CS170A -- HW#2: Eigenfaces -- Octave

Your name: Shirley Xuemin He

Your UID: 204468663

Please upload only this notebook to CCLE by the deadline.

Policy for late submission of solutions: We will use Paul Eggert's Late Policy: N days late ⇔ 2^N points deducted} The number of days late is $N = 0$ for the first 24 hrs, $N = 1$ for the next 24 hrs, etc., and if you submit an assignment H hours late, $2^{\lfloor H/24 \rfloor}$ points are deducted.

Problem 1: Eigenfaces

Chapter 11 of the Course Reader is on Eigenfaces. For this assignment we have included the face files for this chapter in the directory `old_faces`. It includes some Matlab scripts and a database of 177 face images, each a grayscale `.bmp` bitmap file of size 64×64 pixels. The face images have been pre-processed so that the background and hair are removed and the faces have similar lighting conditions.

The Course Reader explains how to reshape each face image into a $1 \times 64^2 \;=\; 1 \times 4096$ row vector, and collect them into a matrix. The principal components of the matrix then define the main dimensions of variance in the faces. The program `more_efficient_eigenfaces.m` shows how to do this. These principal components are called *eigenfaces*.

This Assignment uses a new Face Dataset -- with Normal and Smiling Faces

The goal of this problem is to apply the same ideas to a new set of 200 faces in the directory `new_faces`. The subdirectory `new_faces/faces` has 200 faces that have been normalized, cropped, and equalized. The subdirectory `new_faces/smiling_faces` has 200 images of the same people, but they are smiling. Each of these images is a grayscale `.jpg` file with size 193×162.

1a: The Average Face

Modify the program `more_efficient_eigenfaces.m` (available in this directory) to use the `new_faces` images instead of the `old_faces` images. Also, modify it to use the Matlab function `imresize` to downsample each of the new faces by a factor of 3, so it is 64×54 (instead of 193×162). Then: *pad* both sides of the image with `zeros(64,5)` so the result is a 64×64 image.

Then: create a function that takes as input a string array of filenames of face images, an integer k, and an integer sample size s --- and yields the average face and the first k singular values and eigenfaces as output values for a sample of size s.

Apply your function to both the `new_faces/faces` and the `new_faces/smiling_faces` directories, and plot the absolute value of the difference between your average face and (your downsampled version of) the average face provided in the directory.

(The `imagesc` function can display images with automatic rescaling of numeric values.)

In [5]:

