CS 476: Assignment 3

Daniel Matheson: 20270871

Question 1

Consider the loss function $f(x,y)$, induced by the random variable y with density $p(y)$. Assume that the cumulative distribution function of $f(x,y)$ is continuous. Let the extended function for CVaR be:

$$F_\beta(x,\alpha) = \alpha + (1-\beta)^{-1}\int_{y\in\mathbb{R}^m}[f(x,y)-\alpha]^+ p(y)dy = \alpha + \frac{1}{1-\beta}\mathbf{E}([f(x,y)-\alpha]^+)$$

where

$$[z]^+ = \begin{cases} z & \text{if } z > 0 \\ 0 & \text{otherwise} \end{cases}$$

Let $([z]^+)'(d)$ denote the directional derivative along d, i.e

$$([z]^+)'(d) = \lim_{t\to 0^+}\frac{([z+td]^+) - ([z]^+)}{t}$$

Assume that under the assumed distribution the following property holds:

$$(\mathbf{E}([z]^+))'(d) = \mathbf{E}(([z]^+)'(d))$$

Show that $F_\beta(x,\alpha)$ is convex and continuously differentiable with respect to α.

Solution:

$$\frac{\partial F_\beta}{\partial \alpha} = 1 + \frac{1}{1-\beta}\frac{\partial}{\partial\alpha}\mathbf{E}\big[(f(x,y)-\alpha)^+\big]$$

$$= 1 + \frac{1}{1-\beta}\mathbf{E}\big[\frac{\partial}{\partial\alpha}(f(x,y)-\alpha)^+\big] \quad \text{by assumption above}$$

now note that $\frac{\partial}{\partial\alpha}(f(x,y)-\alpha)^+ = \begin{cases} -1 & \text{if } f(x,y) > \alpha \\ 0 & \text{otherwise} \end{cases}$ and hence;

$$= 1 + \frac{1}{1-\beta}(-1)\mathbf{P}[f(x,y) > \alpha]$$

$$\implies \frac{\partial F_\beta}{\partial\alpha} = 1 - \frac{1}{1-\beta}\Big[1 - \mathbf{P}[f(x,y)\le\alpha]\Big] \quad \text{cts since CDF is cts and } \beta\in[0,1)\text{, so } F_\beta \in C^1$$

$$\implies \frac{\partial^2 F_\beta}{\partial\alpha^2} = \frac{\partial}{\partial\alpha}\frac{1}{1-\beta}\mathbf{P}[f(x,y)\le\alpha] \quad \text{and since we assumed CDF of } f(x,y) \text{ is continuous:}$$

Let $f(x,y) = Z$ and $G(Z)$ be the CDF of $f(x,y) = Z$ (continuous) then we have

$$= \frac{1}{1-\beta}g(\alpha) \quad \text{where } g(z) \text{ is the PDF of } f(x,y) = Z \text{ (by FTC)}$$

$$> 0 \quad \text{since } g(\alpha)\in[0,1] \text{ and } \beta\in[0,1)\text{, so } F_\beta \text{ is convex} \qquad \square$$

Question 2
Consider the equality constrained least-squares problem:

$$\begin{array}{rl} \min & ||A_x - b||_2^2 + \frac{\rho}{2} x^\tau x \\ \text{subject to} & Gx = h \end{array}$$

where $\rho > 0$ is a constant, $A \in \mathbb{R}^{m\times n}$ and $G \in \mathbb{R}^{p\times n}$ with $\operatorname{rank}(G) = p$. Describe the KKT conditions, and derive expressions for the primal solution $x^\star$ and the dual solution $\nu^\star$
Solution:
Rewrite the linear constraint as $q_j(\vec{x}) = G_j\vec{x} - h_j$ where G_j is the j^{th} row of G. Since $\operatorname{rank}(G) = p$ there is no need to be concerned about this function being well defined.
and we let $f(\vec{x}) = ||A\vec{x} - \vec{b}||_2^2 + \frac{\rho}{2}\vec{x}^\tau\vec{x}$
Then the Lagrangian function is

$$\mathcal{L}(\vec{x}, \vec{\nu}) = f(\vec{x}) + \sum_{j=1}^{n} \nu_j q_j(\vec{x})$$

The *KKT Conditions* are:

$$\begin{cases} \text{Primal Feasibility:} & q_j(\vec{x}) = 0 \\ \text{Dual Feasibility:} & \lambda \geq 0 \quad \text{(not in our problem)} \\ \text{Gradient of zero:} & \nabla_{\vec{x}}\mathcal{L}(\vec{x}, \vec{\nu}) = 0 \\ \text{Complementarity:} & \lambda_i^\star g_i(\vec{x}^\star) = 0 \quad \text{(not in our problem)} \end{cases}$$

We must satisfy the first and third conditions:

$$\begin{cases} G_j\vec{x} - h_j = 0 \ \forall\ j = 1, \ldots, n \\ \nabla_{\vec{x}}\mathcal{L}(\vec{x}, \vec{\nu}) = \nabla f(\vec{x}) + \sum_{j=1}^{n} \nu_j \nabla q_j(\vec{x}) = \nabla\Big(||A\vec{x} - \vec{b}||_2^2 + \frac{\rho}{2}\vec{x}^\tau\vec{x}\Big) + \sum_{j=1}^{n} \nu_j \nabla\Big(G_j\vec{x} - h_j\Big) = 0 \end{cases}$$

We will find the gradient of $\mathcal{L}$ one component at a time:

$$\begin{aligned} \nabla\Big(||A\vec{x} - \vec{b}||_2^2 + \frac{\rho}{2}\vec{x}^\tau\vec{x}\Big)_i &= \nabla\Big(\sum_{k=1}^{n}(A_k\vec{x} - b_k)^2 + \frac{\rho}{2}\sum_{k=1}^{n} x_k^2\Big)_i \\ &= \nabla\Big(\sum_{k=1}^{n}(\Big[\sum_{\ell=1}^{n} A_{k\ell}x_\ell\Big] - b_k)^2 + \frac{\rho}{2}\sum_{k=1}^{n} x_k^2\Big)_i \\ &= \frac{\partial}{\partial x_i}\Big(\sum_{k=1}^{n}(\Big[\sum_{\ell=1}^{n} A_{k\ell}x_\ell\Big] - b_k)^2 + \frac{\rho}{2}\sum_{k=1}^{n} x_k^2\Big) \\ &= \Big(\sum_{k=1}^{n} 2(A_k\vec{x} - b_k)(A_{ki})\Big) + \rho x_i \\ \implies \nabla f(\vec{x}) &= 2A^T(A\vec{x} - \vec{b}) + \rho\vec{x} \end{aligned}$$

$$
\begin{aligned}
\nabla\Big(\sum_{j=1}^{n}\nu_j G_j\vec{x}-h_j\Big)_i &= \frac{\partial}{\partial x_i}\Big(\sum_{j=1}^{n}\nu_j G_j\vec{x}-h_j\Big)\\
&= \sum_{j=1}^{n}\nu_j\frac{\partial}{\partial x_i}\Big(\sum_{\ell=1}^{n}G_{j\ell}x_\ell\Big)\\
&= \sum_{j=1}^{n}\nu_j G_{ji} = (G^T\nu)_i\\
\implies \nabla\Big(\sum_{j=1}^{n}\nu_j q_j(\vec{x})\Big) &= G^T\vec{\nu}
\end{aligned}
$$

