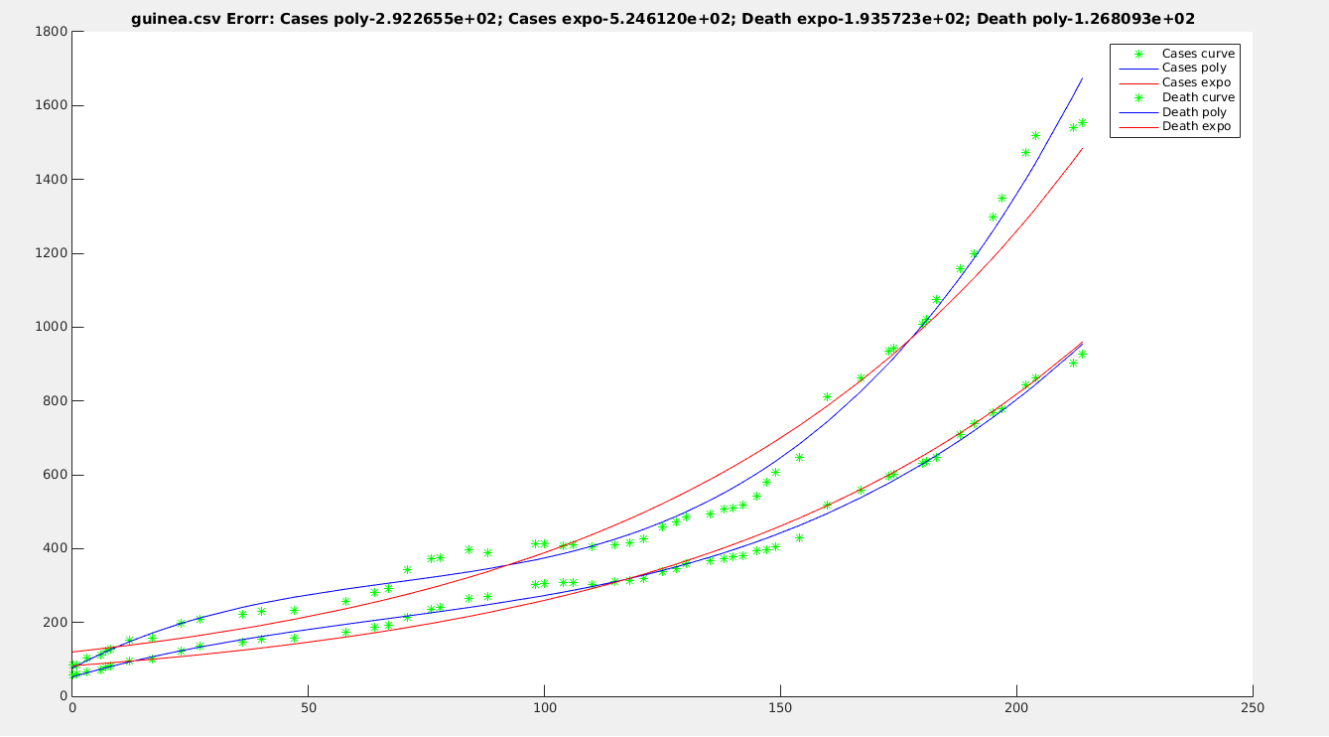
NAME : SRAVANI KAMISETTY
SID : 304414410
MATHEMATICAL MODELLING – CS170 – ASSIGNMENT 2

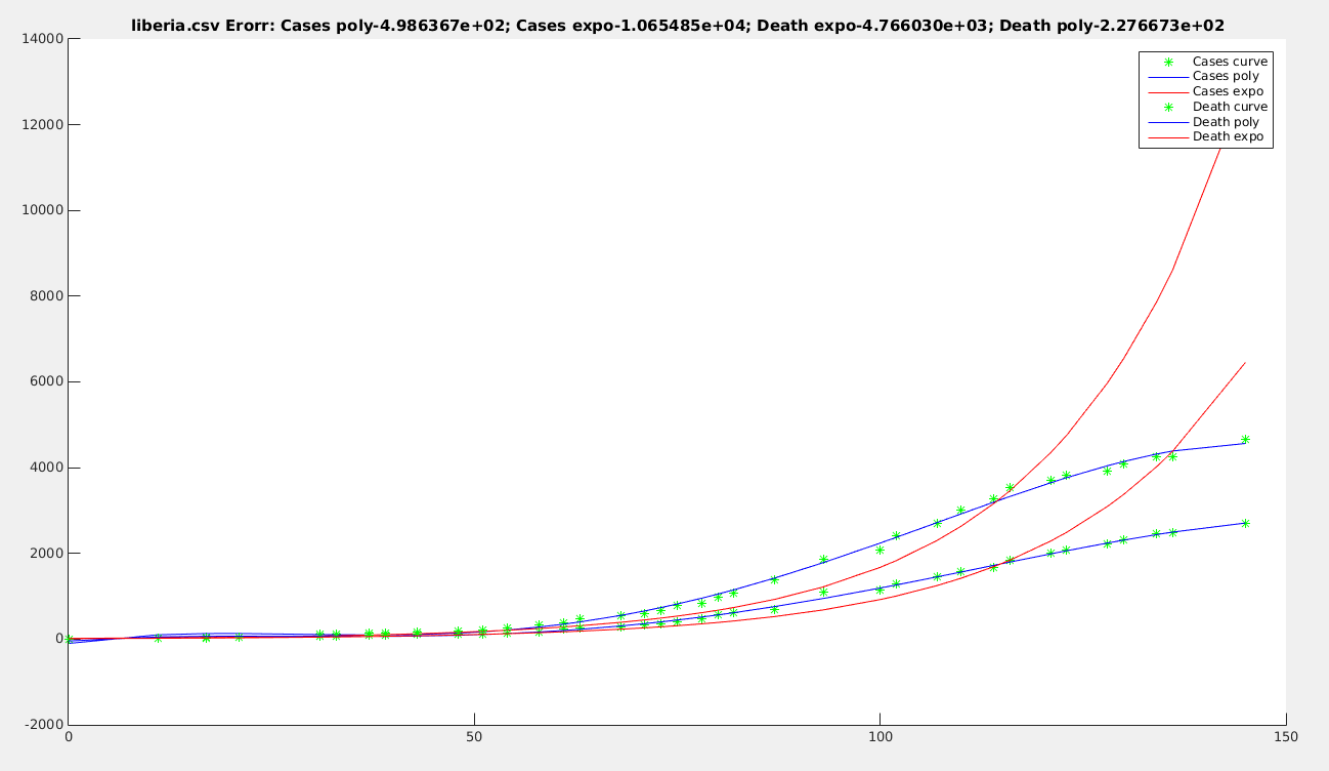
QUESTION 2:

c) Coefficients for 4 degree polynomial fit exponential fit:

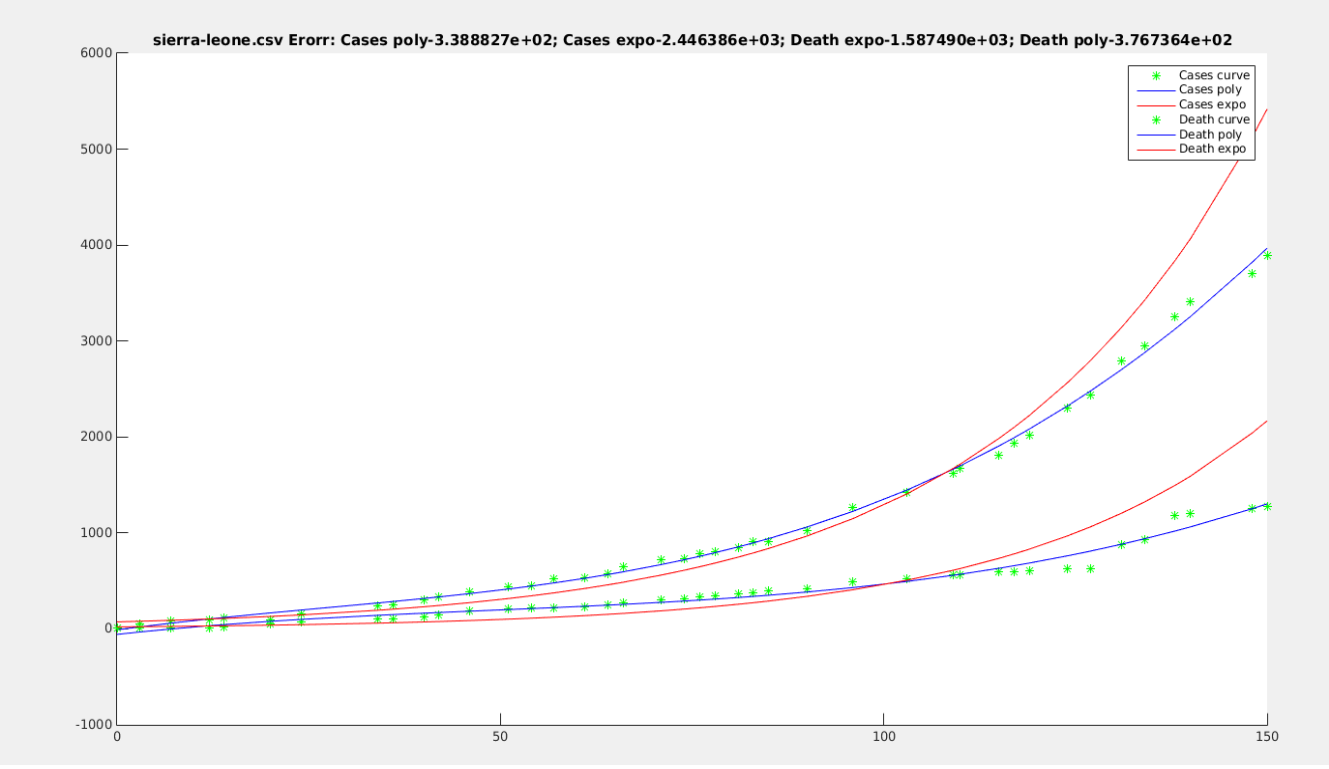
Name	cases_poly	cases_expo	death_poly	death_expo
GUINEA	-0.0000 0.0007 -0.1143 8.5771 63.3879	0.0118 4.7876	0.0000 0.0001 -0.0279 3.5991 53.1402	0.0115 4.4142
LIBERIA	-0.0001 0.0194 -1.2931 30.2170 -99.6852	0.0454 2.8798	-0.0000 0.0088 -0.5858 13.9945 -42.7402	0.0432 2.5062
SIERRA LEONE	0.0000 0.0010 -0.0731 9.0056 0.7709	0.0287 4.2917	0.0000 -0.0025 0.2100 -1.7810 8.2319	0.0309 3.0423



The polynomial fit is better than exponential fit for both cases and deaths for guinea



The polynomial fit is better than exponential fit for both cases and deaths for liberia.