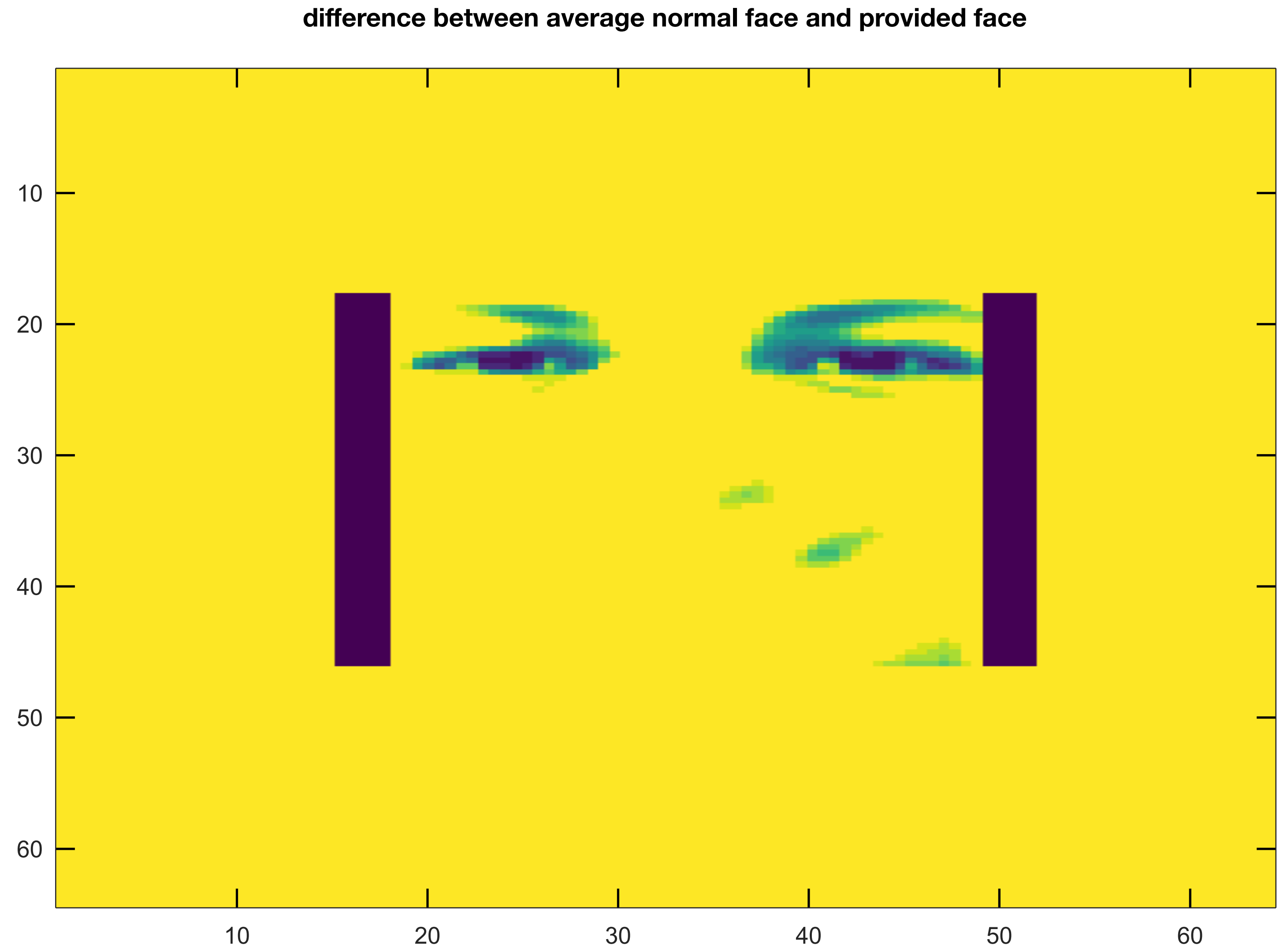
```
pkg load image

function [fbar, F] = diff_with_provided(directory,samplesize,k)

filenames=dir(fullfile(directory,"*.jpg"));
filenames = filenames(1:200);

row = 64;
col = 64;

F = zeros(samplesize,row*col); % the array of sample images (stored as vectors)

numfiles = size(filenames,1);
rp = randperm(numfiles);        %  random permutation of the list of image filena
mes

sample = rp(1:samplesize);      %  use the first <samplesize> images as our sampl
e

image_vector = @(Bitmap) double(reshape(Bitmap,1,row*col));

vector_image = @(Vec) reshape( uint8( min(max(Vec,0),255) ), row, col);

vector_render = @(Vec) imagesc(vector_image(Vec));

for i = 1:samplesize
   Image_File = filenames(sample(i)).name;
   Face_Matrix = imread(fullfile(directory,Image_File));
   Face_Matrix = padarray(imresize(Face_Matrix, [64 54]),[0,5]);
   F(i,:) = image_vector(Face_Matrix);  % the i-th row of F is the i-th image
end
```

```
fbar = sum(F,1)/samplesize;   % average of all rows in F


%%% Jupyter has bugs using imshow()
% figure
% imshow(vector_image(fbar))
% if (strcmp(directory,"new_faces/faces") == 1)
%     title("average normal face");
% elseif (strcmp(directory,"new_faces/smiling_faces") == 1)
%     title("average smiling face");
% endif

if (strcmp(directory,"new_faces/faces") == 1)
    provided_average = padarray(imresize(imread(fullfile(directory,"averagefacei
mage_seta.jpg")), [64 54]), [0,5]);
elseif (strcmp(directory,"new_faces/smiling_faces") == 1)
    provided_average = padarray(imresize(imread(fullfile(directory,"averagefacei
mage_setb.jpg")), [64 54]), [0,5]);
endif

provided_average = image_vector(provided_average);

diff = abs(provided_average - fbar);

figure
vector_render(diff)

if (strcmp(directory,"new_faces/faces") == 1)
    title("difference between average normal face and provided face");
elseif (strcmp(directory,"new_faces/smiling_faces") == 1)
    title("difference between average smiling face and provided face");
endif

endfunction
```

```
warning: function /Users/shirleyhe/octave/image-2.6.1/im2double.m sh
adows a core library function
warning: called from
    load_packages_and_dependencies at line 48 column 5
    load_packages at line 47 column 3
    pkg at line 409 column 7
```

