**Machine Learning for Signal Processing and Pattern Classification.**

**SIMRAN**

1. The given data is of the form  for . The goal is to find the polynomial that approximates the data by minimizing the energy of the residual:



where  is a polynomial, eg: 

The problem can be viewed as solving the over determined system of equations,

, which we denote as.

**Least square solution.**

1. Implement the same for the given data (data\_1.txt).

**CODE:**

clc;

clear all;

close all;

load data\_1.txt; %loading the data

y = data\_1(:,2); % y values

t = data\_1(:,1); % t values

N = length(y); %taking the length of y

figure(1)

plot(y) %ploting y data

title('Data')

e = ones(N, 1);

A = [e t t.^2];

%A matrix

F = A' \* A;

x = F \ A' \* y; %least square solution

figure(2)

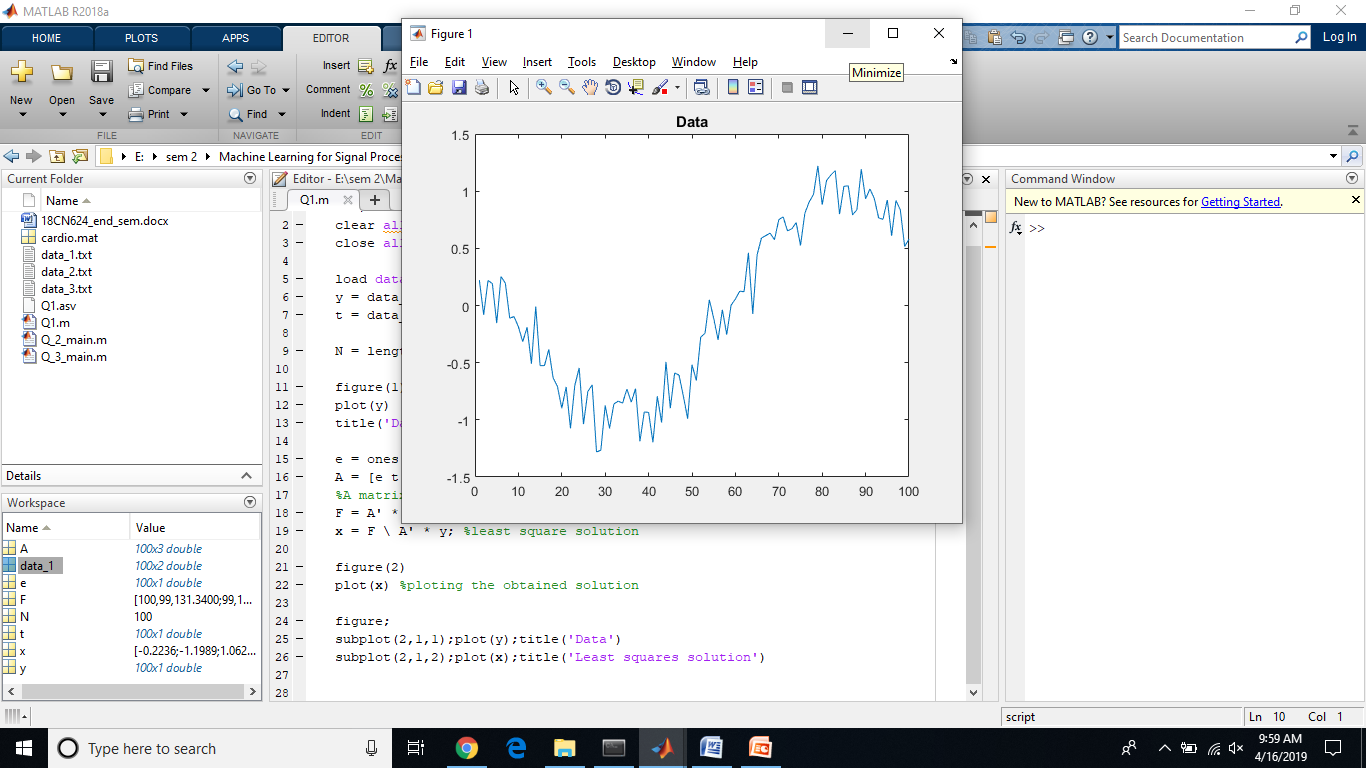
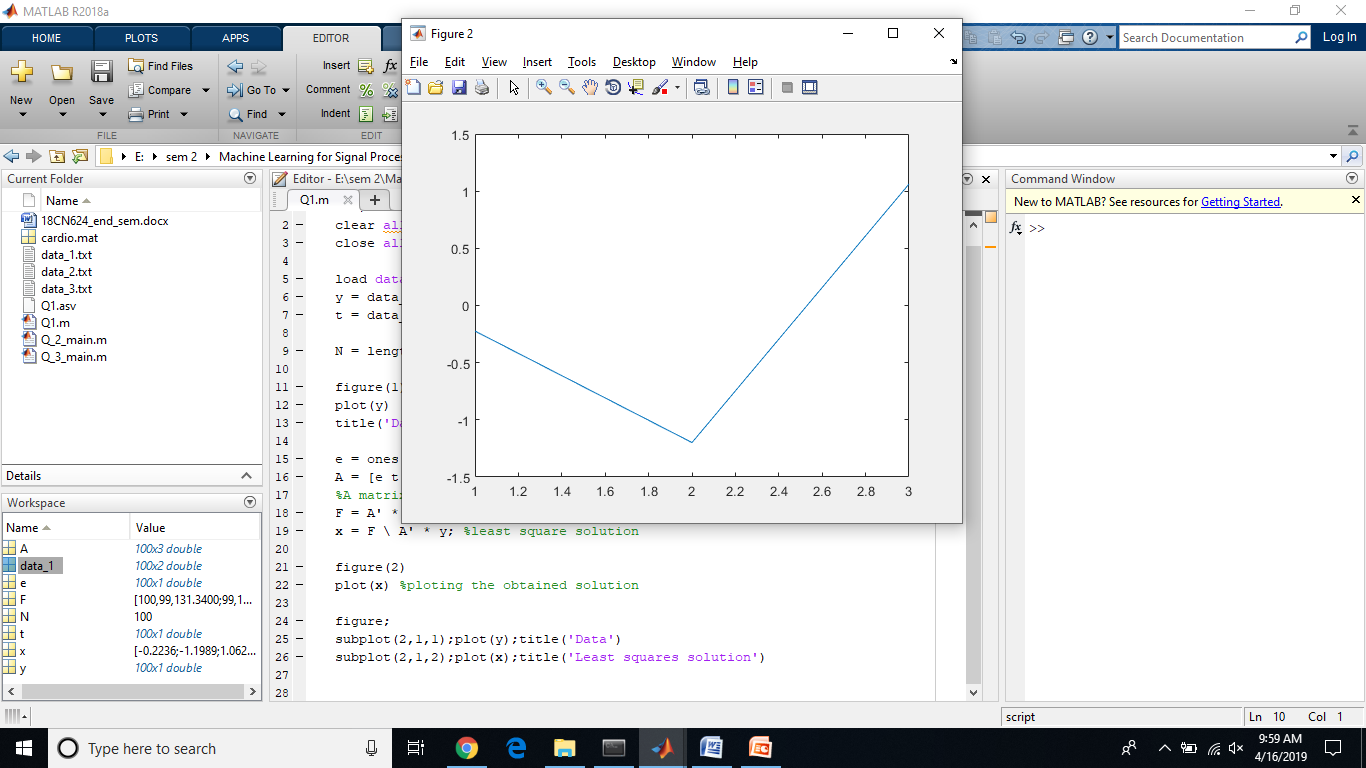
plot(x) %ploting the obtained solution