The polynomial fit is better than exponential fit for both cases and deaths for sierra-loeone as well.

```
clear all;
folder = '/home/kami/Documents/FALL/modelling/hw2/Homework2Updated/';
csvfile = { 'guinea.csv' 'liberia.csv' 'sierra-leone.csv' };

for i=1:3
   filename = strcat(folder,csvfile{i});
   clear data time dateNum;
   data = csvread( filename, 1, 0 );
   %figure
   time = 1:size(data,1);
   [rows,cols] = size(data);
   for j=1:rows
        dateNum(j) = datenum(2014,data(j,1),data(j,2)) -
datenum(2014,data(1,1),data(1,2));
   end
    times = [dateNum(:).^4, dateNum(:).^3, dateNum(:).^2, dateNum(:).^1];
    times(:,5) = times(:,3) ./ times(:,3);
    times(1,5) = 1;
    cases_poly = pinv(times)* data(:,3);
    death_poly = pinv(times) * data(:,4) ;
    times_expo = dateNum(:);
    times_expo(:,2) = times_expo(:,1) ./ times_expo(:,1);
    times_expo(1,2) = 1;
    cases_expo = pinv(times_expo) * log(data(:,3));
    death_expo = pinv(times_expo) * log(data(:,4));
    figure;
    %Plot cases poly and expo
    hold all;
    plot(dateNum(:),data(:,3),'g*');
    plot(dateNum(:),times*cases_poly,'b-');
    plot(dateNum(:),exp(times_expo*cases_expo),'r');
    r_cases_poly = norm((times*cases_poly) - data(:,3));
    r_cases_expo = norm(exp(times_expo*cases_expo) - (data(:,3)));

    %Plot deaths poly and expo
    plot(dateNum(:),data(:,4),'g*');
    plot(dateNum(:),times*death_poly,'b-');
    plot(dateNum(:),exp(times_expo*death_expo),'r');
    r_death_poly = norm((times*death_poly) - data(:,4));
    r_death_expo = norm(exp(times_expo*death_expo) - (data(:,4)));

    title(strcat(csvfile{i},sprintf(' Erorr: Cases poly-
%d',r_cases_poly),sprintf('; Cases expo-%d',r_cases_expo),sprintf('; Death expo-
%d',r_death_expo),sprintf('; Death poly-%d',r_death_poly)));
    legend('Cases curve','Cases poly', 'Cases expo','Death curve','Death
poly','Death expo');
    hold off;

   %plot( time, data(:,3), 'r.-', time, data(:,4), 'b.-')
   %title( csvfile{i}, 'FontSize', 24 )
end
```

QUESTION 3:

a)

```
clear all;
folder = '/home/kami/Documents/FALL/modelling/hw2/Homework2Updated/';
csvfile = 'mpg.csv';
filename = strcat(folder,csvfile);
data = csvread( filename, 1, 0 );
mpg = data(:,8);
displacement = data(:,6);
%part 1: coefficients and error
A = [log(displacement), ones(7287,1)];
x = A \ log(mpg)
A1 = A*x ;
R = norm(A1-log(mpg));
R2_a = norm(A1)/norm(log(mpg));
```

```
x =

   -0.6237
    3.8273
R = 12.9807
```

b)

```
cylinders = data(:,5);
classsize = data(:,10);
guzzler = data(:,11);
A_b = [ones(7287,1), displacement, cylinders, classsize, guzzler];
mpg_1 = mpg.^-1;
x_b = A_b \ mpg_1
A1_b = A_b * x_b;
norm(mpg_1-A1_b)
R2_b = norm(A1_b)/norm(mpg_1);
```

```
>> x_b

x_b =

    0.0157
    0.0060
    0.0014
    0.0007
    0.0037

>>R2_b

R2_b = 0.9921
```

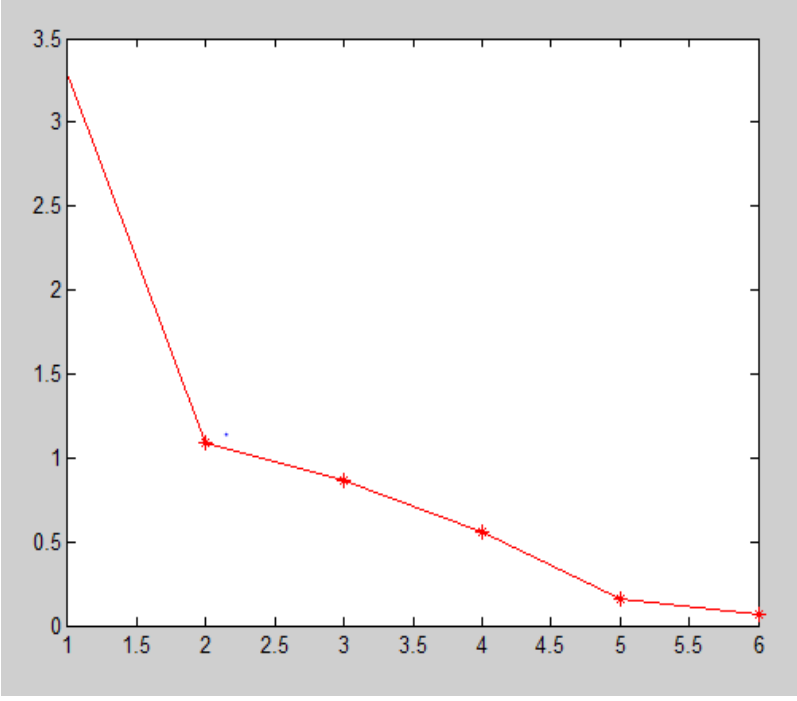
c)

```
origin=data(:,3);
auto_trans=data(:,4);
drive=data(:,7);

A=[];

r_max=0;

flag=0;

for i=0:1 %origin
    for j=0:1 %auto_trans
        for k=0:1 %cyl
            for l=0:1 %displ
                for m=0:1 %drive
                    for n=0:1 %cl_sz
                        A=[];
                        if(i==1)
                            A=[A origin];flag=1;
                        end
                        if(j==1)
                            A=[A auto_trans];flag=1;
                        end
                        if(k==1)
                            A=[A cylinders];flag=1;
                        end
                        if(l==1)
                            A=[A displacement];flag=1;
                        end
                        if(m==1)
                            A=[A drive];flag=1;
                        end
                        if(n==1)
                            A=[A classsize];flag=1;
                        end
                        if(flag==1)
                            A=[A classsize.^0]; %Just adding the 1s
                            y=mpg.^-1;
                            A_pseudo=pinv(A);
                            x_coeff=A_pseudo*y;
                            new_y=A*x_coeff;
                            r_square_six=norm(new_y)/norm(y);
                            if(r_square_six>r_max)
                                r_max=r_square_six;
                                r_set=[i j k l m n];
                            end
                            flag=0;
                        end
                    end
                end
            end
        end
    end
end
```

```
Maximum R-Squared Error for 1/(cityMPG): 0.992845
Sets included (1 1 1 1 1 1)
The best fit is with all the 6 variables taken together.
```

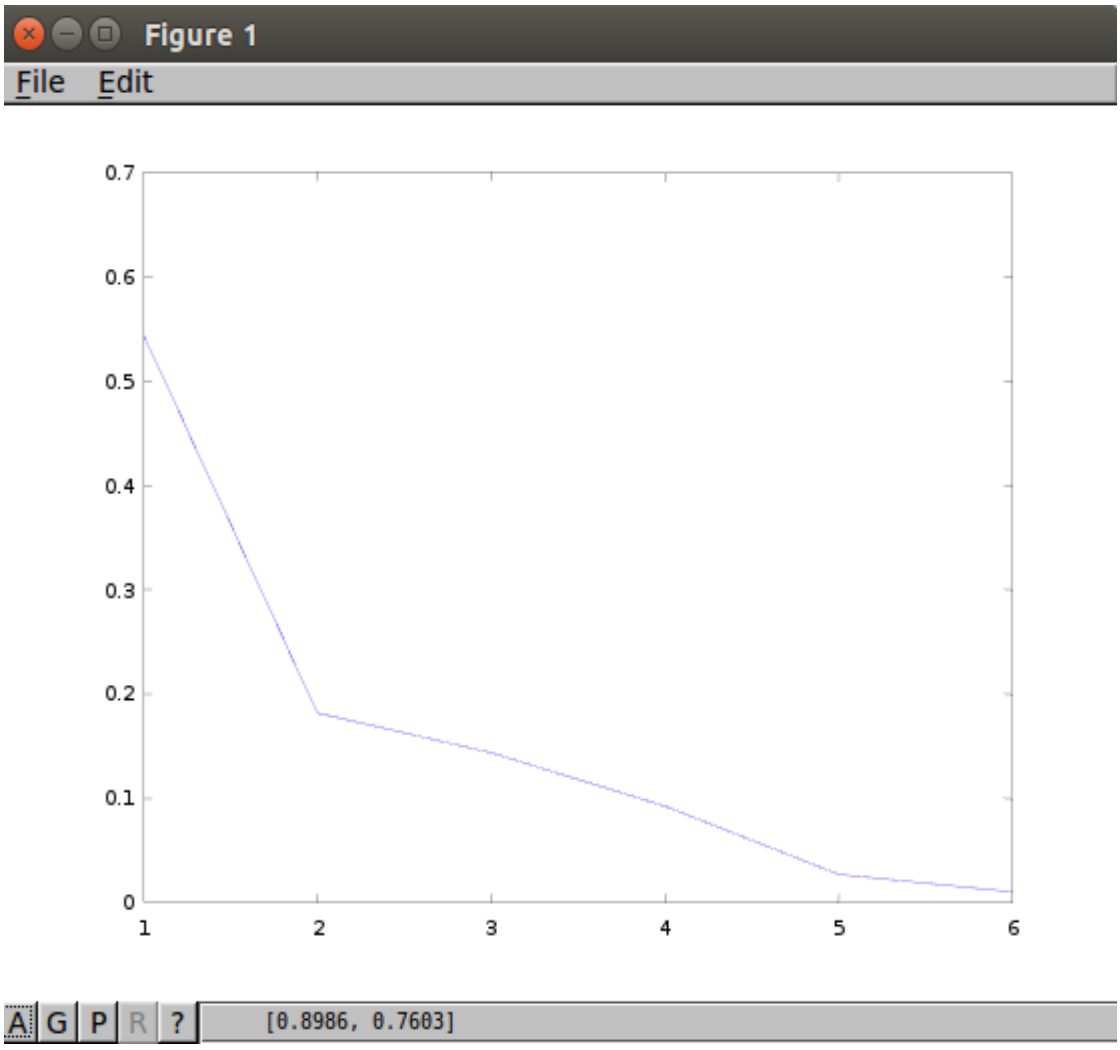
d)

```
origin = data(:,3);
drive = data(:,7);
autotrans = data(:,4);
A_3 = [origin, autotrans, cylinders, displacement, drive, classsize];


year = data(:,2);
A_d = [cylinders, displacement, drive, classsize, year, mpg_1];
A_cor = corr(A_d);
[u,s,v] = svd(A_cor);

u =

  -4.9397e-01  -1.8880e-01   3.4746e-01  -3.0601e-02  -4.7278e-01  -6.1250e-01
  -5.0976e-01  -1.1387e-01   3.2562e-01  -9.5907e-02  -1.6661e-01   7.6432e-01
  -3.3615e-01   4.7453e-01  -2.9199e-01   7.4484e-01  -1.4385e-01   3.3003e-02
  -2.9443e-01   5.5786e-01  -3.9397e-01  -6.5944e-01  -9.7887e-02  -4.9501e-02
   1.7321e-01   6.3258e-01   7.2989e-01  -7.3703e-04   1.8254e-01  -6.1489e-02
  -5.1619e-01  -1.2182e-01   5.7037e-03   1.4840e-02   8.2773e-01  -1.8256e-01

octave:236> s
s =

Diagonal Matrix

   3.264634          0          0          0          0          0
          0   1.091556          0          0          0          0
          0          0   0.862701          0          0          0
          0          0          0   0.555754          0          0
          0          0          0          0   0.163063          0
          0          0          0          0          0   0.062291

octave:237> v
v =

  -4.9397e-01  -1.8880e-01   3.4746e-01  -3.0601e-02  -4.7278e-01  -6.1250e-01
  -5.0976e-01  -1.1387e-01   3.2562e-01  -9.5907e-02  -1.6661e-01   7.6432e-01
  -3.3615e-01   4.7453e-01  -2.9199e-01   7.4484e-01  -1.4385e-01   3.3003e-02
  -2.9443e-01   5.5786e-01  -3.9397e-01  -6.5944e-01  -9.7887e-02  -4.9501e-02
   1.7321e-01   6.3258e-01   7.2989e-01  -7.3703e-04   1.8254e-01  -6.1489e-02
  -5.1619e-01  -1.2182e-01   5.7037e-03   1.4840e-02   8.2773e-01  -1.8256e-01

e)plot(1:6, s);
Plot of singular values
```



```
for i=1:6
  variance(i) = s(i,i)/trace(s);
end
plot(1:6, variance);
```

Plot of ratio of singular values.



Elbow is at 2.