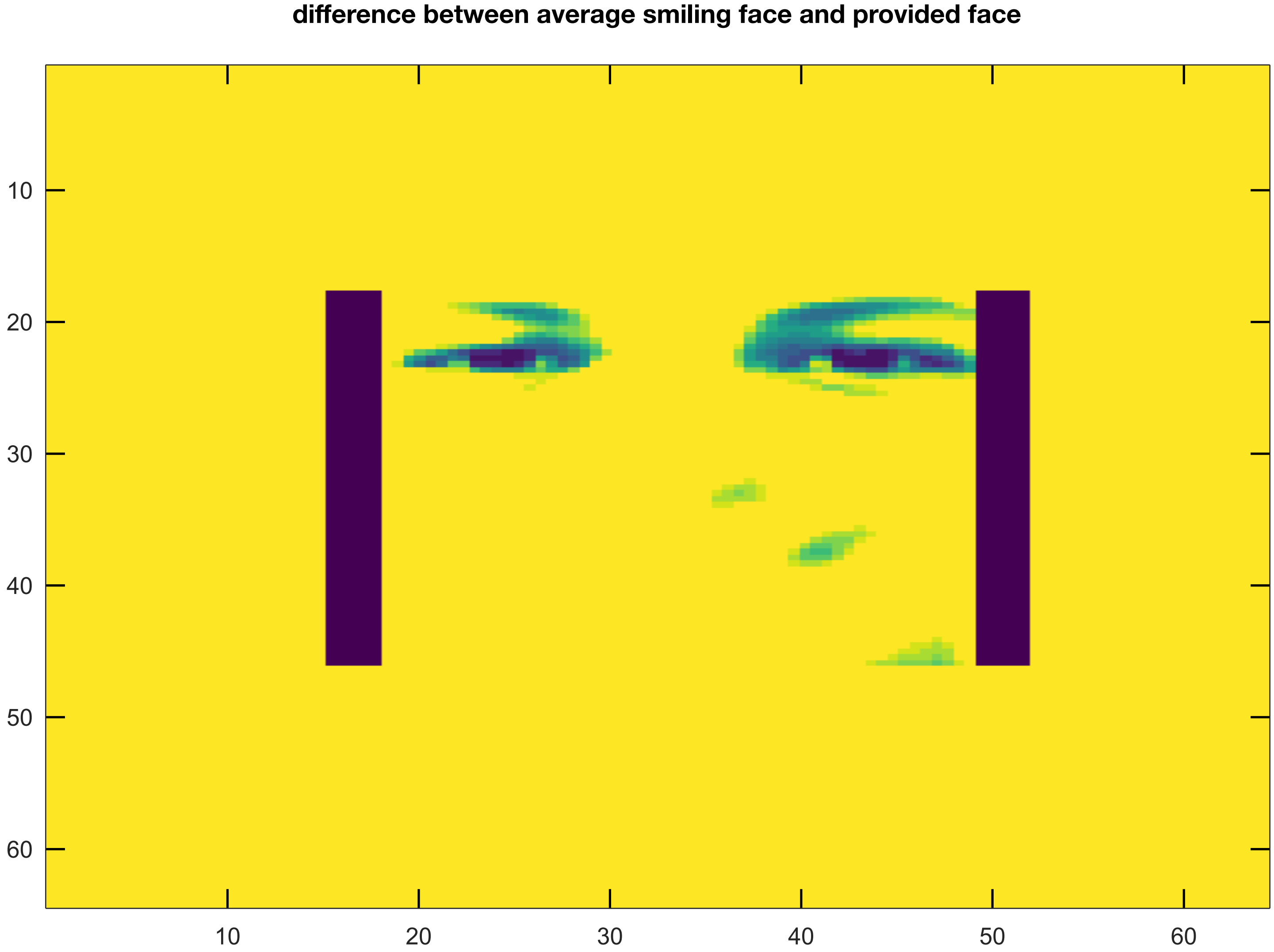
In [6]:

```
%  image showing the difference between your average normal face and the one pro
vided
[meanNormal, normalFaces] = diff_with_provided("new_faces/faces",150,60);
```



In [7]:

```
%  image showing the difference between your average smiling face and the one pr
ovided
[meanSmiling, smilingFaces] = diff_with_provided("new_faces/smiling_faces",150,6
0);
```



1b: Smiling makes a Difference

If your mean normal face is $\bar{\mathbf{f}}_0$, and your mean smiling face is $\bar{\mathbf{f}}_1$, render (using `imagesc`) the difference $\bar{\mathbf{f}}_0 - \bar{\mathbf{f}}_1$ (the average difference between normal faces and smiling faces).

In [13]:

```
%  image showing the difference between the average normal face and average smil
ing face

imshow('diff_normal_smiling.jpg')
```

1c: Scree Plots and k-Approximation

Using your downsampled images, perform PCA on each set of faces (normal and smiling).