Putting these two together we get:

$$
\begin{aligned}
\nabla\mathcal{L}(\vec{x},\vec{\nu}) &= \nabla f(\vec{x}) + \sum_{j=1}^{n}\nu_j\nabla q_j(\vec{x})\\
&= 2A^T(A\vec{x}-\vec{b}) + \rho\vec{x} + G^T\vec{\nu}\\
&= (2A^TA+\rho I)\vec{x} - 2A^T\vec{b} + G^T\vec{\nu}
\end{aligned}
$$

and since A^TA is positive semifdefinite, and $\rho > 0$, $2A^TA+\rho I$ is positive definite and therefore it is invertible, so we have:

$$\vec{x} = \Big(2A^TA+\rho I\Big)^{-1}\Big(2A^T\vec{b}-G^T\vec{\nu}\Big) \quad \textbf{(1)}$$

and now recall the restriction $G\vec{x} = h$:

$$
\begin{aligned}
&\implies G\Big(2A^TA+\rho I\Big)^{-1}\Big(2A^T\vec{b}-G^T\vec{\nu}\Big) = h\\
&\implies G\Big(2A^TA+\rho I\Big)^{-1}2A^T\vec{b} - G\Big(2A^TA+\rho I\Big)^{-1}G^T\vec{\nu} = h\\
&\implies \underbrace{G\Big(2A^TA+\rho I\Big)^{-1}G^T}_{p\times p\text{ matrix w rank }p\text{, invertible}}\ \vec{\nu} = G\Big(2A^TA+\rho I\Big)^{-1}2A^T\vec{b} - h\\
&\implies \nu^\star = \Big(G\Big(2A^TA+\rho I\Big)^{-1}G^T\Big)^{-1}\Big(G\Big(2A^TA+\rho I\Big)^{-1}2A^T\vec{b}-h\Big)
\end{aligned}
$$

This is the solution to the Dual Problem. For the Primal problem we plug in $\nu^\star$ into (1):

$$
\begin{aligned}
\vec{x} &= \Big(2A^TA+\rho I\Big)^{-1}\Big(2A^T\vec{b}-G^T\vec{\nu}\Big)\\
x^\star &= \Big(2A^TA+\rho I\Big)^{-1}\Big(2A^T\vec{b}-G^T\Big(G\Big(2A^TA+\rho I\Big)^{-1}G^T\Big)^{-1}\Big(G\Big(2A^TA+\rho I\Big)^{-1}2A^T\vec{b}-h\Big)\Big)
\end{aligned}
$$

to make things clearer, let $Y = \Big(2A^TA+\rho I\Big)^{-1}$

$$\implies x^\star = Y\Big(2A^T\vec{b}-G^T\Big(GYG^T\Big)^{-1}\Big(GY2A^T\vec{b}-h\Big)\Big)$$

Question 3(a) Part 2 / Also used in Question 5

```
function V_0 = fin_diff(S, sigma,r, T, N, stepmethod, payoff)
% This function takes the following inputs:
% S: the grid of stock prices
% sigma: the volatility function
% r: risk free rate
% T: option expiry
% N: number of time steps
% stepmethod: one of 'Fully Implicit', 'Crank-Nicolson', or 'Rannacher'
% payoff: payoff function of the option
% And calculates the option values at the given points of S

deltaTau = T/N;
num_S = length(S);
alpha = zeros(num_S - 2,1);
beta = zeros(num_S - 2,1);

for i=2:(num_S - 1)
    % Central Difference Method
    alpha(i-1) = ((sigma(S(i))*S(i))^2)/((S(i)-S(i-1))*(S(i+1)-S(i-1))) - r*S(i)/(S(i+1)-S(i-1));
    beta(i-1) =  ((sigma(S(i))*S(i))^2)/((S(i+1)-S(i))*(S(i+1)-S(i-1))) + r*S(i)/(S(i+1)-S(i-1));
    if ((alpha(i-1) < 0) || (beta(i-1) < 0))
        % Forward Difference Method
        alpha(i-1) = ((sigma(S(i))*S(i))^2)/((S(i)-S(i-1))*(S(i+1)-S(i-1)));
        beta(i-1) =  ((sigma(S(i))*S(i))^2)/((S(i+1)-S(i))*(S(i+1)-S(i-1))) + r*S(i)/(S(i+1)-S(i));
        if ((alpha(i-1) < 0) || (beta(i-1) < 0))
            % Backward Difference Method
            alpha(i-1) = ((sigma(S(i))*S(i))^2)/((S(i)-S(i-1))*(S(i+1)-S(i-1)))...
                - r*S(i)/(S(i) - S(i-1));
            beta(i-1) = ((sigma(S(i))*S(i))^2)/((S(i+1)-S(i))*(S(i+1)-S(i-1)));
        end
    end
end

% Creating the 3 diagonals needed for M
lower_diag = [- deltaTau * alpha; 0; 0];
middle_diag = [r* deltaTau; deltaTau * (alpha + beta + r) ; 0];
upper_diag = [0; 0; - deltaTau * beta];

% Putting together the sparse diagonal matrix M
M = spdiags([lower_diag middle_diag upper_diag], [-1 0 1], num_S, num_S);

theta = 0;
if strcmp(stepmethod, 'Crank-Nicolson') % set theta = 0.5 if C-N method
    theta = 0.5;
end

IM = speye(num_S) + (1 - theta)* M; % LHS matrix
RHS_IM = speye(num_S) - theta * M;  % RHS matrix

[L, U] = lu(IM); % LU factorization only once
V = payoff(S)';  % Set V to payoff to begin

for j = 1:N
    % This if command is for the Rannacher method; it will run fully
    % Implicit method twice and then change over to C-N (theta = 0.5), and
    % then recalculate the LU factorization
```

```
    if strcmp(stepmethod, 'Rannacher') && (j == 3)
        theta = 0.5;
        IM = speye(num_S,num_S) + (1 - theta)* M;
        RHS_IM = speye(num_S,num_S) - theta * M;
        [L, U] = lu(IM);
    end
    % Solving [I+(1-theta)M](V^{n+1}) = [I - theta*M]V^n
    V = U\(L\(RHS_IM * V));

end

V_0 = V;

end
```