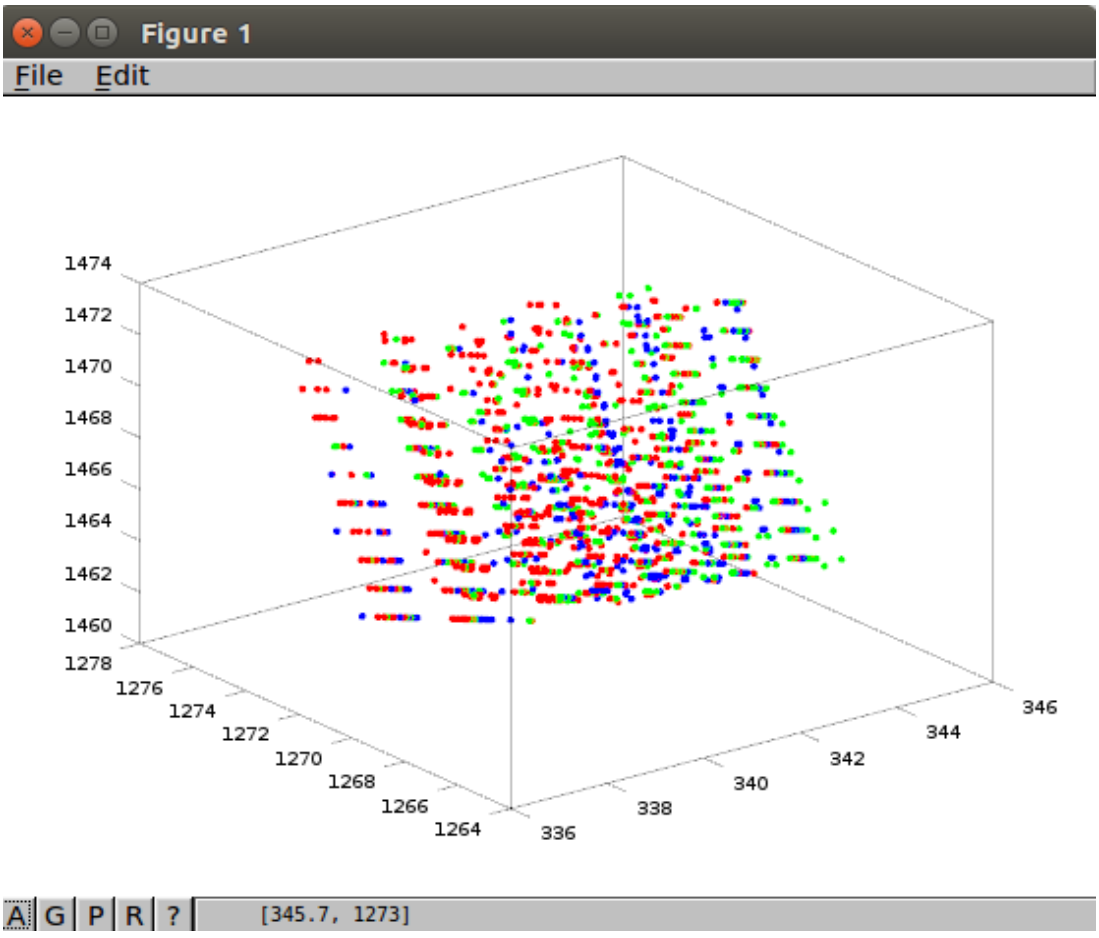
f)
The first 3 eigen vectors are:

```
octave:31> u(:,1:3)
ans =

  -0.4939721  -0.1888034   0.3474625
  -0.5097648  -0.1138690   0.3256194
  -0.3361468   0.4745285  -0.2919902
  -0.2944298   0.5578587  -0.3939745
   0.1732060   0.6325838   0.7298870
  -0.5161855  -0.1218247   0.0057037
```

The dominant loadings are:
First eiegen vector = -0.5161 = 1/cityMPG
Second eigen vector = 0.6325 = year
Third eigen vector = 0.72988 = year
This can be interpreted as: the fuel economy depends on the year in which the vehicle was manufactured and the city gallons/mile value of the engine.

g)

```
pc = A_d * u(:,1:3);

c1=0;c2=0;c3=0;
for i=1:rows(pc)
     if origin(i) == 1
         c1 = c1+1;
         color1(c1,:) = pc(i,:);
     elseif origin(i) == 2
         c2 = c2 +1;
         color2(c2,:) = pc(i,:);
     else
         c3 = c3 + 1;
         color3(c3,:) = pc(i,:);
     end
end

plot3(color3(:,1),color3(:,2),color3(:,3),'g.');
hold on
plot3(color2(:,1),color2(:,2),color2(:,3),'b.');
hold on
plot3(color1(:,1),color1(:,2),color1(:,3),'r.');
```



The 3 principal components do not separate the data which is visible in the above

plot. Hence the column origin is not an separating factor in the data.
QUESTION 4:

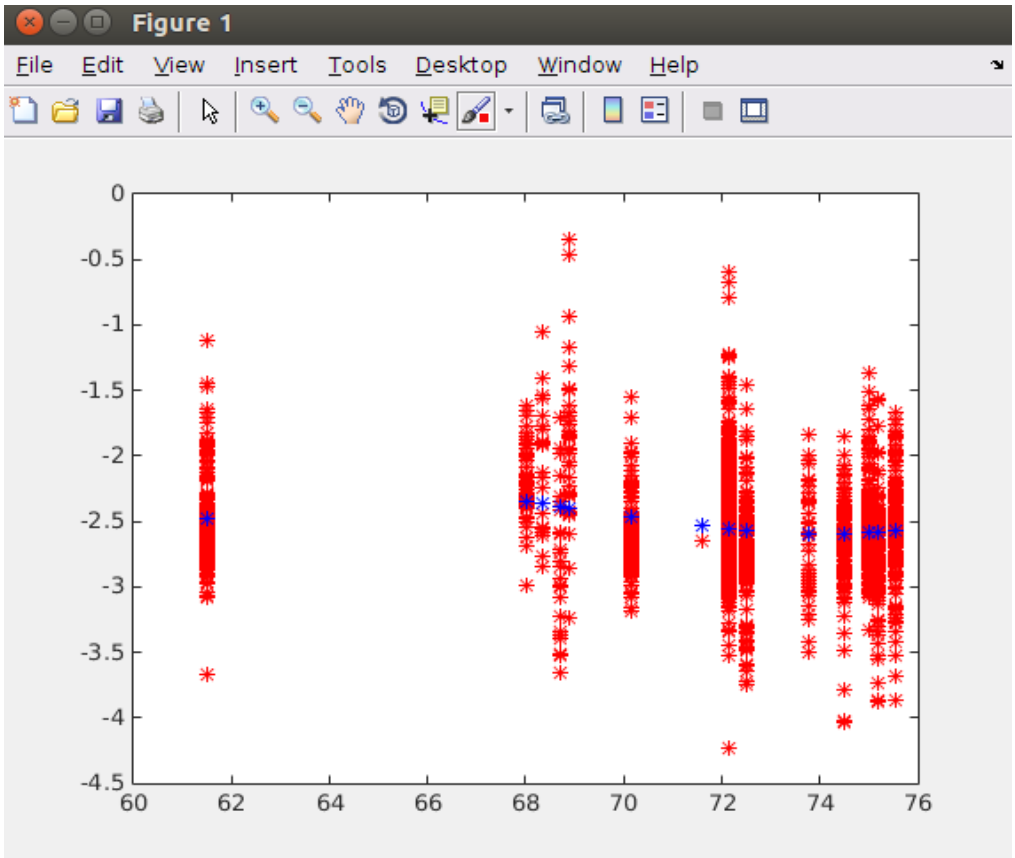
a)

```
clear all;
folder = '/home/kami/Documents/FALL/modelling/hw2/Homework2Updated/';
csvfile = 'LApower.csv';
filename = strcat(folder,csvfile);
data = csvread( filename, 1, 0 );

logPowerJul = data(:,7);
logSqMts = data(:,27);
avglogTempJul = data(:,19);
b = logPowerJul - logSqMts;
a = [avglogTempJul.^3, avglogTempJul.^2, avglogTempJul.^1,ones(2322,1)];
x = a \ b;
```

```
x =

    0.0009
   -0.1936
   13.4209
 -311.2345
```

b)

```
plot( data(:,19),logPowerJul - logSqMts,'r*');
hold on;
plot( data(:,19), a*x,'b*');
```



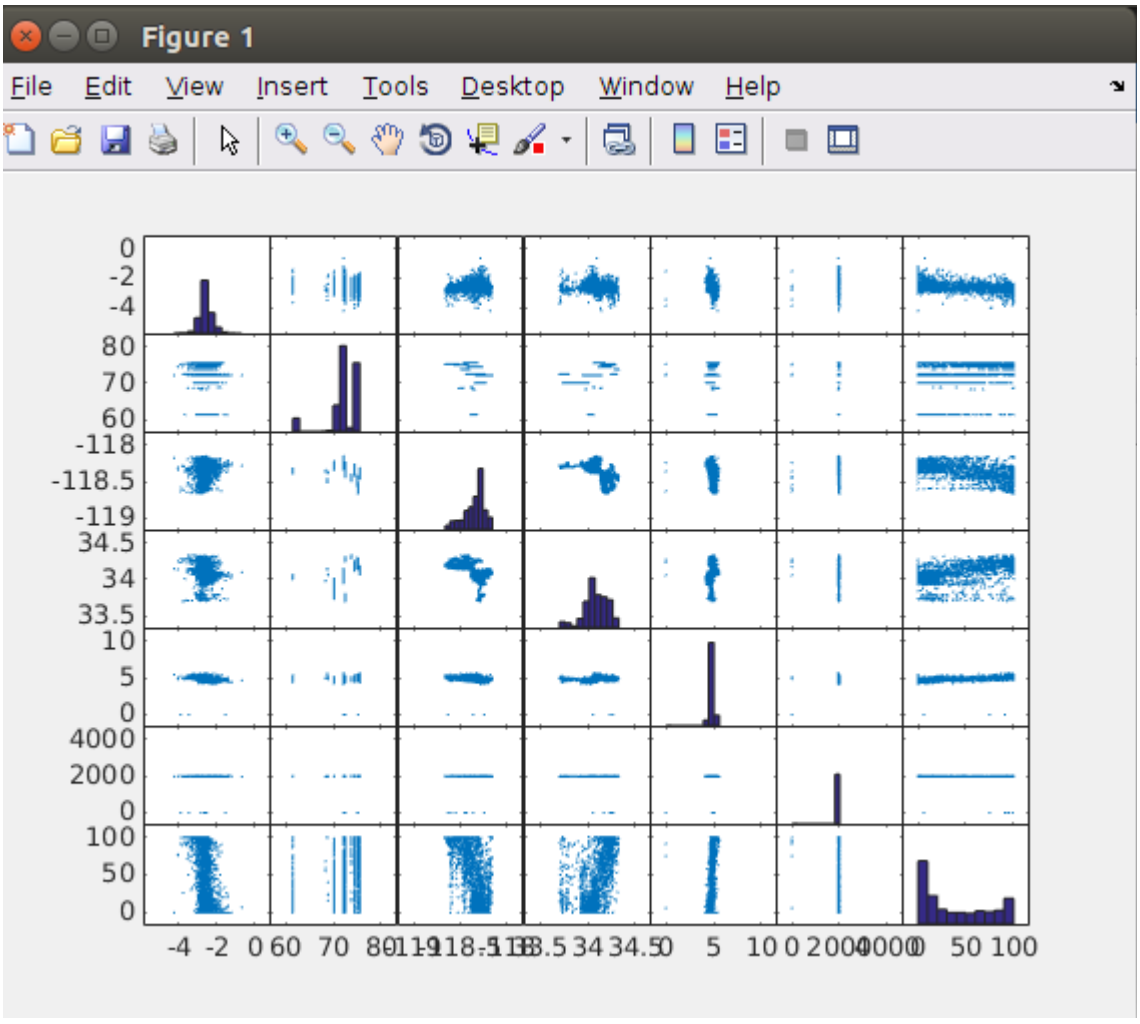
c)

```
b1 = logPowerJul;
a1 = [ones(2322,1), avglogTempJul, data(:,29), data(:,28), logSqMts];
x2 = a1 \b1;
b1_recon = a1*x2;
error = norm(b1_recon)/norm(b1);
```

```
x2 =

    0.5933
   -0.0003
   -0.0001
    0.0151
    0.4522


error =

    0.9944
```

d)

```
count = 0;
for i=1:2322
    if data(1,30) > 50
        count = count +1;
        matrix(count,1:7) = [logPowerJul(i,1) - logSqMts(i,1), avglogTempJul(i,1),
data(i,25), data(i,26), data(i,28), data(i,29), data(i,38)];
    end
end
plotmatrix(matrix);
```



e)

```
corr_data = corr(matrix);
[u,s,v] = svds(corr_data,3 )
```

```
octave:366> corr_data
corr_data =

   1.0000e+00  -1.4713e-02   1.1919e-01  -5.3941e-02  -2.0106e-01   3.2594e-02  -3.4383e-01
  -1.4713e-02   1.0000e+00  -3.4876e-01   4.1746e-01   1.6299e-01   4.1257e-04   3.4425e-01
   1.1919e-01  -3.4876e-01   1.0000e+00  -5.8200e-01  -2.9617e-01   2.0962e-02  -6.2075e-01
  -5.3941e-02   4.1746e-01  -5.8200e-01   1.0000e+00   2.0234e-01  -3.1553e-02   4.8836e-01
  -2.0106e-01   1.6299e-01  -2.9617e-01   2.0234e-01   1.0000e+00   4.0938e-01   4.7079e-01
   3.2594e-02   4.1257e-04   2.0962e-02  -3.1553e-02   4.0938e-01   1.0000e+00  -3.7132e-02
  -3.4383e-01   3.4425e-01  -6.2075e-01   4.8836e-01   4.7079e-01  -3.7132e-02   1.0000e+00

u =

  -0.1970757  -0.1572453  -0.8032445
   0.3517073  -0.2102558  -0.3423795
  -0.4901318   0.1619316   0.0682201
   0.4503140  -0.2545647  -0.2219935
   0.3528940   0.5682655  -0.0060950
   0.0556704   0.7190736  -0.3626823
   0.5165442  -0.0069838   0.2281751

s =

Diagonal Matrix

   2.6980        0        0
        0   1.3325        0
        0        0   1.0796

v =

  -0.1970757  -0.1572453  -0.8032445
   0.3517073  -0.2102558  -0.3423795
  -0.4901318   0.1619316   0.0682201
   0.4503140  -0.2545647  -0.2219935
   0.3528940   0.5682655  -0.0060950
   0.0556704   0.7190736  -0.3626823
   0.5165442  -0.0069838   0.2281751
```