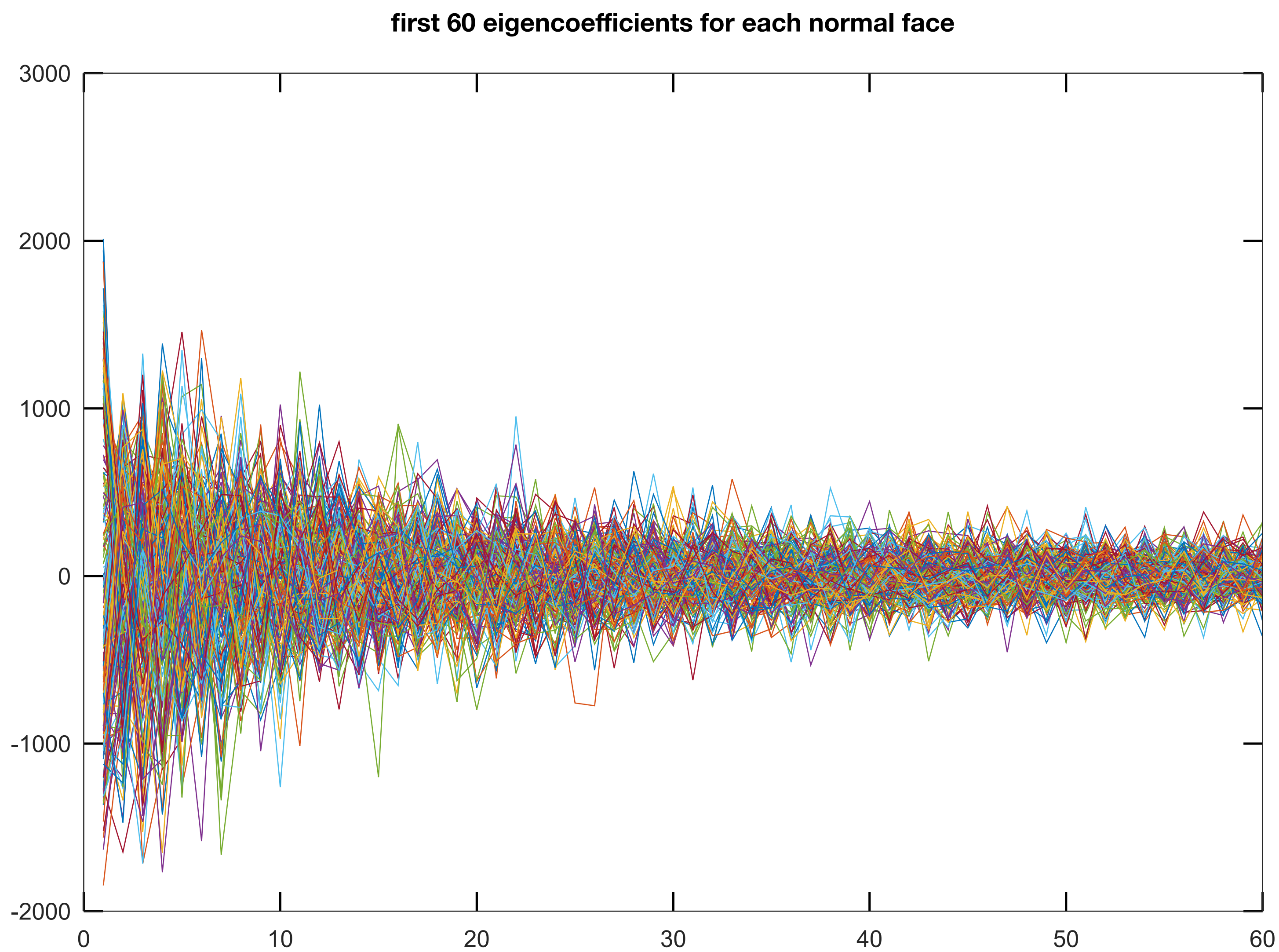
For each image (normal or smiling), construct its $64^2 \times 1$ vector $\mathbf{f}$. Then, subtract the average face ($\bar{\mathbf{f}}_0$ or $\bar{\mathbf{f}}_1$) and project the remainder on the first $k = 60$ eigenfaces. For example, with a smiling face, the projection of $\mathbf{f}$ on the j-th smiling eigenface $\mathbf{e}_j$ is

$$c_j = (\mathbf{f} - \bar{\mathbf{f}}_1)' \mathbf{e}_j \qquad (j = 1, \ldots, k).$$

For each set of faces (normal or smiling), make one large scree plot for the set, showing all sequences of the first k coefficients for each image overplotted (e.g. with `hold on`).