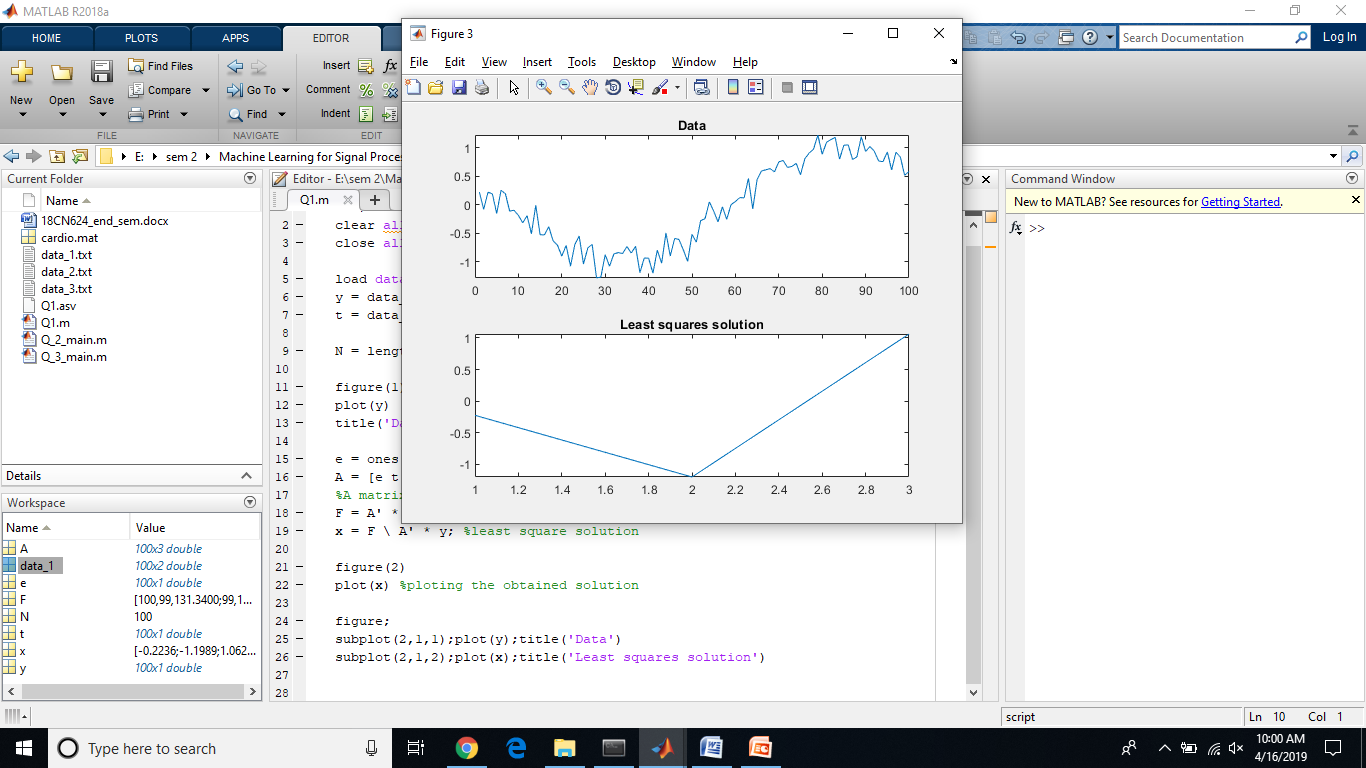
figure;