Dominant loadings are marked in **Bold**. In the first principal component, the max eigen vector value corresponds to last element I.e : Percent SFR. Hence this part shows the max spread. Similarly second principal component has maximum spread about the AvgYearBuilt. And the third has the max spread about logPowerJul(i,1) – logSqMts(i,1). In simpler words this means that the variation in the data is best shown by Percent SFR followed by Avg year built and logPowerJul(i,1) – logSqMts(i,1).

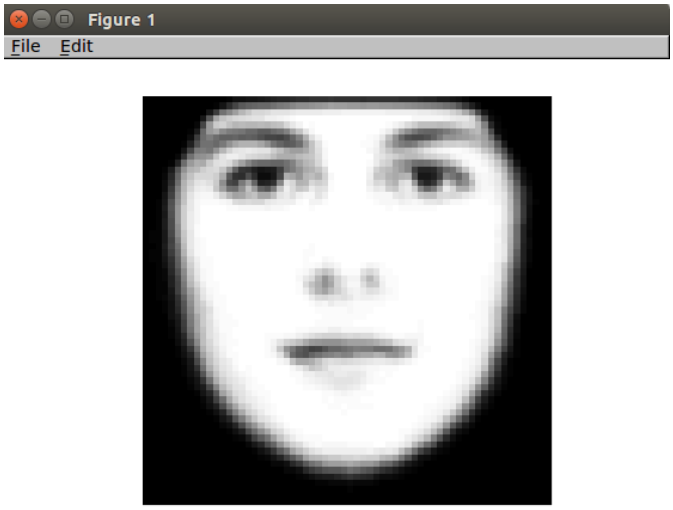
QUESTION 5:

a)

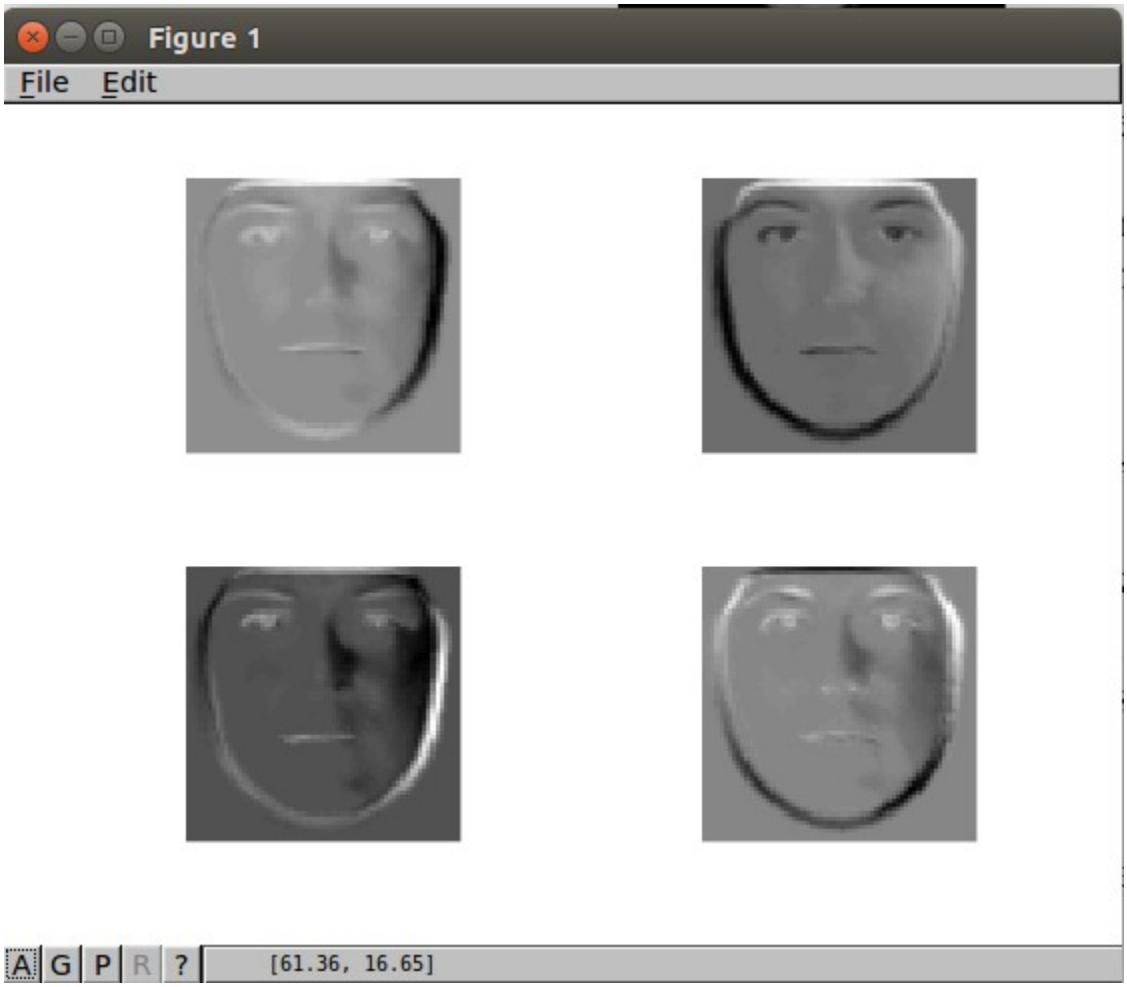
```
function [ avgFace, ksingularValues, eigenFaces ] = eigFaces( loc, s, k )
      %row wise
      % Read all the images

      filelist = readdir(loc);
      for i=1:s
       x = imread(strcat(loc,filelist(i+2,1))(1,1){1});
       a = reshape(x,1,64*64);
       images(i,:) = a;
      end

      %Calculate mean
      [rows,cols] = size(images);
      for i=1:cols
        mean(i) = round(sum(images(:,i)/rows));
      end

      im = reshape(mean,64,64);
      im=(im-min(im(:)))/(max(im(:))-min(im(:)));
      imshow(im);
        avgFace = im;

      %Create caricature
      for i=1:rows
      caricature(i,:) =  images(i,:) - mean(1,:);
      end

      replacement = cov(caricature);
      [u,l,v] = svds(replacement,k);
      ksingularValues = diag(l)
      eigenFaces = v;

      for i=1:k
        figure
        im = reshape(eigenFaces(:,i),64,64);
        im=(im-min(im(:)))/(max(im(:))-min(im(:)));
        imshow(im);
        axis off
      end
end
```

```
ocatve:4> eigFaces('/home/Documents/FALL/hw2/Homework2Updated/eigenfaces/', 177,4);
ksingularValues =

   1.5880e+05
   1.1480e+05
   7.6793e+04
   5.9086e+04
```

Mean Face :



Eigen Faces for k=4 , sample size = 177

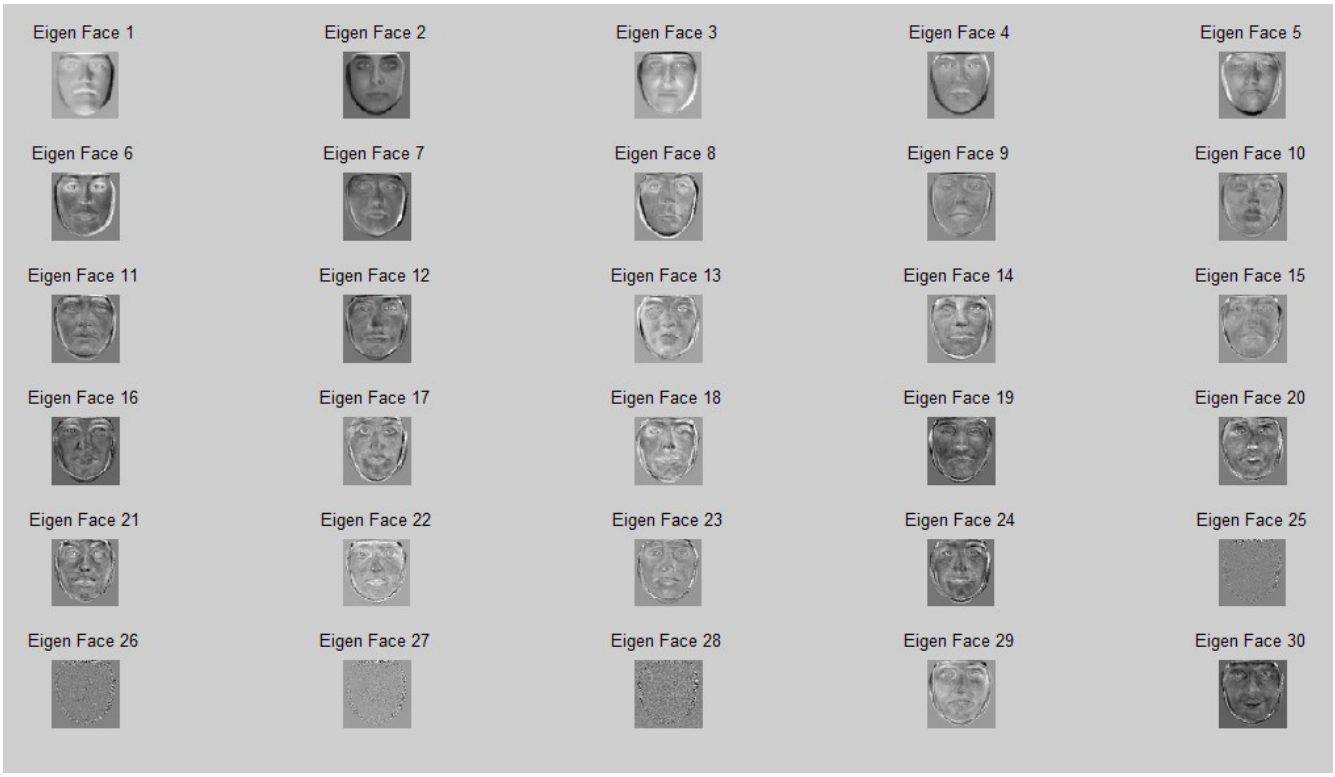


b)

