

for i = 1:length(TT)

    % first time T exceeds TT(i) minus 1 => last time T lower that TT(i)
    j = find(T > TT(i), 1) - 1;       % T_prime between T_j and T_{j+1}
    dT = TT(i) - T(j);
    sigma = esimated_vol(:,j);        % volatilities of that moment
    inv_A = pinv(build_A(sigma, dT, dK));
    C_new_maturities(:,i) = inv_A * result_C(:,j);
end

% Implied Volatility:
[IV_new_mat, ~] = implied_volBS(S0, K, r, TT, q, C_new_maturities, sigma0);

%% Functions:

function f = optimization_function (C_actual,call_obs,dT,dK,para)
% return the function to minimize
% C0 = initial call price
% C_actual = option price ad current step
% call_obs = observed call prices for that T
% lenk = number of strikes and length of C0
```

```
pos_call_obs = find(call_obs); % positions in which call_obs not zero

% Volatility Interpolation:
interpolated_vol = volatility_interpolation (pos_call_obs, para);

% C(T(j+1)) = A^(-1) * C(T(j)u)
inv_A = pinv(build_A(interpolated_vol,dT,dK)); % A^(-1) = psudoinverse↙
of A

% inv_A>=0 (showed by Nabben) imples that the discrete system is↙
stable
if inv_A<0
    fprintf("The discrete system is not stable")
end
%display(inv_A)
%display(C_actual)
%display(size(inv_A))
%display(size(C_actual))

C_next = inv_A * C_actual;              % call price next step

% function to minimize:
f = sum((C_next(pos_call_obs) - call_obs(pos_call_obs,1)).^2);
end

function sigma = volatility_interpolation (positions, sigma_in)
% returns interpolated values of a 1-D function at specific query↙
points
% using linear interpolation
% positions: sample pointsc
% sigma_est: corresponding values of the sample points
% sigma: result

global lenk

% Interpolation of sigmas between the volatilities.
sigma = interp1(positions, sigma_in, (1: lenk)', 'nearest');
% 'nearest': nearest neighbor interpolation
% For a point where the volatility is now known yet, it uses the
% nearest observed value as image, which is in line with the problem.

% the length of sigma is always equal to the number of strikes (lenk)

% all the sigmas before the first estimated sigma set to the first
% estimated
sigma(1:positions(1)) = sigma(positions(1));
% all the sigmas after the last estimated sigma set to the last
% estimated
sigma(positions(end):end) = sigma(positions(end));
```

```
if length(sigma) ~= lenk
    fprintf("Error in the interpolation part");
end

end

function A = build_A (sigma,dT,dK)
% A is a tri-diagonal matrix, diagonally dominante with positive↙
diagonal
% and negative off-diagonals

% define Z as written in explained in slide 12 of lecture 4
Z = 0.5 * dT / (dK)^2 * sigma(2:end).^2;% definition of Z
Z = Z(1:end-1)';                                % skip last value


diag_0 = [1, 1+2*Z, 1];                         % principal diagonal
diag_up1 = [0, -Z];                             % upper diagonal
diag_less1 = [-Z, 0];                           % lower digonal

A = diag(diag_0, 0) + diag(diag_up1,1) + diag(diag_less1,-1);
end
```

```
function [C] = CallBS(S,K,r,T,sigma,q)
% CallBS finds the price of a Call option using BS
% We suppose t=0 => T-t = T
% See slides 19/20 Lecure 1

d1 = (log(S/K) + (r - q + sigma^2 / 2)*(T))/(sigma*sqrt(T));
d2 = d1 - sigma*sqrt(T);

C =  S*exp(-q*T)*normcdf(d1) - K*exp(-r*T)*normcdf(d2);
end
```

```
function [implied_vol,fval] = implied_volBS(S,K,r,T,q,C,sigma0)
%This function finds the implied volatility (slide .8 lecture 2)
%The implied vol is the vol I have to insert in the BS model in order to
%reproduce the observed price. Therefore it is the volatility for which the
%difference between the two prices is zero.
% C = observed prices
% implied_vol = implied vol
% fval =

implied_vol = zeros(size(C));

%find the vol such that: call_price_BS - observed_call_price = 0

for i = 1:length(K)
    for j = 1:length(T)

        % function which must be zero:
        funct = @(impl_vol) (CallBS(S,K(i),r,T(j),impl_vol,q) - C(i,j));
        % find the zero:
        [implied_vol(i,j),fval(i,j)] = fzero(funct,sigma0);
    end
end

if fval ~= zeros(size(fval))
    fprint("Error in finding the implied volatility")
end

end
```