Published with MATLAB® R2016a

Contents

Question 3(a) Part 1

```
function ansT = fin_diff_table_graph()
% This function creates the tables and graphs for Question 3

% Parameters and constants
alpha = 15;
r = 0.025;
T = 0.5;
K = 100;
S_0 = 100;
N_0 = 25;

% Initial grid of prices
S = [0:0.1*K:0.4*K,...
0.45*K:0.05*K:0.8*K,...
0.82*K:0.02*K:0.9*K,...
0.91*K:0.01*K:1.1*K,...
1.12*K:0.02*K:1.2*K,...
1.25*K:.05*K:1.6*K,...
1.7*K:0.1*K:2*K,...
2.2*K, 2.4*K, 2.8*K,...
3.6*K, 5*K, 7.5*K, 10*K];

% Setting volatility function and payoff function
sigma = @(x)(alpha/x);
payoff = @(x)(max(K - x, 0));

numsteps = 5; % number of times to refine grid of prices
Nlist = zeros(numsteps,1); % keeps track of number of timesteps
Snodeslist = zeros(numsteps,1); % keeps track of the length of S

V = zeros(3,numsteps);   % initializing vector for option values
methods = {'Fully Implicit', 'Crank-Nicolson', 'Rannacher'};

N = N_0;
for j = 1:numsteps
    % Loop through numsteps, find the new option value for each method
    % then refine S
    Snodeslist(j) = length(S);
    for i = 1:3 % find the option value with finite difference for each
                % stepping method
       method = methods(i);
       values = fin_diff(S, sigma,r, T, N, method, payoff);
       V(i,j) = values(S == 100); % take the value for S = 100
    end

    if j < numsteps % refine the grid
       insert = (S(2:end) + S(1:end-1))/2 ;
       new_S = zeros(length(S) + length(insert),1);
       new_S(1:2:end) = S;
       new_S(2:2:end-1) = insert;
       S = new_S';
       N = 2*N;
```

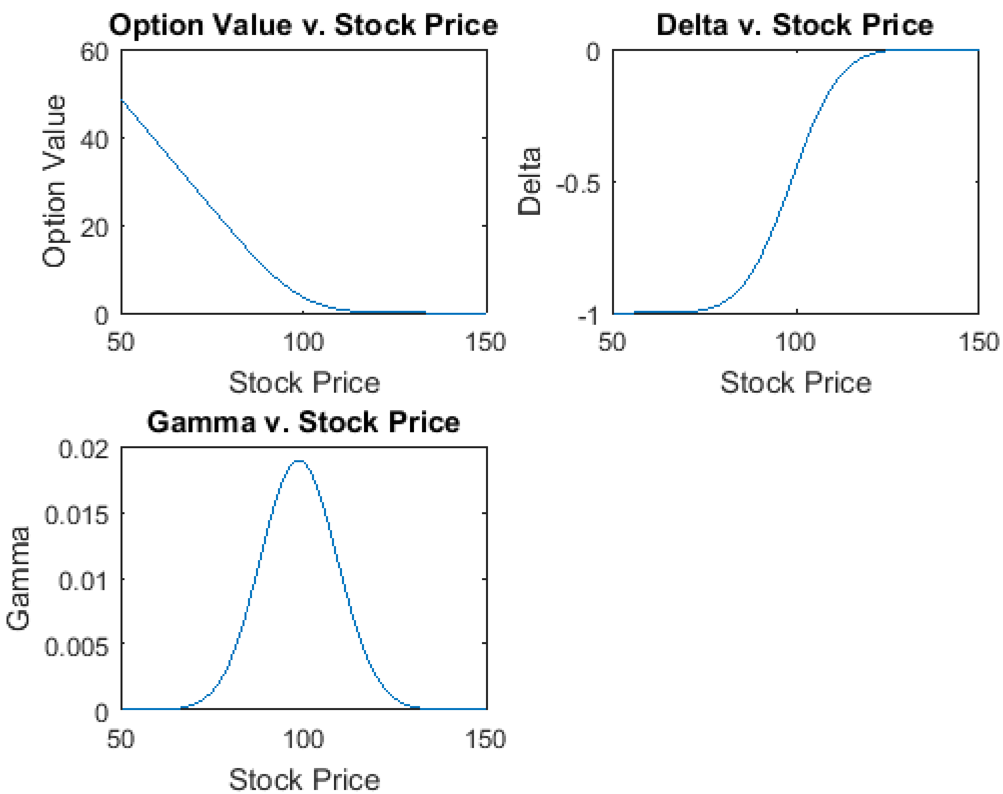
```
    end
    Nlist(j) = N_0 * (2^(j-1));
end

columns = {'Nodes', 'TimeSteps', 'Value', 'Change', 'Ratio'}; % table column names
Vchange = V(:,2:end) - V(:,1:(end-1));  % changes in option price as we refine S
% ratios of option prices as we refine S
Vratio = (V(:,1:(end-2)) - V(:,2:(end-1)))./(V(:,2:(end-1)) - V(:,3:end));

% Output Tables
for k = 1:3
    table(Snodeslist, Nlist, V(k,:)', [0 ; Vchange(k,:)'], [0;0;Vratio(k,:)'], 'VariableNames', columns)
end

% Select only S and corresponding option values in [50,150]
S = S';
newS = S(S <= 150 & S >=50);
newvalues = values(S <= 150 & S >=50);

% Delta and Gamma using forward finite difference
delta = (newvalues(2:end) - newvalues(1:end-1))./(newS(2:end) - newS(1:end-1));
gamma = (delta(2:end) - delta(1:end-1))./(newS(3:end) - newS(1:end-2));

% plots
subplot(2,2,1)
plot(newS, newvalues)
title('Option Value v. Stock Price')
xlabel('Stock Price')
ylabel('Option Value')
subplot(2,2,2)
plot(newS(2:end), delta)
title('Delta v. Stock Price')
xlabel('Stock Price')
ylabel('Delta')
subplot(2,2,3)
plot(newS(3:end), gamma)
title('Gamma v. Stock Price')
xlabel('Stock Price')
ylabel('Gamma')

end
```

```
ans =

    Nodes    TimeSteps    Value      Change      Ratio
    _____    _________    ______    _________    ______

     62         25        3.5851            0         0
    123         50        3.6009     0.015781         0
    245        100        3.6075    0.0065682    2.4026
    489        200        3.6104    0.0029522    2.2248
    977        400        3.6118    0.0013932    2.1191


ans =

    Nodes    TimeSteps    Value       Change       Ratio
    _____    _________    ______    __________    ______
```