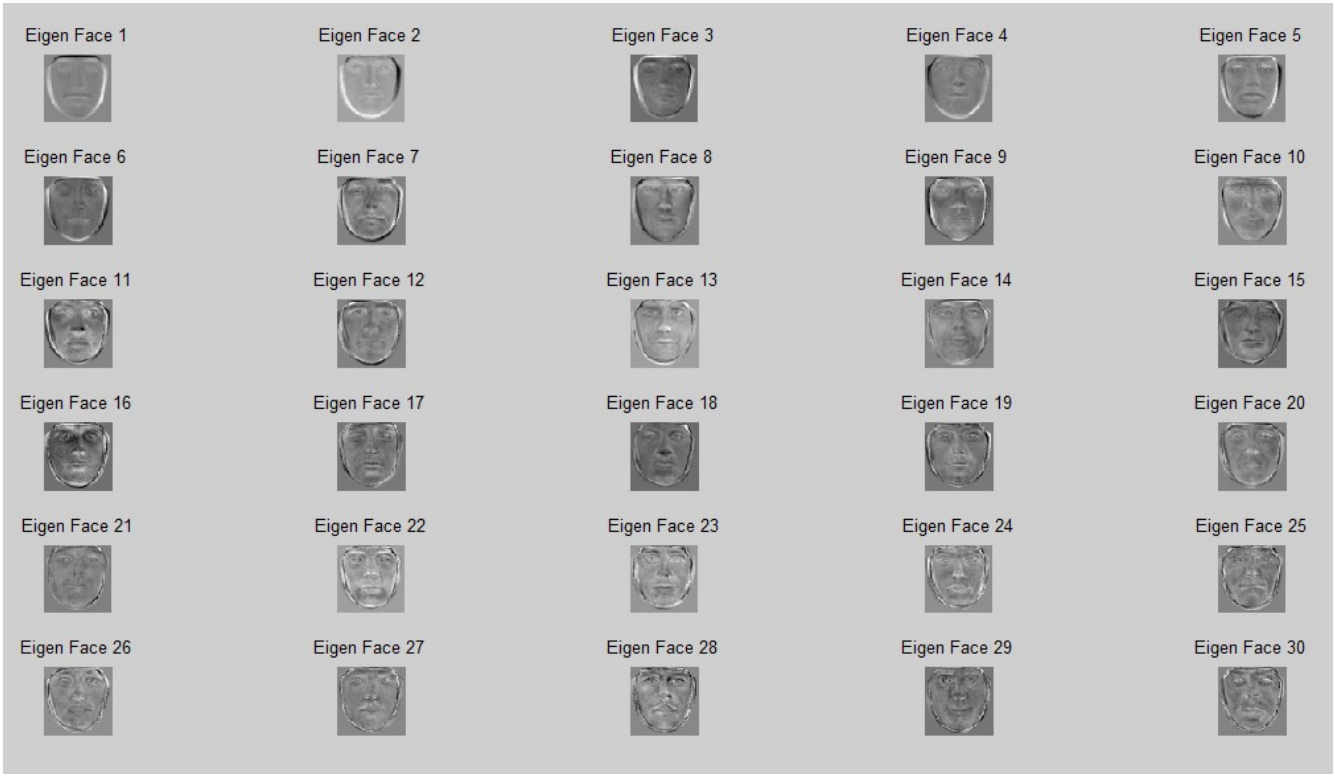
```
function [ avgFace, ksingularValues, eigenFaces ] = eigFaces( loc,k )
      %row wise
      % Read all the images

      filelist = readdir(loc);
      cf =0; cm=0;
      for i=1:168
       x = imread(strcat(loc,filelist(i+2,1))(1,1){1});
       a = reshape(x,1,64*64);
       images(i,:) = a;
         if  strcmp(face_features{i,2}, 'Female') && strcmp(face_features{i,3},
'Blue')
           cf += 1;
               images_female_blue(cf,:) = a;
       end

       if  strcmp(face_features{i,2}, 'Male') && strcmp(face_features{i,3}, 'Blue')
           cm += 1;
               images_male_blue(cm,:) = a;
       end
      end

      %Calculate mean
```

```
      [rows,cols] = size(images_male_blue);
      for i=1:cols
        mean_m(i) = round(sum(images_male_blue(:,i)/rows));
      end

      [rows,cols] = size(images_female_blue);
      for i=1:cols
        mean_f(i) = round(sum(images_female_blue(:,i)/rows));
      end

      %Create caricature
      for i=1:rows
      caricature_f(i,:) =  images_female_blue(i,:) - mean_f(1,:);
      end

      for i=1:rows
      caricature_m(i,:) =  images_male_blue(i,:) - mean_m(1,:);
      end

      replacement = cov(caricature_f);
      [uf,lf,vf] = svds(replacement,k);
      ksingularValues = diag(lf)
      eigenFaces = vf;

      replacement = cov(caricature_m);
      [um,lm,vm] = svds(replacement,k);
      ksingularValues = diag(lm)
      eigenFaces = vm;

      for i=1:k
        figure
          %subplot(6,5,i);
        im = reshape(eigenFaces(:,i),64,64);
        im=(im-min(im(:)))/(max(im(:))-min(im(:)));
        imshow(im);
        axis off
      end
end
```

FEMALE K=30 EIGEN FACES



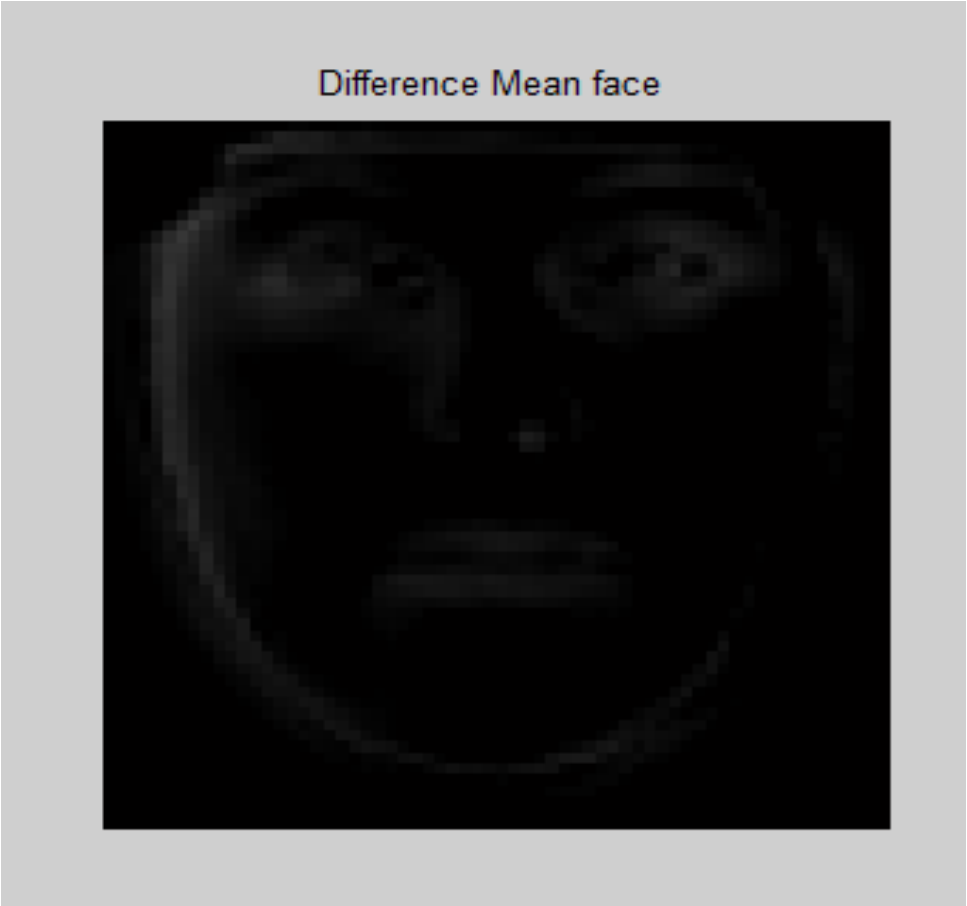
MALE K=30 EIGEN FACES



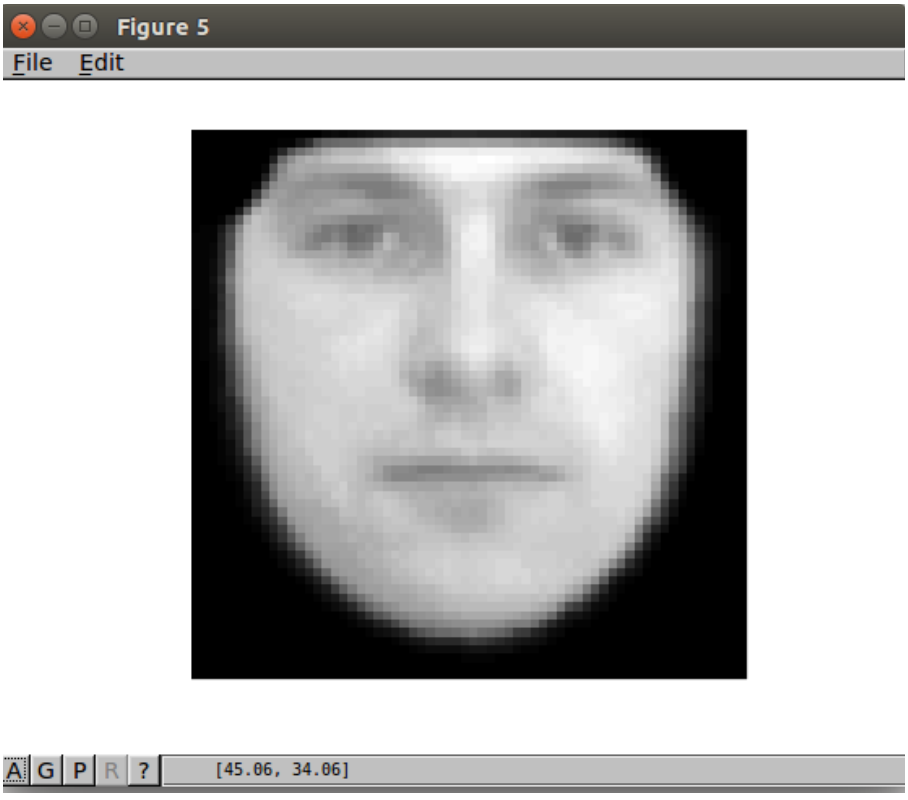

```
function [ avgFace ] = eigFaces( loc, face_features, image_to_omit )
      %row wise
      % Read all the images

      filelist = readdir(loc);
```

```
      cf = 0; cm = 0;
      for i=1:168
       x = imread(strcat(loc,filelist(i+2,1))(1,1){1});
       a = reshape(x,1,64*64);
       images(i,:) = a;
         if  strcmp(face_features{i,2}, 'Female') && strcmp(face_features{i,3},
'Blue')
           flag = 1;
         for j = 1:10
            if strcmp(filelist(i+2,1),image_to_omit{j,1})
                flag = 0;
            end
         if flag == 1
            cf += 1;
            images(cf,:) = a;
         end
       end

       if  strcmp(face_features{i,2}, 'Male') && strcmp(face_features{i,3}, 'Blue')
           flag = 1;
         for j = 1:10
            if strcmp(filelist(i+2,1),image_to_omit{j,1})
                flag = 0;
            end
         if flag == 1
            cf += 1;
            images(cf,:) = a;
         end
       end
      end

      %Calculate mean
      [rows,cols] = size(images_female_blue);
      for i=1:cols
        mean_f(i) = round(sum(images_female_blue(:,i)/rows));
      end
      [rows,cols] = size(images_male_blue);
      for i=1:cols
        mean_m(i) = round(sum(images_male_blue(:,i)/rows));
      end

      figure
      im = reshape(mean_f,64,64);
      im=(im-min(im(:)))/(max(im(:))-min(im(:)));
      imshow(im);
        avgFace = im;

      figure
      im = reshape(mean_m,64,64);
      im=(im-min(im(:)))/(max(im(:))-min(im(:)));
      imshow(im);

      figure
      im = reshape(mean_f-mean_m,64,64);
      im=(im-min(im(:)))/(max(im(:))-min(im(:)));
      imshow(im);
end
```