subplot(2,1,1);plot(y);title('Data')

subplot(2,1,2);plot(x);title('Least squares solution')

**OUTPUT:**



1. One approach to predict future values of a time series is based on linear prediction, eg



If past data is available, then the problem of finding can be solved using least squares. Finding  can be viewed as one of solving an over determined system of equations. For example, if is available for , and we seek a third order linear predictor then the overdetermined system of equations are given by,

, which we can write as where **A** is a matrix of size  . The least squares solution is given by. Note that is small of size only. Hence,  is obtained by solving a small linear system of equations.

**Implement the fourth order linear predictor for the given data.**

**CODE:**

clc;

clear all;

close all

%% Load data

load data\_2.txt;

y = data\_2; % data value

%% Display data

figure(1)

plot(y)

title('Data')

L = 100;

%% 4th order linear predictor

N = length(y);

H = [y(4:N-1) y(3:N-2) y(2:N-3) y(1:N-4)];

b = y(5:N);

a = (H' \* H) \ (H' \* b) % a : coefficients of linear predictor

g1 = [y; zeros(L, 1)];

for i = N+1:N+L

g1(i) = a(1) \* g1(i-1) + a(2) \* g1(i-2) + a(3) \* g1(i-3) +a(4) \*g1(i-4);

end

figure

subplot(2,1,1);plot(y);title('Data')

subplot(2,1,2);plot(g1);line([N N], [-2 2], 'linestyle', '--');title('Data and predicted values (deg(4))')