62	25	3.6063	0	0
123	50	3.6114	0.0051347	0
245	100	3.6127	0.001292	3.9743
489	200	3.6131	0.0003236	3.9925
977	400	3.6131	8.0956e-05	3.9972

```
ans =
```

Nodes	TimeSteps	Value	Change	Ratio
62	25	3.6047	0	0
123	50	3.611	0.0063581	0
245	100	3.6126	0.0016014	3.9704
489	200	3.613	0.00040149	3.9886
977	400	3.6131	0.00010049	3.9954



Observations

```
% According to the convergence tables, for fully implicit method we see
% the ratios converging to ~2, and for both Crank-Nicolson and Rannacher,
% the ratios are converging to ~4 which is the predicted result from the
% theory.
```

Published with MATLAB® R2016a

Question 4 Subfunction

```
function [V_0, n] = american_finite_diff(S, sigma,r, deltaTau, T, dnorm , payoff, time_step_variable)
% This function takes the following inputs:
% S: the grid of stock prices
% sigma: the volatility function
% r: risk free rate
% deltaTau: initial time step
% T: option expiry
% dnorm: scaling factor of timesteps
% payoff: payoff function of the American option
% time_step_variable: indicates whether variable or constant timestepping
% is to be used
% And calculates the American option values at the given points of S

num_S = length(S);
alpha = zeros(num_S - 2,1);
beta = zeros(num_S - 2,1);

for i=2:(num_S - 1)
    % Central Difference Method
    alpha(i-1) = ((sigma(S(i))*S(i))^2)/((S(i)-S(i-1))*(S(i+1)-S(i-1))) - r*S(i)/(S(i+1)-S(i-1));
    beta(i-1) =  ((sigma(S(i))*S(i))^2)/((S(i+1)-S(i))*(S(i+1)-S(i-1))) + r*S(i)/(S(i+1)-S(i-1));
    if ((alpha(i-1) < 0) || (beta(i-1) < 0))
        % Forward Difference Method
        alpha(i-1) = ((sigma(S(i))*S(i))^2)/((S(i)-S(i-1))*(S(i+1)-S(i-1)));
        beta(i-1) =  ((sigma(S(i))*S(i))^2)/((S(i+1)-S(i))*(S(i+1)-S(i-1))) + r*S(i)/(S(i+1)-S(i));
        if ((alpha(i-1) < 0) || (beta(i-1) < 0))
            % Backward Difference Method
            alpha(i-1) = ((sigma(S(i))*S(i))^2)/((S(i)-S(i-1))*(S(i+1)-S(i-1)))...
                - r*S(i)/(S(i) - S(i-1));
            beta(i-1) = ((sigma(S(i))*S(i))^2)/((S(i+1)-S(i))*(S(i+1)-S(i-1)));
        end
    end
end

% Creating the 3 diagonals needed for M; note that we did not include
% deltaTau in this formulation, since deltaTau may change. Therefore we
% multiply M by deltaTau later (since all values in M contain deltaTau)
lower_diag = [- alpha; 0; 0];
middle_diag = [r;  (alpha + beta + r) ; 0];
upper_diag = [0; 0; -  beta];

% Putting together the sparse diagonal matrix M
M = spdiags([lower_diag middle_diag upper_diag], [-1 0 1], num_S, num_S);

theta = 0;
V = payoff(S)';
t = 0; % counter for cumulative deltaTau
D = 1; % constant for MaxRelChange
timesteps = 0; % counter to keep track of the number of timesteps
Vstar = payoff(S)'; % initial guess
large = 10^6;       % large constant for P matrix in penalty method
tol = 1/large;      % tolerance at which the penalty method quits

while t < T
    timesteps = timesteps + 1;
    if (timesteps == 3)
```

```
        theta = 0.5; % Rannacher; 2 steps of fully implicit then C-N
    end

    % Get LHS and RHS matrices at every time step
    IM = speye(num_S) + (1 - theta)* deltaTau * M;
    RHS_IM = speye(num_S) - theta * deltaTau * M;

    Vold_t = V;
    V = Vstar;
    while (1) % the bulk of the penalty method
        Vold_k = V;
        P = large*spdiags(Vold_k < Vstar, 0, num_S, num_S);
        [L, U] = lu(IM + P);
        V = U\(L\(RHS_IM * Vold_t + P * Vstar));
        if (max(abs(V - Vold_k)./max(1, abs(V))) < tol)
            break % leaves the while loop when the value of V converges
        end

    end

    t = t + deltaTau; % keeps track of total cumulative deltaTau's

    % if Variable time stepping is being used, this changes the value of
    % deltaTau. Otherwise it remains the same.
    if strcmp('variable', time_step_variable)
        MaxRelChange = max(abs(V - Vold_t)./max(max(D, abs(V)), abs(Vold_t)));
        deltaTau = min((dnorm / MaxRelChange) * deltaTau, T-t);
    end
end

% returns option values and timesteps used (for the table)
V_0 = V;
n = timesteps;

end
```