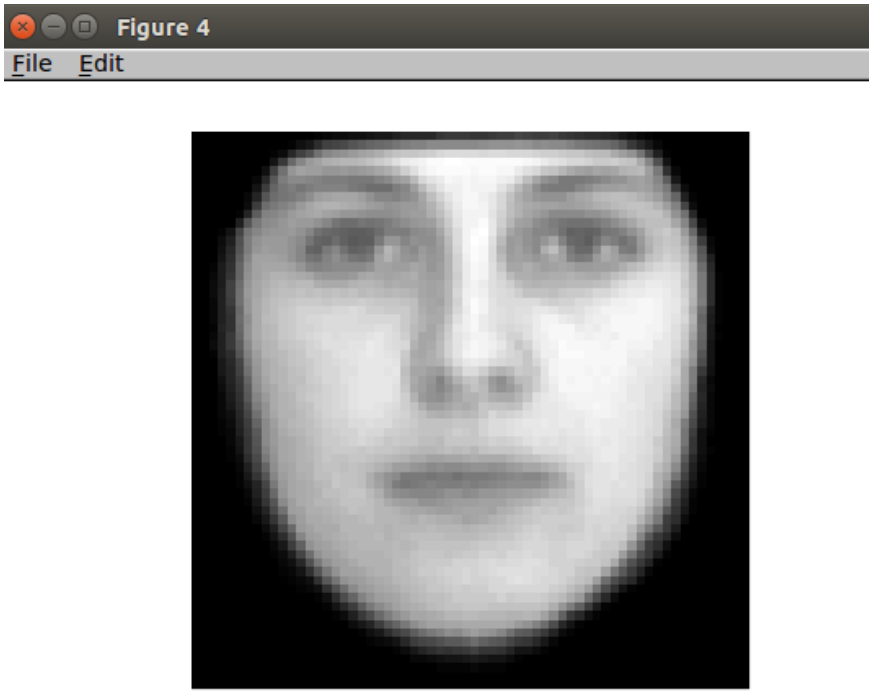
Female mean- Male mean



Avg Male Face



Avg Female Face




```
female_singular_Values =

   run_eigen_faces
    1.0e+05 *

     8.7041
     5.3455
     2.7811
     2.0929
     1.6299
     1.3608
     1.0906
     0.9244
     0.7754
     0.6539
     0.5806
     0.4975
     0.4802
     0.4248
     0.3715
     0.3484
     0.3117
     0.2819
     0.2632
     0.2287
     0.1958
     0.1649
     0.1598
     0.0000
     0.0000
```

```
    0.0000
    0.0000
         0
         0
         0

   1.0e+05 *

    4.2833
    3.2753
    1.7863
    1.2890
    1.0972
    0.9487
    0.6857
    0.6806
    0.5425
    0.4383
    0.4070
    0.3817
    0.3633
    0.3147
    0.2977
    0.2884
    0.2602
    0.2363
    0.2147
    0.2064
    0.1878
    0.1766
    0.1603
    0.1372
    0.1347
    0.1215
    0.1202
    0.1137
    0.1045
   0.0985
```

c)
```
function [ female_face, male_male ] = eigFaces( loc, face_features, k  )
    %row wise
    % Read all the images

    filelist = readdir(loc);
    cf = 0; cm = 0; count = 0;
    for i=1:177
     x = imread(strcat(loc,filelist(i+2,1))(1,1){1});
     a = reshape(x,1,64*64);
       if  strcmp(face_features{i,2}, 'Female') && strcmp(face_features{i,3}, 'Blue')
         cf += 1;
       count += 1;
       images_female_blue(cf) = count;
       images(count,:) = a;
     end

     if  strcmp(face_features{i,2}, 'Male') && strcmp(face_features{i,3}, 'Blue')
         cm += 1;
       count += 1;
       images_male_blue(cm) = count;
       images(count,:) = a;
     end
    end

    %Calculate mean
    [rows,cols] = size(images);
    for i=1:cols
      mean(i) = round(sum(images(:,i)/rows));
```

```
        end

        %Create caricature
        [rows,cols] = size(images);
        for i=1:rows
        caricature(i,:) =  images(i,:) - mean(1,:);
        end

        replacement = cov(caricature);
        [u,l,v] = svds(replacement,k);
        c = v' * double(caricature');
        [rows,cols] = sizeof(c)

        max = 0;
        [rows,cols] = size(images_female_blue);
        for i=1:rows
         index = images_female_blue(i,1);
         coeff = norm(c(:,index));
         if (max < coeff)
           max = coeff;
             coeff_female = c(:,index);
             %female_face = images(index,:);
            end
        end

        female_face = mean + v' * coeff_female;
        figure
        im = reshape(female_face,64,64);
        imshow(im);

        max = 0;
        [rows,cols] = size(images_male_blue);
        for i=1:rows
         index = images_male_blue(i,1);
         coeff = norm(c(:,index));
         if (max < coeff)
           max = coeff;
           coeff_male = c(:,index);
             %male_face = images(index,:);
            end
        end

        male_face = mean + v' * coeff_male;
        figure
        im = reshape(male_face,64,64);
        imshow(im);
end
```

Most extreme female

Most extreme male

d)
To classify images I calculated the squared difference Zm, Zf between each face and the mean female and male faces. A face is classified as female face if Zf< Zm else as male face. After comparing the classification with the real values from the face_features.m we can tell weather a decesion is right ot wrong. At the end, after classifying all images percentage of success is calculated.

```
function [ decesion ] = eigFaces( filename, loc,face_features )
       %row wise
       % Read all the images

       filelist = readdir(loc);
       cf = 0; cm = 0;
       for i=1:168
        x = imread(strcat(loc,filelist(i+2,1))(1,1){1});
        a = reshape(x,1,64*64);
        images(i,:) = a;
          if  strcmp(face_features{i,2}, 'Female') && strcmp(face_features{i,3}, 'Blue')
            cf += 1;
           images_female_blue(cf,:) = a;
        end

        if  strcmp(face_features{i,2}, 'Male') && strcmp(face_features{i,3}, 'Blue')
            cm += 1;
          images_male_blue(cm,:) = a;
        end
       end

       %Calculate mean
       [rows,cols] = size(images_female_blue);
       for i=1:cols
         mean_f(i) = round(sum(images_female_blue(:,i)/rows));
       end
       [rows,cols] = size(images_male_blue);
       for i=1:cols
         mean_m(i) = round(sum(images_male_blue(:,i)/rows));
       end

       x = imread(filename);
       a = reshape(x,1,64*64);
       input_image = a;

       diff_f = norm(double(images(i,:)) - mean_f);
       diff_m = norm(double(images(i,:)) - mean_m);
       if diff_m < diff_f
          decesion = 0;
       else
          decesion = 1;
       end
end
```

Classifier which uses the difference between the mean and every face to find out the percentage of success foe this classifier

```
function [ percentage ] = eigFaces( loc, face_features )
      %row wise
      % Read all the images

      filelist = readdir(loc);
      cf = 0; cm = 0;
      for i=1:168
       x = imread(strcat(loc,filelist(i+2,1))(1,1){1});
       a = reshape(x,1,64*64);
       images(i,:) = a;
         if  strcmp(face_features{i,2}, 'Female') && strcmp(face_features{i,3}, 'Blue')
           cf += 1;
          images_female_blue(cf,:) = a;
       end

       if  strcmp(face_features{i,2}, 'Male') && strcmp(face_features{i,3}, 'Blue')
           cm += 1;
         images_male_blue(cm,:) = a;
       end
      end

      %Calculate mean
      [rows,cols] = size(images_female_blue);
      for i=1:cols
        mean_f(i) = round(sum(images_female_blue(:,i)/rows));
      end
      [rows,cols] = size(images_male_blue);
      for i=1:cols
        mean_m(i) = round(sum(images_male_blue(:,i)/rows));
      end

      true =0; false =0;
      for i=1:168
        diff_f = norm(double(images(i,:)) - mean_f);
        diff_m = norm(double(images(i,:)) - mean_m);
        if (strcmp(face_features{i,2}, 'Male') && diff_m < diff_f) ||
( strcmp(face_features{i,2}, 'Female') && diff_f<diff_m)
            true += 1;
          else
            false +=1;
          end
         end
      percent = true/(true+false) * 100
end
```

```
Percentage of success :
percent =  64.286
```

QUESTION 1:

d)

```
octave:11> sort(roots(poly(hilb(7))),'descend')
ans =

   1.6609e+00
   2.7192e-01
   2.1290e-02
   1.0086e-03
```

```
   2.9386e-05
   4.8568e-07
   3.4939e-09

octave:12> sort(roots(poly(magic(7))),'descend')
ans =

   175.000
    56.485
    31.088
    25.397
   -25.397
   -31.088
   -56.485

octave:16> sort(roots(poly(pascal(7,1))),'descend')
ans =

   1.00013 + 0.00000i
  -1.00000 + 0.00000i
   1.00000 + 0.00013i
   1.00000 - 0.00013i
  -1.00000 + 0.00000i
  -1.00000 - 0.00000i
   0.99987 + 0.00000i

octave:17> svd(pascal(7,1))
ans =

   34.958043
    6.857853
    2.160743
    1.000000
    0.462804
    0.145818
    0.028606
```

They are not the same in the case of Pascal(7,1)

```
octave:18> sort(roots(poly(vander(1:7))),'descend')
ans =

   5.5645e+02
   1.4196e+02
   5.1654e+00
   1.0014e-02
  -3.7270e-01
  -3.5714e+01
  -4.5750e+02

octave:19> svd(vander(1:7))
ans =

   1.2899e+05
   1.4653e+03
   4.8179e+01
   4.0750e+00
   9.3089e-01
   1.3659e-01
   5.2738e-03
```

They are not the same in case of Vander(1:7)

```
octave:20> sort(roots(poly(rosser())),'descend')
ans =

   1.0201e+03 + 0.0000e+00i
  -1.0200e+03 + 0.0000e+00i
   1.0199e+03 + 3.8475e-02i
   1.0199e+03 - 3.8475e-02i
   1.0000e+03 + 4.8404e-03i
```

```
   1.0000e+03 - 4.8404e-03i
   9.8049e-02 + 0.0000e+00i
  -1.1304e-13 + 0.0000e+00i

octave:21> svd(rosser())
ans =

   1.0200e+03
   1.0200e+03
   1.0200e+03
   1.0199e+03
   1.0000e+03
   1.0000e+03
   9.8049e-02
   9.6347e-15

e)
octave:23> digits(50);
octave:23> s = svd(hilb(7));
octave:29> for i=1:7
> x = vpa(s(i))
> end
x =

1.6608853389269313538534333929419517517089843750
x =

0.27192019814934514920778951818647328764200210571289062500
x =

0.021289754908327924304289879842144728172570466995239257812500
x =

0.0010085876107701306769737215063287294469773769378662109375
x =

2.9386368145932812334826522704389617501874454319477081298830E-5
x =

4.8567633617416478580722065394592590337197179906070232391357421875E-7
x =

3.4938985950232211808140808022380557895658625966461841017007827758789062500E-9
octave:41> %cond num
octave:41>
1.6608853389269313538534333929419517517089843750/3.4938985950232211808140808022380557895
6586259664618410170078277587890625000E-9
ans =    4.7537e+08

octave:31> s = svd(magic(7))
s =

   175.000
    57.436
    57.436
    31.553
    31.553
    24.609
    24.609

octave:32> for i=1:7
> x = vpa(s(i))
> end
x =

175.0
x =

57.435623593261333041937177767977118492126464843750
x =
```