**OUTPUT:**

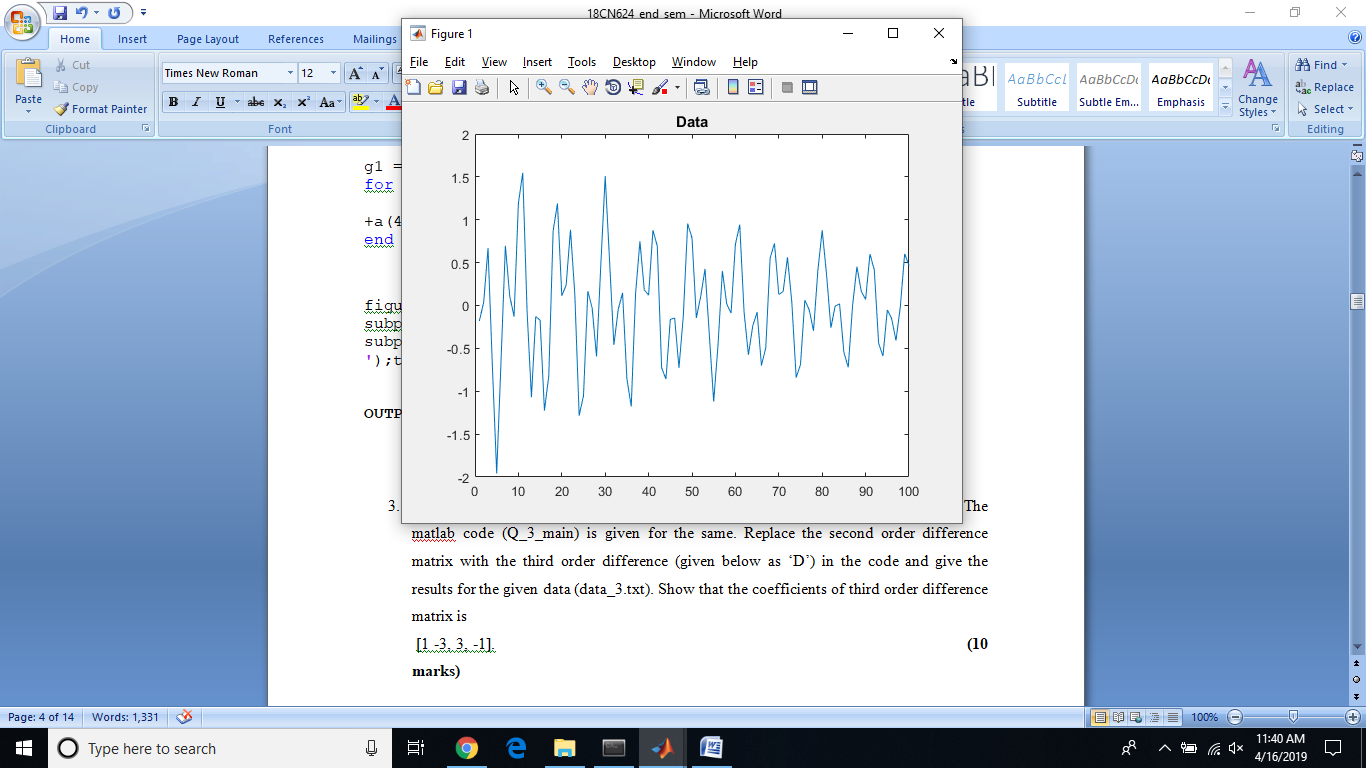
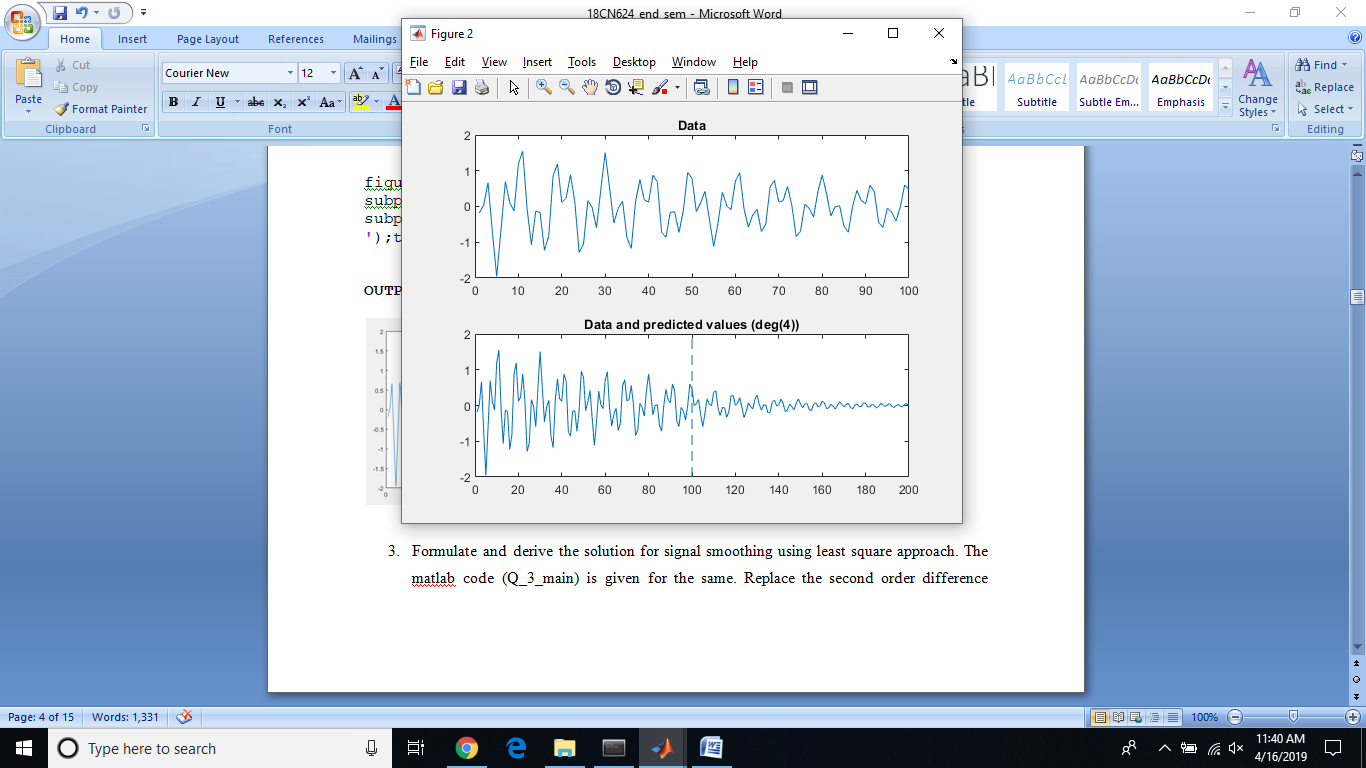
**a =**