Published with MATLAB® R2016a

Contents

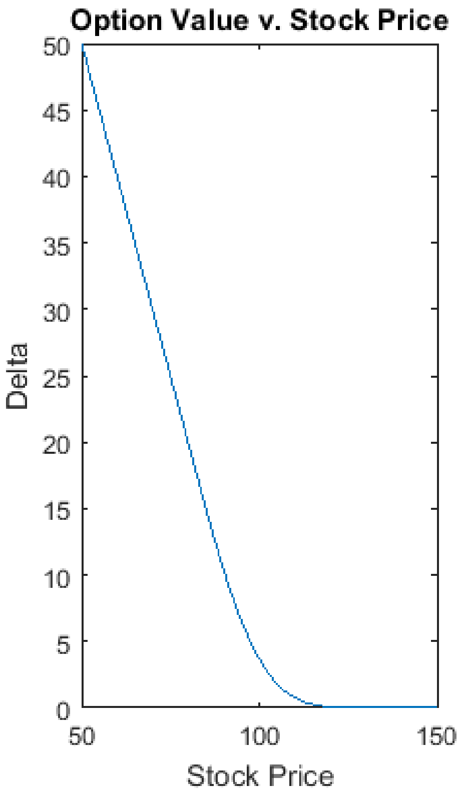
Question 4 Part 1

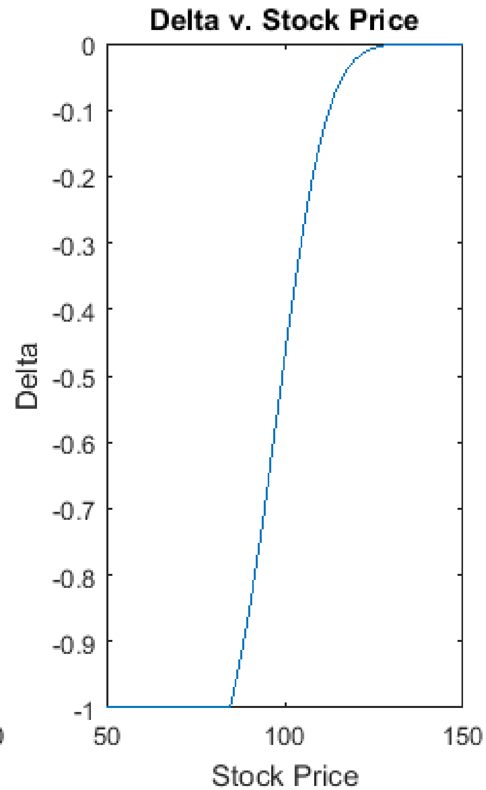
```
function ans = american_finite_diff_table_graph(time_step_variable)
% This function creates the table and graphs for Question 4
% if time_step_variable = 'variable' then this corresponds to the variable
% time-stepping option.

% parameters and constants
alpha = 15;
sigma = @(x)(alpha/x); % volatility function
r = 0.025;
T = 0.5;
K = 100;
S_0 = 100;
payoff = @(x)(max(K - x, 0));
N_0 = 25;

% initial grid of stock prices
S = [0:0.1*K:0.4*K,...
0.45*K:0.05*K:0.8*K,...
0.82*K:0.02*K:0.9*K,...
0.91*K:0.01*K:1.1*K,...
1.12*K:0.02*K:1.2*K,...
1.25*K:.05*K:1.6*K,...
1.7*K:0.1*K:2*K,...
2.2*K, 2.4*K, 2.8*K,...
3.6*K, 5*K, 7.5*K, 10*K];

numsteps = 6; % number of times to refine S

dnorm = 0.1;
deltaTau = T/25;

Vall = zeros(numsteps,1);
Nlist = zeros(numsteps,1) + N_0;
Snodeslist = zeros(numsteps,1);
timesteps = zeros(numsteps,1);

for i = 1:numsteps
    % This loop calculates the american option price, then refines S
    % and finds the option price again.
    N = N_0;
   [V_array, timesteps(i)] = american_finite_diff(S, sigma,r, deltaTau, T, dnorm , payoff, time_step_variable);
   Vall(i) = V_array(S == S_0);

   % reduces deltaTau and dnorm at every refinement step
   deltaTau = deltaTau/4;
   dnorm = dnorm/2;

   Snodeslist(i) = length(S);
   % refines S
   if i < numsteps
       insert = (S(2:end) + S(1:end-1))/2 ;
       new_S = zeros(length(S) + length(insert),1);
       new_S(1:2:end) = S;
```

```
        new_S(2:2:end-1) = insert;
        S = new_S';
        N = 2*N;
    end

end

columns = {'Nodes', 'TimeSteps', 'Value', 'Change', 'Ratio'}; % column names
Vchange = Vall(2:end) - Vall(1:(end-1)); % change in option values
Vratio = (Vall(1:(end-2)) - Vall(2:(end-1)))./(Vall(2:(end-1)) - Vall(3:end));

% table
table(Snodeslist, timesteps, Vall, [0 ; Vchange], [0;0;Vratio], 'VariableNames', columns)

% Restricts S and corresponding option values to S in [50,150]
S = S';
S_new = S(S <= 150 & S >= 50);
V_array = V_array(S <= 150 & S >= 50);
delta = (V_array(2:end) - V_array(1:end-1))./(S_new(2:end) - S_new(1:end-1));

% plots
subplot(1,2,1)
plot(S_new, V_array)
title('Option Value v. Stock Price')
xlabel('Stock Price')
ylabel('Delta')
subplot(1,2,2);
plot(S_new(2:end), delta)
title('Delta v. Stock Price')
xlabel('Stock Price')
ylabel('Delta')

end
```

ans =

Nodes	TimeSteps	Value	Change	Ratio
62	25	3.6987	0	0
123	100	3.7089	0.010176	0
245	401	3.7151	0.0061701	1.6492
489	1601	3.7125	-0.0025375	-2.4315
977	6401	3.7119	-0.0006434	3.9439
1953	25601	3.7117	-0.00016247	3.9601