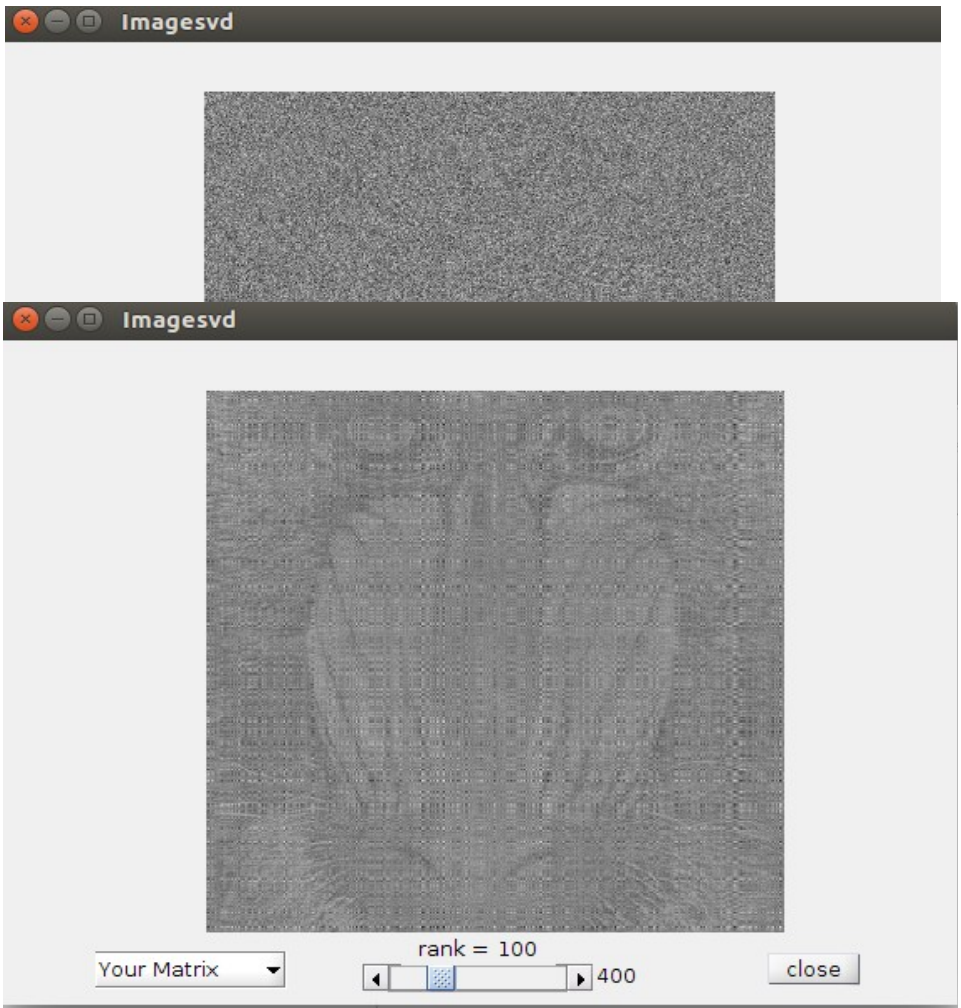
```
57.435623593261325936509820166975259780883789062 5
x =

31.552558854447305236590182175859808921813964843 75
x =

31.552558854447305236590182175859808921813964843 75
x =

24.608640193839217857885159901343286037445068359 375
x =

24.608640193839196541603087098337709903717041015625
octave:42> %cond num
octave:42> 175.0/24.608640193839196541603087098337709903717041015625
ans =  7.1113

octave:33> s = svd(pascal(7,1))
s =

   34.958043
    6.857853
    2.160743
    1.000000
    0.462804
    0.145818
    0.028606

octave:34> for i=1:7
> x = vpa(s(i))
> end
x =

34.958043109200062303898448590189218521118164062 5
x =

6.857852514958071132866734842536970973014831542968 75
x =

2.160743478394193672897927172016352415084838867187 5
x =

0.999999999999998889776975374843459576368331909179 6875
x =

0.462803664571591222287594291628920473158359527587 890625
x =

0.145818242346104659512562307099869940429925918579 1015625
x =

0.028605720202250765893392525640592793934047222137 451171875
octave:43> %cond num
octave:43>
34.958043109200062303898448590189218521118164062 5/0.028605720202250765893392525640592793934047222137 451171875
ans =  1222.1

octave:35> s = svd(vander(1:7))
s =

   1.2899e+05
   1.4653e+03
   4.8179e+01
   4.0750e+00
   9.3089e-01
   1.3659e-01
   5.2738e-03

octave:36> for i=1:7
> x = vpa(s(i))
```

```
> end
x =

128993.35754429368535056710243225097656 25
x =

1465.27325069777475619048345834016799926 7578125
x =

48.17945739251121040069847367703914642333984375
x =

4.07497724102072922391926113050431013107299 8046875
x =

0.93089037201777291574700257115182466804981231689453125
x =

0.13658648190401936894922130250051850453019142150878906 25
x =

0.00527383954289458410646940222932244068942964076995849609375

octave:37>%condition-num
128993.35754429368535056710243225097656 25/0.00527383954289458410646940222932244068942964076995849609375
ans =    2.4459e+07

octave:38> s = svd(rosser())
s =

   1.0200e+03
   1.0200e+03
   1.0200e+03
   1.0199e+03
   1.0000e+03
   1.0000e+03
   9.8049e-02
   9.6347e-15

octave:39> for i=1:8
> x = vpa(s(i))
> end
x =

1020.04901842999674954626243561506271362304 6875
x =

1020.04901842999652217258699238300323486328125
x =

1019.99999999999977262632455676794052124023 4375
x =

1019.90195135927865521807689219713211059570 3125
x =

999.99999999999977262632455676794052124023 4375
x =

999.99999999999658939486835151910781860351 5625
x =

0.09804864072158099574938461273632128722965717315673828125
x =

9.63472243888937619260813748519738697846724433115905839031 3E-15
octave:40> %cond num
octave:40>
1020.04901842999674954626243561506271362304 6875/9.63472243888937619260813748519738697846724433115905839031 3E-15
```

```
ans =    1.0587e+17

f)

function imagesvd(varargin)
% IMAGESVD   Principal component analysis of monochrome and color images.
%    IMAGESVD('file1.fmt','file2.fmt', ... ) reads the specified image
%    files.  Any format known to IMREAD is acceptable.
%    IMAGESVD, with no arguments, provides popup menu access to several
%    images from the NCM and demos directories.

%   Copyright 2013 Cleve Moler and The MathWorks, Inc.

%    imagesvd('slide') is the callback from the rank slider.
%    imagesvd('menu') is the callback from the popup menu.

if nargin == 0  || ~isequal(varargin{1},'slide')
   if nargin == 0  || ~isequal(varargin{1},'menu')

      % Initialize uicontrols

      shg
      clf
      set(gcf,'menu','none','numbertitle','off','name','Imagesvd');
      X = [];                       %Store numeric values in X if the input is a
matrix.
      if nargin > 0
         L = {'Your Matrix'};
         X = varargin{1};
      else
         L = {'detail.mat','durer.mat','fern.png','clown.mat', ...
             'earth.mat','mandrill.mat','gatlin.mat'};
      end
      startwith = 1;
      h.popup = uicontrol('units','norm','pos',[.10 .03 .20 .05], ...
         'style','popup','val',startwith,'string',L, ...
         'callback','imagesvd(''menu'')');
      h.slider = uicontrol('units','norm','pos',[.38 .02 .24 .04], ...
         'style','slider','value',0,'callback','imagesvd(''slide'')');
      h.limit = uicontrol('units','norm','pos',[.62 .02 .05 .04], ...
         'style','text');
      h.rank = uicontrol('units','norm','pos',[.42 .06 .16 .04], ...
         'style','text','string',' ');
      h.close = uicontrol('units','norm','pos',[.80 .03 .10 .05], ...
         'string','close','callback','close');
      h.X = X;                      %Copy numeric matrix into structure h.
      set(gcf,'userdata',h)
   end

   % Read or load a new image.
   % Monochrome is a single 2-D array of intensities.
   % Color is a 3-D array of red, green and blue intensities.

   h = get(gcf,'userdata');
   L = get(h.popup,'string');
   name = L{get(h.popup,'val')};

   if isempty(strfind(name,'.mat'))     %If it does not find .mat in the name,
assume we passed a numeric matrix.
      % Read numeric matrix.
      X = h.X;

      %If X is not between 0 and 1, normalize it!
      maximum = max( max( X ) );
      minimum = min( min( X ) );
      X = ( X - minimum ) / ( maximum - minimum );

   else
```

```
      % Load .mat file containing indexed image 'X' and colormap 'map'.
      % Convert to intensities.
      load(name)
      if norm(diff(map'),1) == 0
         % Monochrome image
         T = map(X,1);
         X = reshape(T,size(X));
      else
         % Color image
         T = [map(X,1) map(X,2) map(X,3)];
         X = reshape(T,[size(X) 3]);
      end
   end

   % Resize large images to reduce computation time.

   [m,n,p] = size(X);
   while m >= 768
      i = 1:2:m-1;
      j = 1:2:n-1;
      X = (X(i,j,:)+X(i+1,j,:)+X(i,j+1,:)+X(i+1,j+1,:))/4;
      [m,n,p] = size(X);
   end

   % Slider parameters depend upon size the image.

   mn = min(m,n);
   set(h.slider,'val',1,'min',0,'max',mn,'sliderstep',[1/mn 10/mn])
   set(h.limit,'string',int2str(mn))
   set(h.rank,'string','')

   % Compute the singular value decomposition of the image.

   msg = uicontrol('units','norm','pos',[.25 .56 .50 .10], ...
      'style','text','fontsize',14, ...
      'string',['Computing ' int2str(n*p) '-by-' int2str(m) ' SVD...']);
   drawnow

   X = reshape(X,m,p*n);
   [V,S,U] = svd(X',0);

   % Save the SVD in the figure's user data.

   h.U = U;
   h.S = S;
   h.V = V;
   h.m = m;
   h.n = n;
   h.p = p;
   set(gcf,'userdata',h)
   delete(msg);
end

% Update the plot.

h = get(gcf,'userdata');
U = h.U;
S = h.S;
V = h.V;
m = h.m;
n = h.n;
p = h.p;

% Obtain the rank from the slider.

r = round(get(h.slider,'value'));
set(h.slider,'value',r)
```

```
set(h.rank,'string',['rank = ' num2str(r)]);

% Rank r approximation.

%************************ Modify/Add your code here *********************


k = 1:r;
Y = U(:,k)*S(k,k)*V(:,k)';
X = U * S * V';
error = X - Y;
maximum = max( max( error ) );
minimum = min( min( error ) );
error = ( error - minimum ) / ( maximum - minimum );
error = reshape(error,m,n,p);
imager(error)
drawnow


% ------------------------------------

function imager(X)

% Display the image.

X(X<0) = 0;
X(X>1) = 1;
if ndims(X) == 3
   image(X)
else
   image(255*X)
   colormap(gray(256));
end
axis image
axis off
```



(a) Implementation of mysvd()

```
function [U,S,V] = mysvd(X)
    [Q1 L] = Jacobi(X' * X);
    [Q2 L] = Jacobi(X * X');
    U = Q2;
    V = Q1;
    S = L.^(1/2);
end

function [Q L] = Jacobi(X)
        n = size(X,1);
        Q = id_function(n);

     for j = 1 : (n-1)
           for i = (n-1) : (-1) : (j+1)
                 i1 = i+1;
                 x = X(i,j);
                 y = X(i1,j);
                 T = QRotation(x,y);
                 Q( 1:n, i:i1 ) = Q( 1:n, i:i1 ) * T;
                 X( i:i1, j:n ) = T' * X( i:i1, j:n );
                 X( j:n, i:i1 ) = X( j:n, i:i1 ) * T;
            end
     end

     OffDiagonal = 1 - id_function(n);
     OffDiagonal = (OffDiagonal == 1);
     while( ((X(OffDiagonal)' * X(OffDiagonal))^(1/2)) /
     ((ones_function(1,n) * (X' * X) * ones_function(n,1))^(1/2)) > n * n *
     eps )
        for i = 1:(n-1)
          i1 = i+1;
          x = X(i,i);
          y = X(i1,i);
          T = QRotation(x,y);
          Q( 1:n, i:i1 ) = Q( 1:n, i:i1 ) * T;
          X( i:i1, 1:n ) = T' * X( i:i1, 1:n  );
          X( 1:n, i:i1 ) = X( 1:n, i:i1 ) * T;
        end
      end
      L = X .* id_function(n);
```

```
            end

            function T = QRotation(x,y)
              c = 1;
              s = 0;

              if (abs_function(y) > 0)
                 if (abs_function(y) >= abs_function(x))
                    cotangent = x/y;
                    s = 1/((1 + cotangent^2)^(1/2));
                    c = s * cotangent;
                 else
                    tangent = y/x;
                    c = 1/((1 + tangent^2)^(1/2));
                    s = c * tangent;
                 end
              end
              T = [c -s ; s c];
            end

            function I = id_function(n)
               I (1:n, 1:n) = 0;
               I (1: n+1 : end) = 1;
            end

            function O = ones_function(n,p)
               O(1:n,1:p) = 1;
            end

            function X = abs_function(X)
               X = -1 .* X .* (X<0) + X.* (X>=0);
            end
```