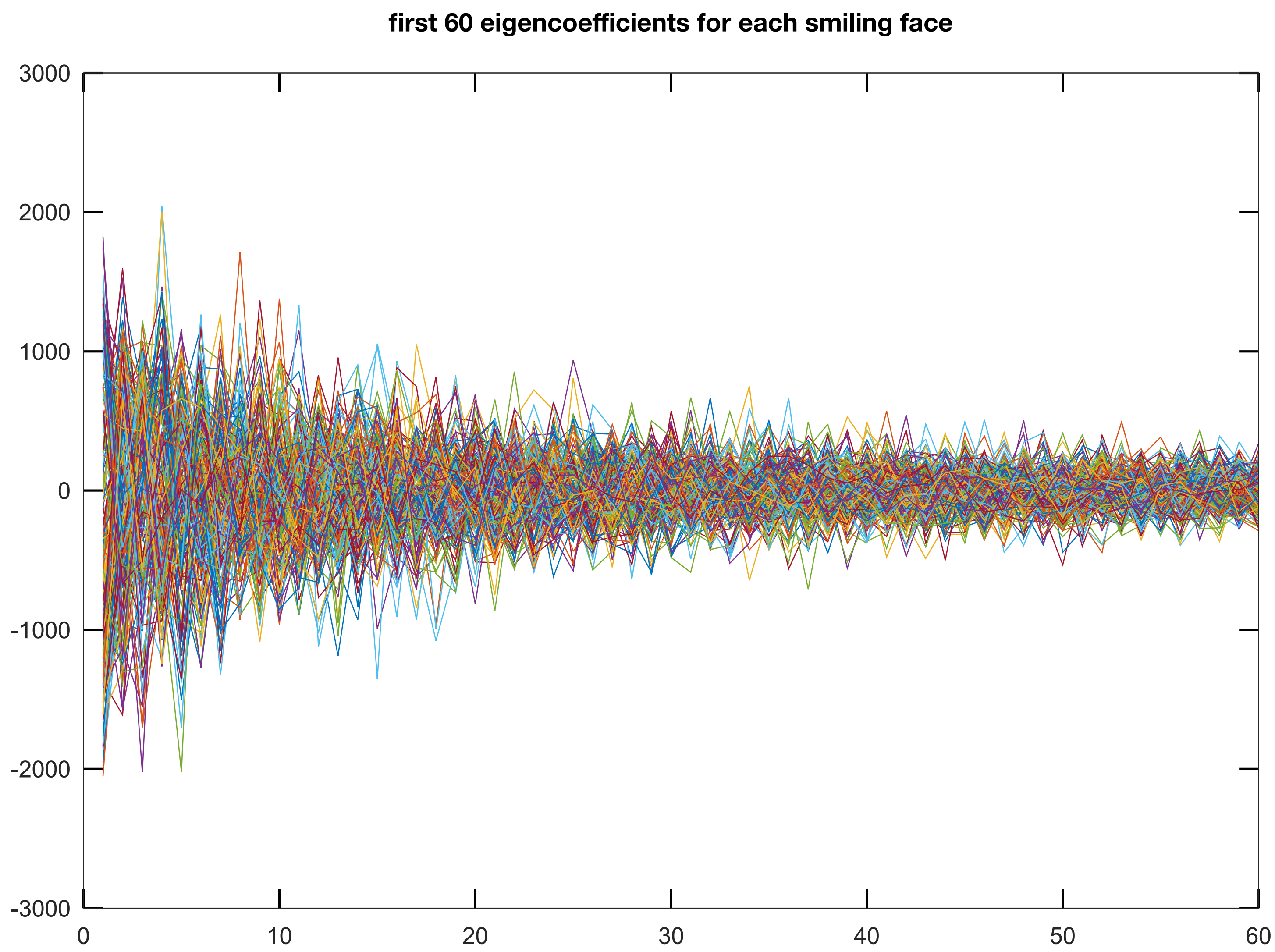
In [15]:

```
%  (overlaid) scree plots for normal faces

for i = 1:150
    normalFaces(i,:) = normalFaces(i,:) - meanNormal;
end

% UNormal are eigenfaces
[UNormal, singular_values_normal] = more_efficient_pca( normalFaces, 60);

allnormalfaces = dir(fullfile("new_faces/faces","*.jpg"));
allnormalfaces = allnormalfaces(1:200);

for j = 1:numel(allnormalfaces)
    f = imread(fullfile("new_faces/faces",allnormalfaces(j).name));
    f = padarray(imresize(f, [64 54]),[0,5]);
    f = double(reshape(f,1,64*64));
    t = f - meanNormal;
    c = t * UNormal;
    plot(c); hold on;
end

title("first 60 eigencoefficients for each normal face");
hold off;
```



In [16]:

```
%  (overlaid) scree plots for smiling faces

for i = 1:150
    smilingFaces(i,:) = smilingFaces(i,:) - meanSmiling;
end

% UNormal are eigenfaces
[USmiling, singular_values_smiling] = more_efficient_pca( smilingFaces, 60);

allsmilingfaces = dir(fullfile("new_faces/smiling_faces","*.jpg"));
allsmilingfaces = allsmilingfaces(1:200);

for j = 1:numel(allsmilingfaces)
    f = imread(fullfile("new_faces/smiling_faces",allsmilingfaces(j).name));
    f = padarray(imresize(f, [64 54]),[0,5]);
    f = double(reshape(f,1,64*64));
    t = f - meanSmiling;
    c = t * USmiling;
    plot(c); hold on;
end

title("first 60 eigencoefficients for each smiling face");
hold off;
```



1d: Unusualness of a Face

Let us say the *unusualness* of a face is the L_2 norm of its eigenface-coefficient vector (using the first $k = 60$ eigenfaces).

Determine, for each set (normal or smiling), the most unusual face.

In [27]:

```
%  the most unusual normal face

unusual_normal = allnormalfaces(1).name;
max_normal = 0;

for j = 1:numel(allnormalfaces)
    f = imread(fullfile("new_faces/faces",allnormalfaces(j).name));
    f = padarray(imresize(f, [64 54]),[0,5]);
    f = double(reshape(f,1,64*64));
    t = f - meanNormal;
    c = t * UNormal;
    if (norm(c) > max_normal)
        max_normal = norm(c);
        unusual_normal = allnormalfaces(j).name;
    endif
end

%%% Jupyter has bugs using imshow()
% f = imread(fullfile("new_faces/faces",unusual_normal));
% imshow(f)

%%% This image is supposed to be black and white
imshow('unusual_normal.jpg')
```

In [28]:

```
%  the most unusual smiling face

unusual_smiling = allsmilingfaces(1).name;
max_smiling = 0;

for j = 1:numel(allsmilingfaces)
    f = imread(fullfile("new_faces/smiling_faces",allsmilingfaces(j).name));
    f = padarray(imresize(f, [64 54]),[0,5]);
    f = double(reshape(f,1,64*64));
    t = f - meanSmiling;
    c = t * USmiling;
    if (norm(c) > max_smiling)
        max_smiling = norm(c);
        unusual_smiling = allsmilingfaces(j).name;
    endif
end




%%% Jupyter has bugs using imshow()
% f = imread(fullfile("new_faces/smiling_faces",unusual_smiling));
% imshow(f)

%%% This image is supposed to be black and white
imshow('unusual_smiling.pdf')
```

Problem 2: Face Classifiers

Develop two different face classifiers using the eigenfaces you've computed; each should be a function that, given a face image $\mathbf{f}$ as input, yields the output value 1 if $\mathbf{f}$ is smiling, and 0 otherwise. (NOTE: or vice-versa; we just need the function to be a classifier)

Specifically, implement the following 3 classifiers that take an input image $\mathbf{f}$:

- {Classifier X}: yield 1 if the normal face unusualness of $\mathbf{f}$ is greater than smiling face unusualness of $\mathbf{f}$, else 0.