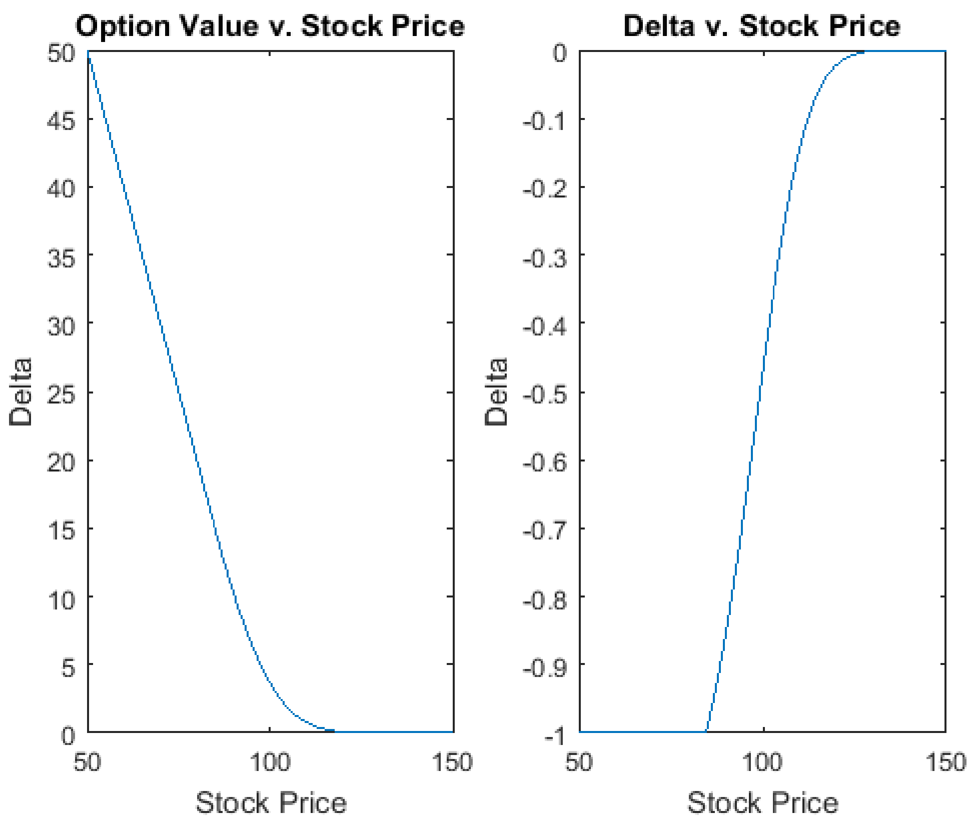
Oberservations

```
% As opposed to the graphs generated in Question 3, we can see that there
% is a slight 'pinch', or discontinuity where Delta hits -1, due to the
% fact that we are now dealing with American options rather than European.

% Furthermore, there seems to be little difference between Constant and
% Variable time stepping, other than the fact that Variable time stepping
% converged to a more accurate result much more quickly than Constant

% Looking at the tables we can see that Constant timestepping went up to
% 25,601 steps whereas Variable only went up to 1,065 but had better
% convergence (lower "Change" column)

% With that said, they still both converged to 4 as the theory predicts.
% Constant time stepping converged from below and Variable time stepping
% from above; it is not clear why this is the case.
```

Published with MATLAB® R2016a

Question 4 Part 1

```
ans =

    Nodes    TimeSteps    Value       Change       Ratio 
    _____    _________    ______    __________    ______

      62        29        3.7008             0         0
     123        63        3.7093     0.0084356         0
     245       131        3.7111     0.0018555    4.5463
     489       264        3.7116    0.00042984    4.3168
     977       531        3.7117    0.00010211    4.2095
    1953      1065        3.7117    2.4613e-05    4.1486
```

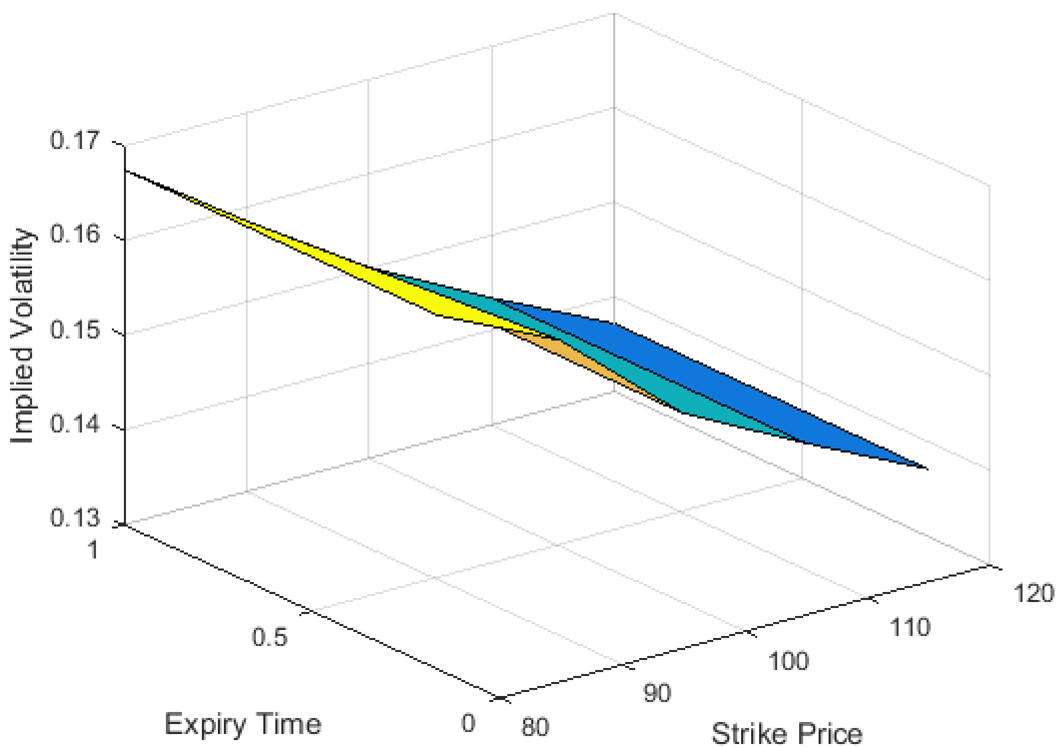


Question 5(a) Implied Volatility Surface

```
function [vol_plot, values] = implied_vol_surf(ploton)
% plots the implied volatility surface, if ploton = 't', and also
% returns the option values

% constants
alpha = 15;
r = 0.025;
sigma = @(x)(alpha/x);
N_0 = 25;
K = [80, 90, 100, 110, 120];
T = [1/6, 1];
option_values = zeros(length(K), length(T));
stepmethod = 'Rannacher';


for t = 1:length(T);
    for i = 1:length(K)
        N = N_0;
        % create a grid of S centered around K(i)
        S = [0:0.1*K(i):0.4*K(i),...
        0.45*K(i):0.05*K(i):0.8*K(i),...
        0.82*K(i):0.02*K(i):0.9*K(i),...
        0.91*K(i):0.01*K(i):1.1*K(i),...
        1.12*K(i):0.02*K(i):1.2*K(i),...
        1.25*K(i):.05*K(i):1.6*K(i),...
        1.7*K(i):0.1*K(i):2*K(i),...
        2.2*K(i), 2.4*K(i), 2.8*K(i),...
        3.6*K(i), 5*K(i), 7.5*K(i), 10*K(i)];

        Srefinements = 4; % number of times to refine S

        % refining S
        for j = 1:Srefinements
           insert = (S(2:end) + S(1:end-1))/2 ;
           new_S = zeros(length(S) + length(insert),1);
           new_S(1:2:end) = S;
           new_S(2:2:end-1) = insert;
           S = new_S';
           N = 2*N;
        end

        payoff = @(x)(max(x - K(i),0)); % find payoff
        % find option values using fin_diff
        V_array = fin_diff(S, sigma,r, T(t), N, stepmethod, payoff);
        % restrict to value corresponding to S closest to 100
        option_values(i,t) = V_array(min(abs(100 - S)) == abs(100 - S));
        % find implied vol using builtin function
        implied_vol(i,t) = blsimpv(100, K(i), r, T(t), option_values(i,t));
    end
end

% just return 0 if ploton is 'f'
vol_plot = 0;
% return the plot if requested
if strcmp(ploton, 't')
    vol_plot = surf(K,T,implied_vol');
```

```
    xlabel('Strike Price')
    ylabel('Expiry Time')
    zlabel('Implied Volatility')
end

values = option_values; % return option values

end
```

```
ans =

  Surface with properties:

       EdgeColor: [0 0 0]
       LineStyle: '-'
       FaceColor: 'flat'
    FaceLighting: 'flat'
       FaceAlpha: 1
           XData: [80 90 100 110 120]
           YData: [0.1667 1]
           ZData: [2x5 double]
           CData: [2x5 double]

  Use GET to show all properties
```