**1.42**

**-1.67**

**1.36**

**-0.90**

1. Derive the solution for signal smoothing using least square approach using the third order derivative 

**CODE:**

%% Least squares smoothing

clc;

clear all;

close all;

%% Load data

load data\_3.txt;

y = data\_3; % data value

N = length(y);

%% Display data

figure(1)

clf

plot(y)

title('Data')

%% Smoothing (degree = 3)

% D is the third-order difference matrix.

% It approximates the third-order derivative.

% In order to exploit fast banded solvers in Matlab,

% we define D as a sparse matrix using 'spdiags'.

e = ones(N, 1);

D = spdiags([e -3\*e 3\*e -e], 0:3, N-3, N); %third order derivative coefficient matrix

%%

% Observe the first and last corners of D.

% (D is too big to display, so we show

% the first and last corners only.)

%%

% First corner of D:

full(D(1:5, 1:5))

%%

% Last corner of D:

full(D(end-4:end, end-4:end))

%%

% Solve the least square problem

lam = 0.5;

F = lam\*(speye(N)) +(D' \* D);% F is a banded matrix

%

x = F \ (lam\*y); % Matlab uses a fast solver for banded systems)

% F = (speye(N)) +lam\*(D' \* D);% F is a banded matrix

%

% x = F \ y;

figure;

subplot(2,1,1);plot(y);title('Data')

subplot(2,1,2);plot(x);title('Least squares smoothing')