- {Classifier Y}: yield 1 if $||\mathbf{f} - \bar{\mathbf{f}}_0||^2 \;\geq\; ||\mathbf{f} - \bar{\mathbf{f}}_1||^2$, else 0.

2a: Unusual Face Classification

Using each of these classifiers, determine the classification it yields for the two most unusual images you found in the previous question.

In [29]:

```
function [ans] = classifierX(f,meanNormal,meanSmiling,UNormal,USmiling)
    normal_u = norm((f - meanNormal) * UNormal);
    smiling_u = norm((f - meanSmiling) * USmiling);
    if (normal_u > smiling_u)
        ans = 1;
    else
        ans = 0;
    endif
endfunction

function [ans] = classifierY(f,meanNormal,meanSmiling,UNormal,USmiling)
    normal_norm = norm(f - meanNormal)*norm(f - meanNormal);
    smiling_norm = norm(f - meanSmiling)*norm(f - meanSmiling);
    if (normal_norm >= smiling_norm)
        ans = 1;
    else
        ans = 0;
    endif
endfunction
```

In [30]:

```
%  X, Y, Z classifications of the most unusual normal face

most_unusual_normal = imread(fullfile("new_faces/faces",unusual_normal));
most_unusual_normal = double(reshape(padarray(imresize(f, [64 54]),[0,5]),1,64*6
4));

classifierX(most_unusual_normal,meanNormal,meanSmiling,UNormal,USmiling)
classifierY(most_unusual_normal,meanNormal,meanSmiling,UNormal,USmiling)
```

```
ans =  1
ans =  1
```

In [31]:

```
%  X, Y, Z classifications of the most unusual smiling face

most_unusual_smiling = imread(fullfile("new_faces/smiling_faces",unusual_smiling
));
most_unusual_smiling = double(reshape(padarray(imresize(f, [64 54]),[0,5]),1,64*
64));

classifierX(most_unusual_smiling,meanNormal,meanSmiling,UNormal,USmiling)
classifierY(most_unusual_smiling,meanNormal,meanSmiling,UNormal,USmiling)
```

```
ans =  1
ans =  1
```

2b: Splitting into Training and Test sets

Write a function `[Sublist1 Sublist2] = randsplit(List)` that takes an array `List` of length `n` and splits it randomly into two sublists of size `floor(n/2)` and `ceil(n/2)`. (Hint: `randperm`)

Use `randsplit` to split each of the 200-face sets into a training subset and testing subset of equal size.

For both sets of faces (100 normal faces and 100 smiling faces), compute the average normal and smiling faces $\bar{\mathbf{f}}_0$ and $\bar{\mathbf{f}}_1$ for the training set.

In [32]:

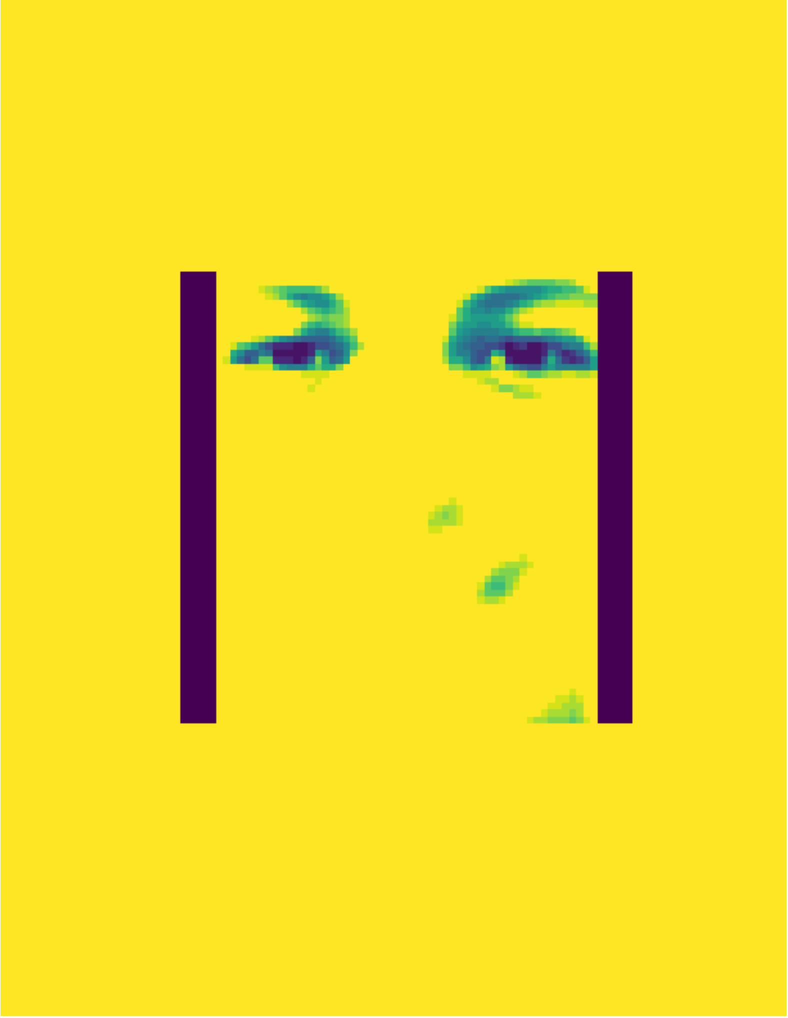
```
function [Sublist1 Sublist2] = randsplit(List)
    n = numel(List);
    rp = randperm(size(List,1));
    for k = 1:floor(n/2)
        Sublist1(k) = List(rp(k));
        Sublist2(k) = List(rp(k+floor(n/2)));
    end
endfunction
```