Published with MATLAB® R2016a

Question 5 Subfunction

```
function ans = BlkSholesValue(S_0, T, sigma, r, K, callput)
% Returns the black sholes option value for
% S_0 = stock price
% T = expiry
% sigma = volatility function
% r = risk free rate
% K = expiry
% callput = call or put option

% We create the grid of S prices (assuming S_0 < 10*k)
S = [0:0.1*K:0.4*K,...
0.45*K:0.05*K:0.8*K,...
0.82*K:0.02*K:0.9*K,...
0.91*K:0.01*K:1.1*K,...
1.12*K:0.02*K:1.2*K,...
1.25*K:.05*K:1.6*K,...
1.7*K:0.1*K:2*K,...
2.2*K, 2.4*K, 2.8*K,...
3.6*K, 5*K, 7.5*K, 10*K];

N = 25;
Srefinements = 4; % number of times to refine S
for j = 1:Srefinements
   % refines S
   insert = (S(2:end) + S(1:end-1))/2 ;
   new_S = zeros(length(S) + length(insert),1);
   new_S(1:2:end) = S;
   new_S(2:2:end-1) = insert;
   S = new_S';
   N = 2*N;
end

% sets payoff based on 'call' or 'put'
if strcmp(callput, 'call')
    payoff = @(S)(max((S-K),0));
else
    payoff = @(S)(max((K-S),0));
end

% finds the array of option values using fin_diff
Varray = fin_diff(S, sigma, r, T, N, 'Rannacher', payoff);
% take the option value representing S closest to 100
V = Varray(min(abs(S - S_0)) == abs(S - S_0));
ans = V(1);

end
```

Published with MATLAB® R2016a

Question 5 Subfunction

```
function [resid, values] = residual_vector(x, nu, mktV, K, T, callput, S_0, r)
% Returns the residual vector and the model's option values
% x = vector of constants in the model
% nu = constant of the model
% mktV = market values
% K = vector of strike prices
% T = vector of expiry times
% callput = call or put option
% S_0 = price at which we interested in the option value
% r = risk free rate

% RBF Kernel volatility function
sigma = @(S) abs(x(1) + exp(- ((S - K).^2)/(2 * nu^2)) * x(2:end));

option_values = zeros(length(K),length(T));
% Find option values with BlkSholesValue function and RBF Kernel volatility
for i = 1:length(K)
    for j = 1:length(T)
        option_values(i,j) = BlkSholesValue(S_0, T(j), sigma, r, K(i), callput);
    end
end

val = option_values - mktV; % residuals
resid = reshape(val, numel(val), 1); % reshape val matrix into long column vector
values = option_values;

end
```

Published with MATLAB® R2016a

Question 5 Subfunction

```
function [F, J] = optimizer (x, nu, mktV, K, T, callput, S_0, r)
% Optimizer is a function used by the lsqnonlin (non linear least squares
% minimizer) function to determine the values of x which minimize the least
% squares of the residuals between the model's option prices and market's
% option prices
% x = vector of constants in the model
% nu = constant of the model
% mktV = market values
% K = vector of strike prices
% T = vector of expiry times
% callput = call or put option
% S_0 = price at which we interested in the option value
% r = risk free rate

F = residual_vector(x, nu, mktV, K, T, callput, S_0, r); % residuals
jacob = zeros(length(F),length(x));                        % initialize Jacobian

if nargout > 1
    delta = sqrt(eps); % sqrt(machine epsilon)

        for i = 1:length(x)
            I = speye(length(x)); % identity matrix
            e_i = I(:, i);        % create elementary column vectors
            % find Jacobian using forward finite difference method
            Fright = residual_vector(x + delta*e_i, nu, mktV, K, T, callput, S_0, r);
            jacob(:,i) =  (Fright - F)/delta;
        end
    J = jacob;
end
end
```