**OUTPUT:**

**ans =**

**1 -3 3 -1 0**

**0 1 -3 3 -1**

**0 0 1 -3 3**

**0 0 0 1 -3**

**0 0 0 0 1**

**ans =**

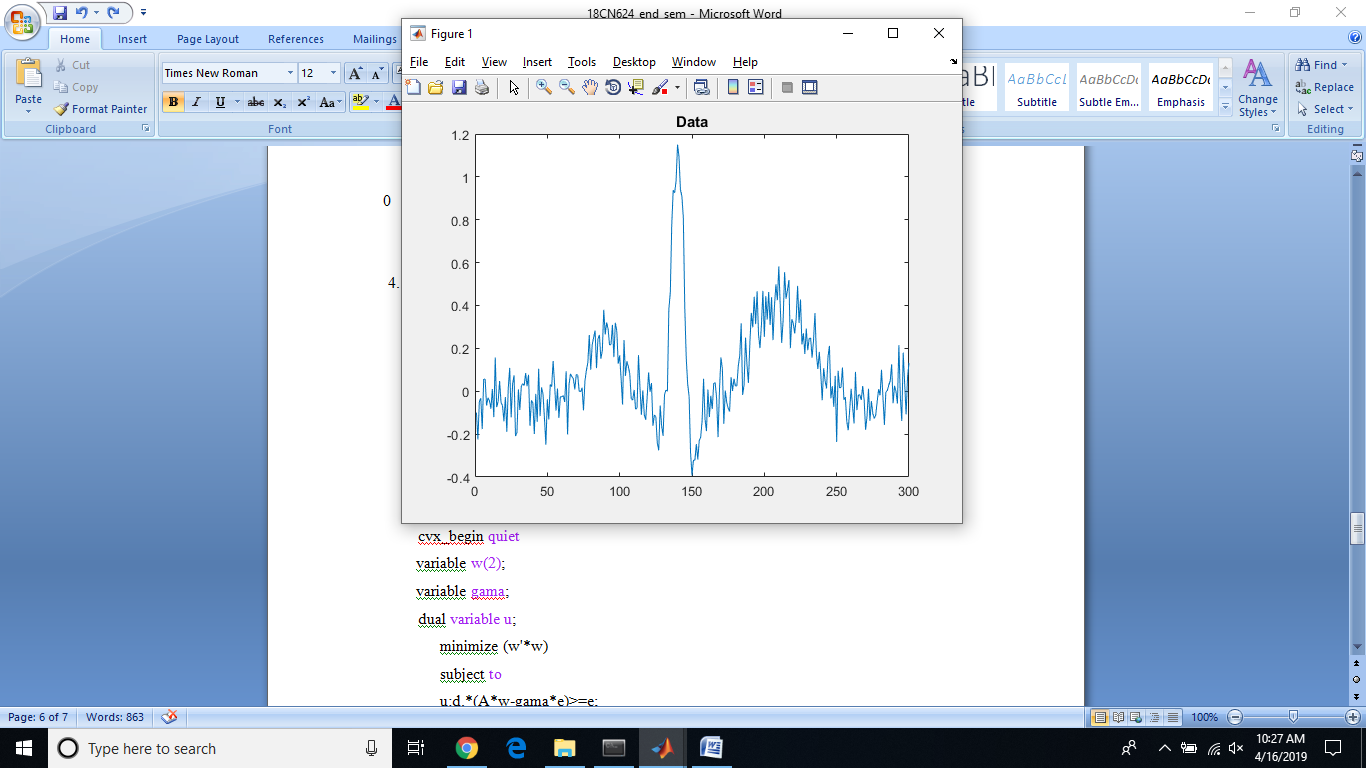
**-1 0 0 0 0**

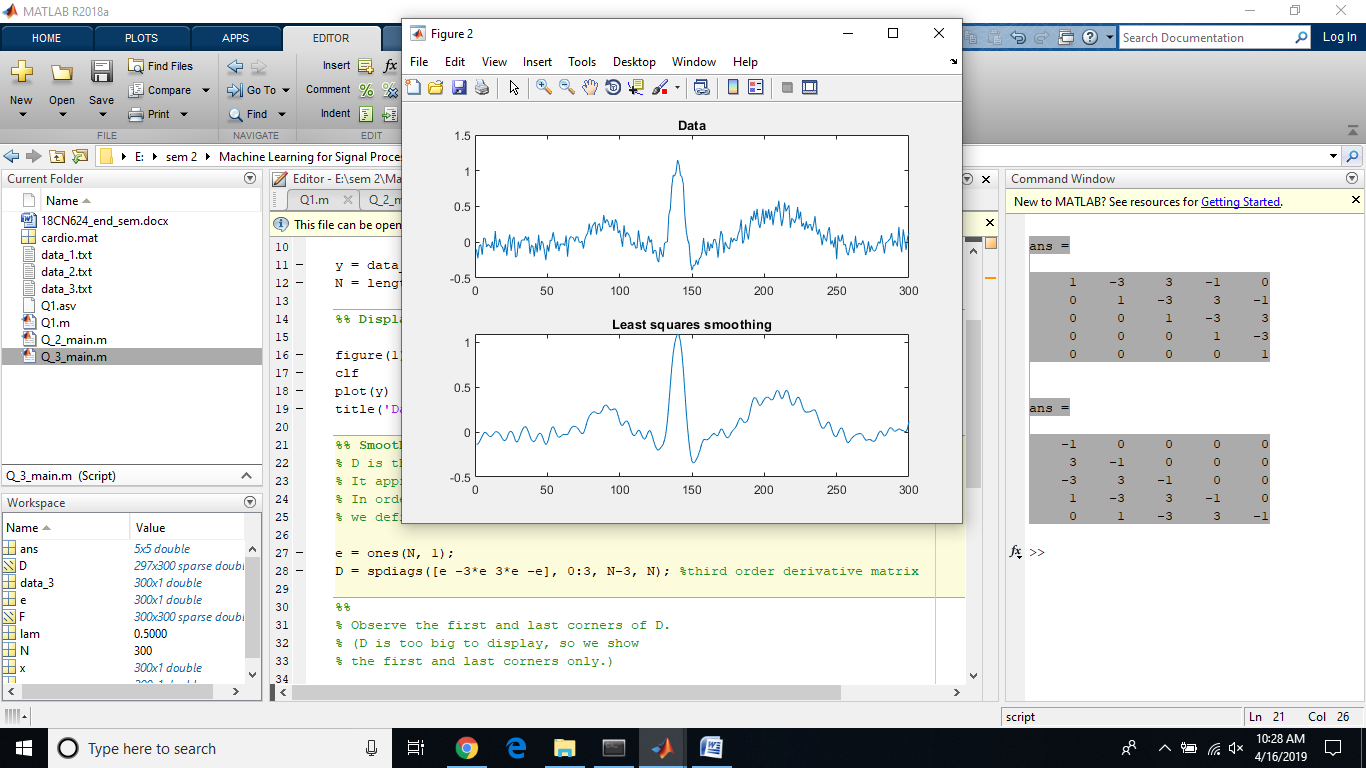
**3 -1 0 0 0**

**-3 3 -1 0 0**

**1 -3 3 -1 0**

**0 1 -3 3 -1**

****

****

1. Consider the data given below:

|  |  |  |  |
| --- | --- | --- | --- |
| Point id | X­1 | X2 | d |
| 1 | 1 | 1 | -1 |
| 2 | 0.5 | 0.5 | -1 |
| 3 | 3 | 3 | 1 |
| 4 | 4 | 5 | 1 |

Implement SVM hard using cvx.