In [34]:

```
% The average normal face (for the training set)

[train_normal test_normal] = randsplit(allnormalfaces);

for i = 1:100
   Image_File = train_normal(i).name;
   Face_Matrix = imread(fullfile("new_faces/faces",Image_File));
   Face_Matrix = padarray(imresize(Face_Matrix, [64 54]),[0,5]);

   F(i,:) = double(reshape(Face_Matrix,1,64*64));
end

f0bar = sum(F,1)/100;

%%% Jupyter has bugs using imshow()
% imshow(reshape( uint8( min(max(f0bar,0),255) ), 64, 64))

%%% This image is supposed to be black and white
imshow('f0bar.pdf')
```

In [35]:

```
% The average smiling face (for the training set)

[train_smiling test_smiling] = randsplit(allsmilingfaces);

for i = 1:100
   Image_File = train_smiling(i).name;
   Face_Matrix = imread(fullfile("new_faces/smiling_faces",Image_File));
   Face_Matrix = padarray(imresize(Face_Matrix, [64 54]),[0,5]);

   F(i,:) = double(reshape(Face_Matrix,1,64*64));
end

f1bar = sum(F,1)/100;

%%% Jupyter has bugs
% imshow(reshape( uint8( min(max(f1bar,0),255) ), 64, 64));

%%% This image is supposed to be black and white
imshow('f1bar.pdf')
```



2c: Classifier Error Rate

For each of the Classifiers (X, Y), using the average faces you just computed:

- classify each of the 200 faces $\mathbf{f}$ in the testing set, and count classification errors.
- compute the *error rate* (percentage of errors in test face classifications) for the Classifier.

Then rank the classifiers by their error rate.

For normal faces (using the test set):

In [45]:

```
% X, Y error rates

errX = 0;
errY = 0;

for k = 1:100
    Image_File = test_normal(k).name;
    Face_Matrix = imread(fullfile("new_faces/faces",Image_File));
    Face_Matrix = double(reshape(padarray(imresize(Face_Matrix, [64 54]),[0,5]),
1,64*64));
    ansX = classifierX(Face_Matrix,meanNormal,meanSmiling,UNormal,USmiling);
    if (ansX == 1)
        errX = errX + 1;
    endif
    ansY = classifierY(Face_Matrix,meanNormal,meanSmiling,UNormal,USmiling);
    if (ansY == 1)
        errY = errY + 1;
    endif
end

error_rate_X = double(errX/100)
error_rate_Y = double(errY/100)
```

```
error_rate_X =  0.080000
error_rate_Y =  0.040000
```

In [46]:

```
%  which of the classifiers has lowest error rate for normal faces in the test s
et?

if (error_rate_X < error_rate_Y)
    printf("classifier X");
else
    printf("classifier Y");
end
```

```
classifier Y
```

For smiling faces (using the test set):

In [47]:

```
% X, Y error rates

errX = 0;
errY = 0;

for k = 1:100
    Image_File = test_smiling(k).name;
    Face_Matrix = imread(fullfile("new_faces/smiling_faces",Image_File));
    Face_Matrix = double(reshape(padarray(imresize(Face_Matrix, [64 54]),[0,5]),
1,64*64));
    ansX = classifierX(Face_Matrix,meanNormal,meanSmiling,UNormal,USmiling);
    if (ansX == 0)
        errX = errX + 1;
    endif
    ansY = classifierY(Face_Matrix,meanNormal,meanSmiling,UNormal,USmiling);
    if (ansY == 0)
        errY = errY + 1;
    endif
end

error_rate_X = double(errX/100)
error_rate_Y = double(errY/100)
```

```
error_rate_X =  0.25000
error_rate_Y =  0.080000
```

In [48]:

```
%  which of the classifiers has lowest error rate for smiling faces in the test set?

if (error_rate_X < error_rate_Y)
    printf("classifier X");
else
    printf("classifier Y");
end
```

```
classifier Y
```

Problem 3: Face Compression

In the previous problem you computed the first 60 Eigenface coefficients, and used these to find the most unusual face.