Published with MATLAB® R2016a

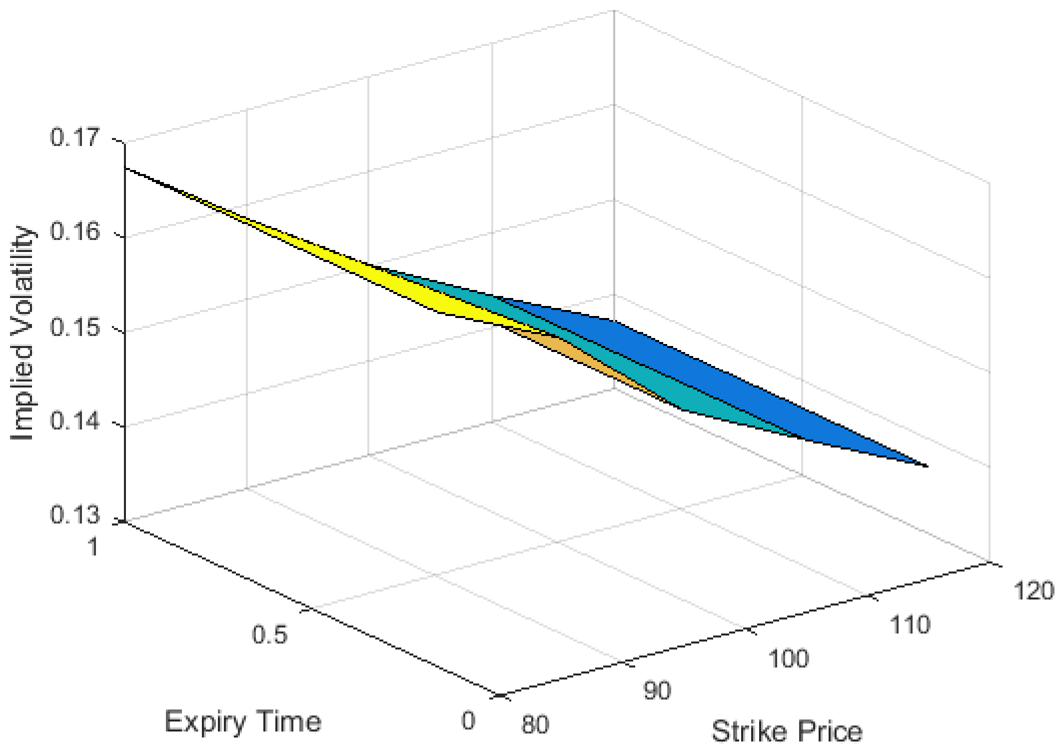
Contents

Question 5(b,c,d) Local Volatility Comparisons

This code creates the implied volatility surface for the RBF Kernel volatility model, and creates a plot comparing the RBF model volatility to the original alpha/S volatility in the Black Scholes model

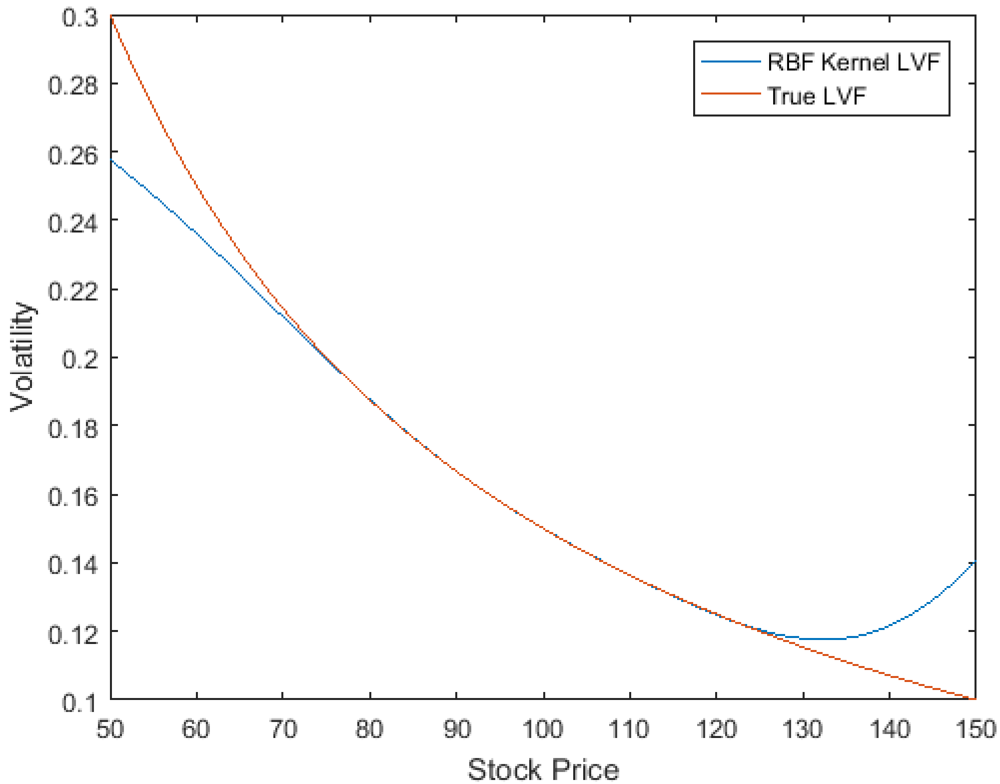
```
[noplot, values] = implied_vol_surf('f'); % values = market option values
K = [80, 90, 100, 110, 120];
T = [1/6, 1];
r = 0.025;
alpha = 15;
S_0 = 100;
nu = 30;

% original guess for minimizing x values
x_0 = zeros(length(K) + 1,1);
x_0(1) = alpha/S_0;

options = optimset('Jacobian', 'on', 'Algorithm', 'levenberg-marquardt', ...
    'Display', 'iter', 'MaxIter', 50); % optimization options
% finds the x value xstar which minimizes the sum of squares
[xstar, calib] = lsqnonlin(@(x) optimizer(x, 30, values, K, T, 'call', S_0, r), x_0);

S = 50:150; % S from 50 to 150
% define the RBF kernel volatility using xstar:
sigmastar = @(s)(abs(xstar(1) + (exp(-((s - K).^2)/(2 * nu^2)) * xstar(2:end))));
LVF_array = arrayfun(sigmastar,S); % apply sigma star to S in [50,150]

estvalues = zeros(length(K), length(T));
% find estimated option values (estvalues) based on sigmastar volatility
[noresid, estvalues] = residual_vector(xstar, nu, values, K, T, 'call', S_0, r);
for i = 1:length(K)
    for t = 1:length(T)
        % calculate implied volatility
        implied_vol_est(i,t) = blsimpv(100, K(i), r, T(t), estvalues(i,t));
    end
end

surf(K,T,implied_vol_est') % implied volatility surface
ylabel('Expiry Time')
xlabel('Strike Price')
zlabel('Implied Volatility')
```

```
Local minimum possible.

lsqnonlin stopped because the final change in the sum of squares relative to
its initial value is less than the default value of the function tolerance.
```




```
X_Star = xstar % outputs xstar
Calibration_Error = calib % outputs calibration error
% plots estimated (RBF Kernel) implied vol vs. True implied vol
plot(S, LVF_array, S, 15./S)
xlabel('Stock Price');
ylabel('Volatility');
legend('RBF Kernel LVF', 'True LVF');
```

```
X_Star =

    0.3181
   -0.8167
    3.0590
   -5.3586
    4.7034
   -1.8713


Calibration_Error =

   4.6982e-08
```



Observations

```
% The implied volatility surface from this RBF Kernel volatility model
% appears to be identical to the previous one. This should not be
% surprising since xstar was chosen so that they would be as close as
% possible; and from the graph of RBF v. True volatility we can see that
% around S_0 = 100 they are very close.

% However, the volatilities when mapped against stock prices are not the
% same. The RBF Kernel vol seems to perform very well for prices near S_0 =
% 100 but once we get beyond about [75,125] it performs poorly. At low
% prices it undershoots the volatility and at high prices it overshoots it.

% This may have something to do with the volatility smile seen in the
% previous assignment?
```

Published with MATLAB® R2016a