**CODE:**

clear all

clc

A=[1 0.5 3 4;1 0.5 3 5]';

d=[-1 -1 1 1]';

e=ones(size(A,1),1);

cvx\_begin quiet

variable w(2);

variable gama;

dual variable u;

minimize (w'\*w)

subject to

u:d.\*(A\*w-gama\*e)>=e;

cvx\_end

format bank

disp('w vector')

w

disp('gama')

gama

disp('u vector (lagange multipliers)')

u

plot(A(1:2,1), A(1:2,2),'\*');% plot-1 points

hold on

plot(A(3:4,1), A(3:4,2),'o');% plot+1 points

hold on

x1=-1:4;

x2=-(u(1)/u(3))\*x1+(gama/u(3));

plot(x1,x2); % draw the classifier line

hold on

x2=-(u(1)/u(3))\*x1+((-1+gama)/u(3));

plot(x1,x2) % draw the lower bounding line

hold on

x2=-(u(1)/u(3))\*x1+((1+gama)/u(3));

plot(x1,x2) % draw the upper bounding line

**OUTPUT:**

w vector

**w =**

**0.50**

**0.50**

gama

gama =

2.00

**u vector (lagange multipliers)**

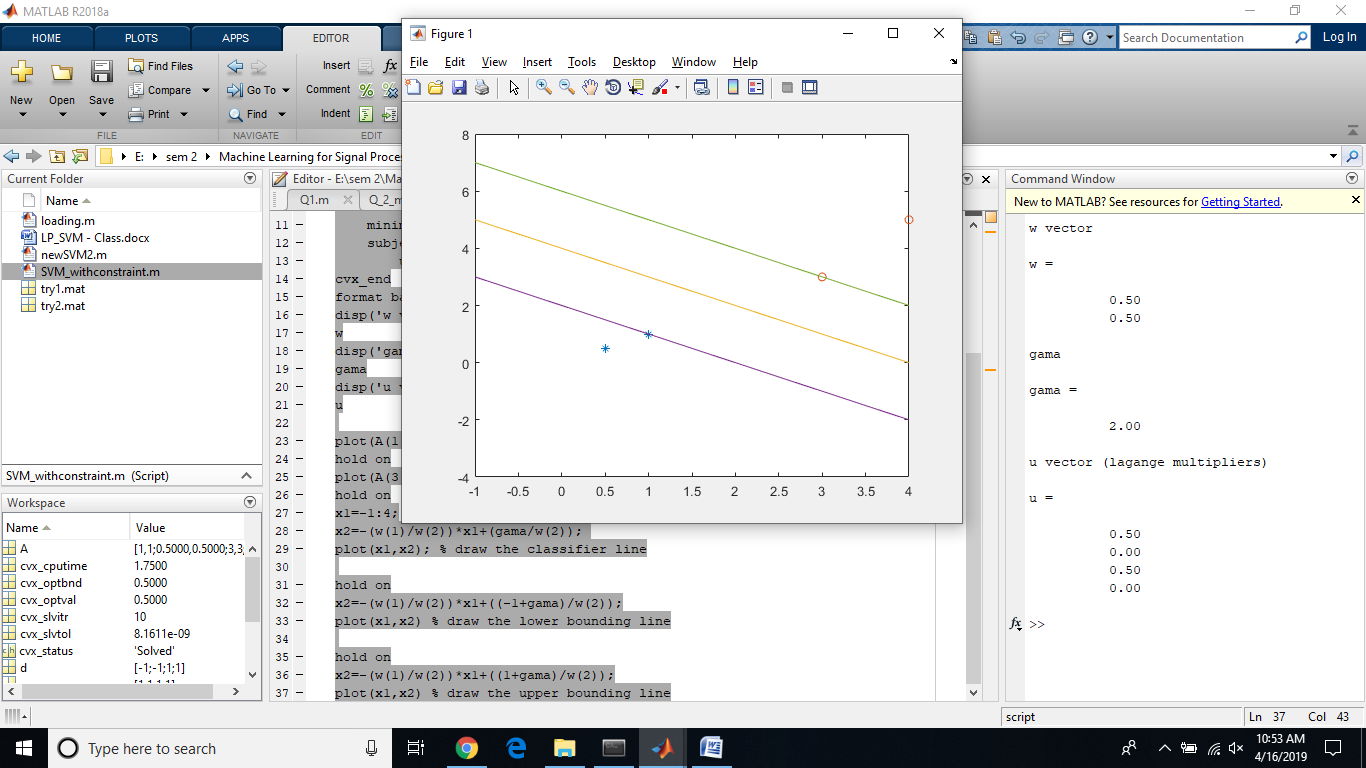
**u =**

**0.50**

**0.00**

**0.50**

**0.00**



**Support vectors are: (X1,X2) = (1,1) and (3,3).**

1. Use matlab/python programming to prove that “Convolution in time domain is equal to the multiplication in frequency domain”.

**CODE:**

clear all

clc

x=[1 2 3 4] % 1st signal

y=[5 6 7] % 2nd signal

timedomain\_conv=conv(x,y) %convolving the two signals in time domain.

m=length(x); % m:length of the 1st signal

n=length(y); % n:length of the 2nd signal

l=m+n-1; % l:length of the convolved sequence

w=fft(x,l); % fft of the 1st signal

z=fft(y,l); % fft of the 2nd signal

a=w.\*z; % multipying the coefficients of the signals(frequency domain).

frequencydomain\_mul=ifft(a) % converting it back to time domain using ifft.

**OUTPUT:**

**x =**

**1.00 2.00 3.00 4.00**

**y =**

**5.00 6.00 7.00**

**timedomain\_conv =**

**5.00 16.00 34.00 52.00 45.00 28.00**

**frequencydomain\_mul =**

**5.00 16.00 34.00 52.00 45.00 28.00**

1. Use the matlab/python programming to compute the DFT matrix for length ‘N’, which must be the power of 2. (Hint: WNnk,, where WN = e-j\*2\*pi/N, n = 0 to N-1 and k=0 to N-1). Avoid using loop

**CODE:**

clear all

clc