Results of implementation:

```
>> A = hilb(7)

A =

    1.0000    0.5000    0.3333    0.2500    0.2000    0.1667    0.1429
    0.5000    0.3333    0.2500    0.2000    0.1667    0.1429    0.1250
    0.3333    0.2500    0.2000    0.1667    0.1429    0.1250    0.1111
    0.2500    0.2000    0.1667    0.1429    0.1250    0.1111    0.1000
    0.2000    0.1667    0.1429    0.1250    0.1111    0.1000    0.0909
    0.1667    0.1429    0.1250    0.1111    0.1000    0.0909    0.0833
    0.1429    0.1250    0.1111    0.1000    0.0909    0.0833    0.0769

>> [U S V] = mysvd(A)

U =

    0.7332   -0.6232    0.2608   -0.0752    0.0160   -0.0025    0.0002
    0.4364    0.1631   -0.6706    0.5268   -0.2279    0.0618   -0.0098
    0.3198    0.3215   -0.2953   -0.4257    0.6288   -0.3487    0.0952
    0.2549    0.3574    0.0230   -0.4617   -0.2004    0.6447   -0.3713
    0.2128    0.3571    0.2337   -0.1712   -0.4970   -0.1744    0.6825
    0.1831    0.3446    0.3679    0.1827   -0.1849   -0.5436   -0.5911
    0.1609    0.3281    0.4523    0.5098    0.4808    0.3647    0.1944


  S =

   1.6609         0         0         0         0
0         0
        0    0.2719         0         0         0
0         0
```

```
         0              0         0.0213              0              0
   0              0
         0              0              0         0.0010              0
   0              0
         0              0              0              0         0.0000
   0              0
         0              0              0              0              0
   0.0000              0
         0              0              0              0              0
   0         0.0000 + 0.0000i
```

```
   V =

       0.7332   -0.6232    0.2608   -0.0752    0.0160   -0.0025    0.0002
       0.4364    0.1631   -0.6706    0.5268   -0.2279    0.0618   -0.0098
       0.3198    0.3215   -0.2953   -0.4257    0.6288   -0.3487    0.0952
       0.2549    0.3574    0.0230   -0.4617   -0.2004    0.6447   -0.3713
       0.2128    0.3571    0.2337   -0.1712   -0.4970   -0.1744    0.6825
       0.1831    0.3446    0.3679    0.1827   -0.1849   -0.5436   -0.5911
       0.1609    0.3281    0.4523    0.5098    0.4808    0.3647    0.1944
```

```
>> B = magic(7)

B =

    30    39    48     1    10    19    28
    38    47     7     9    18    27    29
    46     6     8    17    26    35    37
     5    14    16    25    34    36    45
    13    15    24    33    42    44     4
    21    23    32    41    43     3    12
    22    31    40    49     2    11    20

>> [U S V] = mysvd(B)

U =

    0.3780   -0.5972   -0.0000    0.5462    0.0000   -0.4496    0.0000
    0.3780   -0.3478   -0.3820   -0.1169    0.3883    0.6377   -0.1385
    0.3780    0.1237   -0.5200   -0.5326   -0.0452   -0.4937    0.2049
    0.3780    0.3602   -0.2274    0.3002   -0.6425    0.2039   -0.3621
    0.3780    0.5227    0.1120    0.3764    0.3795    0.0808    0.5284
    0.3780    0.1952    0.4783   -0.2228    0.3389   -0.2122   -0.6171
    0.3780   -0.2569    0.5391   -0.3505   -0.4189    0.2331    0.3844


S =

  175.0000         0         0         0         0         0         0
         0   57.4356         0         0         0         0         0
         0         0   57.4356         0         0         0         0
         0         0         0   31.5526         0         0         0
         0         0         0         0   31.5526         0         0
         0         0         0         0         0   24.6086         0
         0         0         0         0         0         0   24.6086


V =

    0.3780   -0.4269   -0.0000    0.7054    0.0000    0.4210   -0.0000
    0.3780   -0.3260    0.4065   -0.0049    0.3638   -0.6617   -0.1240
    0.3780    0.1280    0.5622   -0.4481    0.1172    0.5413    0.1314
    0.3780    0.6275    0.1841    0.2907   -0.4759   -0.1902   -0.2870
    0.3780    0.4114   -0.4039    0.1002    0.4522   -0.0901    0.5496
    0.3780   -0.0547   -0.5182   -0.3215    0.1704    0.1438   -0.6577
    0.3780   -0.3593   -0.2306   -0.3220   -0.6277   -0.1642    0.3877
```

```
>> C = pascal(7,1)

C =

     1     0     0     0     0     0     0
     1    -1     0     0     0     0     0
     1    -2     1     0     0     0     0
     1    -3     3    -1     0     0     0
     1    -4     6    -4     1     0     0
     1    -5    10   -10     5    -1     0
     1    -6    15   -20    15    -6     1

>> [U S V] = mysvd(C)

U =

    0.0013   -0.0313    0.2674   -0.7385    0.5778   -0.2144    0.0472
    0.0084   -0.1163    0.4827   -0.3693   -0.4652    0.5833   -0.2457
    0.0309   -0.2684    0.5473    0.1231   -0.3255   -0.4597    0.5431
    0.0876   -0.4598    0.3342    0.3693    0.2747   -0.1901   -0.6490
    0.2096   -0.5871   -0.1263    0.1231    0.3404    0.5196    0.4411
    0.4452   -0.4265   -0.4714   -0.3693   -0.3777   -0.3043   -0.1613
    0.8655    0.4189    0.2146    0.1231    0.0993    0.0611    0.0248


S =

   34.9580         0         0         0         0         0         0
         0    6.8579         0         0         0         0         0
         0         0    2.1607         0         0         0         0
         0         0         0    1.0000         0         0         0
         0         0         0         0    0.4628         0         0
         0         0         0         0         0    0.1458         0
         0         0         0         0         0         0    0.0286


V =

   -0.0472    0.2144    0.5778   -0.7385    0.2674    0.0313    0.0013
    0.2457   -0.5833   -0.4652   -0.3693    0.4827    0.1163    0.0084
   -0.5431    0.4597   -0.3255    0.1231    0.5473    0.2684    0.0309
    0.6490    0.1901    0.2747    0.3693    0.3342    0.4598    0.0876
   -0.4411   -0.5196    0.3404    0.1231   -0.1263    0.5871    0.2096
    0.1613    0.3043   -0.3777   -0.3693   -0.4714    0.4265    0.4452
   -0.0248   -0.0611    0.0993    0.1231    0.2146   -0.4189    0.8655

>> D = vander(1:7)

D =

           1           1           1           1           1           1
1
          64          32          16           8           4           2
1
         729         243          81          27           9           3
1
        4096        1024         256          64          16           4
1
       15625        3125         625         125          25           5
1
       46656        7776        1296         216          36           6
1
      117649       16807        2401         343          49           7
1
```

```
>> [U S V] = mysvd(D)


U =

    0.0000   -0.0008    0.0269   -0.3393    0.8313   -0.4324    0.0779
    0.0005   -0.0180    0.1943   -0.6473    0.1015    0.6583   -0.3150
    0.0059   -0.1021    0.4913   -0.4455   -0.3811   -0.2467    0.5861
    0.0326   -0.3078    0.6230    0.1669   -0.0586   -0.3160   -0.6205
    0.1234   -0.5934    0.2282    0.3794    0.3306    0.4228    0.3854
    0.3667   -0.6485   -0.5084   -0.3019   -0.1977   -0.1974   -0.1315
    0.9215    0.3491    0.1465    0.0667    0.0388    0.0343    0.0191

S =

   1.0e+05 *

    1.2899         0         0         0         0         0         0
         0    0.0147         0         0         0         0         0
         0         0    0.0005         0         0         0         0
         0         0         0    0.0000         0         0         0
         0         0         0         0    0.0000         0         0
         0         0         0         0         0    0.0000         0
         0         0         0         0         0         0    0.0000

V =

    0.9891   -0.1445    0.0263   -0.0062    0.0024   -0.0011    0.0002
    0.1454    0.9347   -0.3063    0.0950   -0.0424    0.0223   -0.0053
    0.0215    0.3142    0.7855   -0.4386    0.2490   -0.1639    0.0502
    0.0032    0.0799    0.4913    0.4415   -0.4884    0.5114   -0.2391
    0.0005    0.0184    0.2029    0.5859   -0.0854   -0.5089    0.5907
    0.0001    0.0041    0.0726    0.4297    0.4246   -0.3653   -0.7045
    0.0000    0.0009    0.0249    0.2751    0.7142    0.5645    0.3081

>> E = rosser()

E =

   611   196  -192   407    -8   -52   -49    29
   196   899   113  -192   -71   -43    -8   -44
  -192   113   899   196    61    49     8    52
   407  -192   196   611     8    44    59   -23
    -8   -71    61     8   411  -599   208   208
   -52   -43    49    44  -599   411   208   208
   -49    -8     8    59   208   208    99  -911
    29   -44    52   -23   208   208  -911    99

>> [U S V] = mysvd(E)

U =

    0.6325    0.0000   -0.2236   -0.0623   -0.3847   -0.0000    0.6294    0.0447
    0.3162   -0.0000    0.4472    0.0312   -0.2495   -0.7278   -0.3147    0.0894
    0.3162   -0.0000    0.4472   -0.0312    0.7694    0.0000    0.3147   -0.0894
    0.6325    0.0000   -0.2236    0.0623    0.1248    0.3639   -0.6294   -0.0447
   -0.0000    0.3162   -0.4472   -0.3147    0.3327   -0.3119   -0.0312    0.6261
    0.0000    0.3162    0.4472    0.3147   -0.1872    0.4159    0.0312    0.6261
   -0.0000   -0.6325   -0.2236    0.6294    0.1664   -0.1560    0.0623    0.3130
    0.0000   -0.6325    0.2236   -0.6294   -0.0936    0.2080   -0.0623    0.3130


S =

   1.0e+03 *
```

```
   1.0200         0         0         0         0
0         0         0
         0    1.0200         0         0         0
0         0         0
         0         0    1.0200         0         0
0         0         0
         0         0         0    1.0199         0
0         0         0
         0         0         0         0    1.0000
0         0         0
         0         0         0         0         0
1.0000         0         0
         0         0         0         0         0
0    0.0001         0
       0         0         0         0         0         0    0.0000 +0.0000i
```

V =

```
    0.6325    0.0000   -0.2236   -0.0623   -0.3847   -0.0000    0.6294    0.0447
    0.3162   -0.0000    0.4472    0.0312   -0.2495   -0.7278   -0.3147    0.0894
    0.3162   -0.0000    0.4472   -0.0312    0.7694    0.0000    0.3147   -0.0894
    0.6325    0.0000   -0.2236    0.0623    0.1248    0.3639   -0.6294   -0.0447
   -0.0000    0.3162   -0.4472   -0.3147    0.3327   -0.3119   -0.0312    0.6261
    0.0000    0.3162    0.4472    0.3147   -0.1872    0.4159    0.0312    0.6261
   -0.0000   -0.6325   -0.2236    0.6294    0.1664   -0.1560    0.0623    0.3130
    0.0000   -0.6325    0.2236   -0.6294   -0.0936    0.2080   -0.0623    0.3130
```

(b)

```
function checkEquivalence(A)
        [n,p] = size(A);
        [U S V] = mysvd(A);
        [U1 S1 V1] = svd(A);
        X1 = [cos(0) sin(0)];
        X2 = [cos(pi) sin(pi)];
        Y = [1; i];
        flag1 = 0; flag2 = 0;

        %Checking for U
        for I = 1: size(U,2)
            if( ~((uint8((X1*Y) * U(:,I)' * U1(:,I)) ==  1) || (uint8((X2*Y) *
U(:,I)' * U1(:,I)) ==1)))
                flag1 = 1;
            end
        end

         %Checking for V
         for I = 1:size(V,2)
            if( ~((uint8((X1*Y) * V(:,I)' * V1(:,I)) ==  1) || (uint8((X2*Y) *
V(:,I)' * V1(:,I)) ==1)))
                flag2=1;
            end
         end

         if(flag1==1 || flag2==1)
             disp('The two SVD results are not unit equivalent');
         else
             disp('The two SVD results are unit equivalent');
         end
end
```

```
>> checkEquivalence(A)
The two SVD results are unit equivalent

>> checkEquivalence(B)
```

```
The two SVD results are not unit equivalent
-The difference is that column number 4 and 5 are flipped in the two results,
because the corresponding singular values are equal.

>> checkEquivalence(C)
The two SVD results are unit equivalent

>> checkEquivalence(D)
The two SVD results are unit equivalent

>> checkEquivalence(E)
The two SVD results are not unit equivalent
-The SVD that has been implemented appears to have a few columns different from the
built in SVD.


(c)function c = condition_number(A)
    [U S V] = mysvd(A);
    [n,p] = size(A);
    c = S(1,1)/S(n,p)

>> condition_number(A);
c =

   2.8764e-08 - 4.6975e+08i

>> condition_number(B);
c =

    7.1113

>> condition_number(C);
c =

   1.2221e+03

>> condition_number(D);
c =

   2.4457e+07

>> condition_number(E);
c =

   6.0093e-09 - 9.8139e+07i
```