x=[2 4 8 16 32];

if(log2(x))

N=length(x); % N is the length of signal

n=0:N-1; % n is the number of columns

theta=2\*pi\*n/N; % theta value are taken

k=(0:N-1)'; % k is the number of rows

W=exp(-i\*k\*theta)\*x' % coefficients of DFT

w=dftmtx(N)\*x' % built-in command for dft coefficients

X=fft(x)' % built-in command for fft

else

disp('Not a valid signal')

end

**OUTPUT:**

W =

62.00

-6.29

-19.71

-19.71

-6.29

w =

62.00

-6.29

-19.71

-19.71

-6.29

X =

62.00

-6.29

-19.71

-19.71

-6.29

**Taking x=[1 2 3 4 5 6 7 8]**

**OUTPUT:**

x =

Columns 1 through 6

1.00 2.00 3.00 4.00 5.00 6.00

Columns 7 through 8

7.00 8.00

**Not a valid signal**

1. Use the below given formulation to classify the given data



The classifier is: 

**CODE:**

clear all

clc

load('cardio.mat')

d=y;

a=size(X)

e=ones(size(X,1),1);

cvx\_begin quiet

variable w(21);

variable gama;

variable psii(1831);

minimize sum(psii)

subject to

d.\*(X\*w-gama\*e)>=1-psii;

psii>=0;

cvx\_end

labels=sign(X\*w-gama); % predict the class label using the given calssifier

1. Apply SVM hard margin formulation to classify the given data (‘cardio.mat’)

**CODE:**

clear all

clc

load('cardio.mat')

d=y;

d(d==0) = -1; % converting the labels of 0 to -1

e=ones(size(X,1),1);

A = (d\*d').\*(X\*X');

cvx\_begin quiet

variables u(1831)

minimize (((1/2)\*u'\*A\*u) - (e'\*u));

subject to

u'\*y == 0;

u>=0;

cvx\_end

w=sum(u\*d'\*X);% calculating w

gamma=(w\*X(99,:)') + 1; % calculating gamma

predict\_labels=sign(w\*X'-gamma); % predicting the labels

**OUTPUT:**

w =

Columns 1 through 7

-9.91 -288.39 118.73 -107.81 183.90 176.96 640.33

Columns 8 through 14

420.16 53.40 352.52 -277.11 41.67 -80.54 -37.73

Columns 15 through 21

30.65 5.19 -452.36 -449.45 -414.66 353.38 -216.11

gamma =

-2911.55

**predict\_labels will predicts the labels of the classes using sign classifier.(1831)**

1. Develop a simple 1D-CNN architecture to classify the given data (‘cardio.mat’)

**Code and output in Q9.ipython file.**