For each 64×64 image X from your (downsampled) smiling faces, compute the following sequences:

- (descendingly sorted absolute values of) the first 60 Eigenface coefficients for X
- (descendingly sorted absolute values of) the first 60 coefficient values from the two-sided FFT of X (in Matlab: `fft2(X)`)
- (descendingly sorted absolute values of) the first 60 coefficient values from the two-sided DCT of X (in Matlab: `dct2(X)`)
- the first 60 singular values from the SVD of X.

We get an *image compression* scheme if we keep only the first $k \leq 60$ coefficients, and discard the rest.

Define

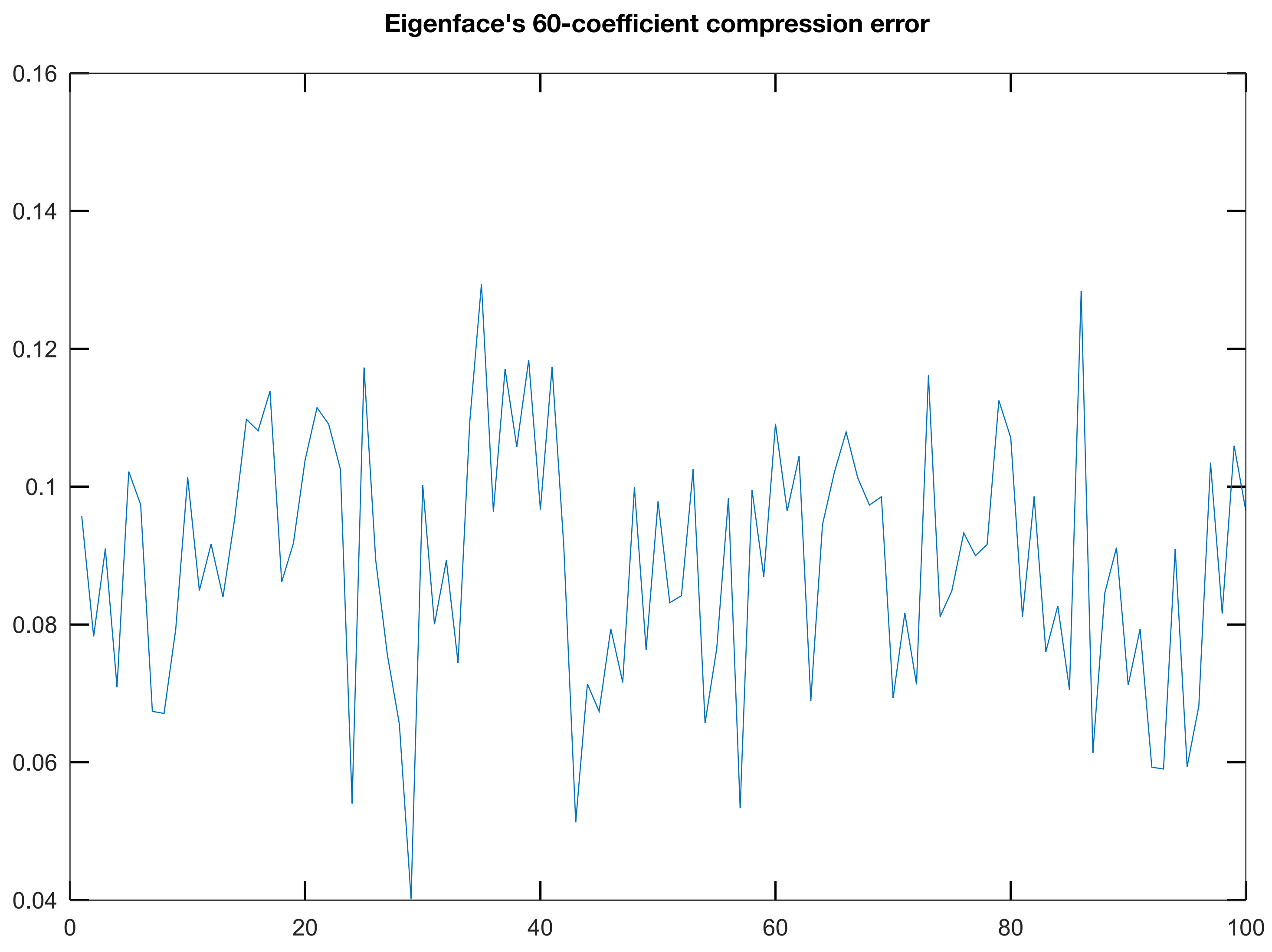
$$k\text{-coefficient compression error} \quad = \quad \frac{\text{(the sum of absolute values of all coefficients after the first } k\text{)}}{\text{(the sum of absolute values of all coefficients)}}.$$

Compute the k-coefficient compression error for each of the 4 transforms, $1 \leq k \leq 60$, <u>for the smiling test set</u>.

Rank the 4 transforms above by their average compression error (for $k \leq 60$).

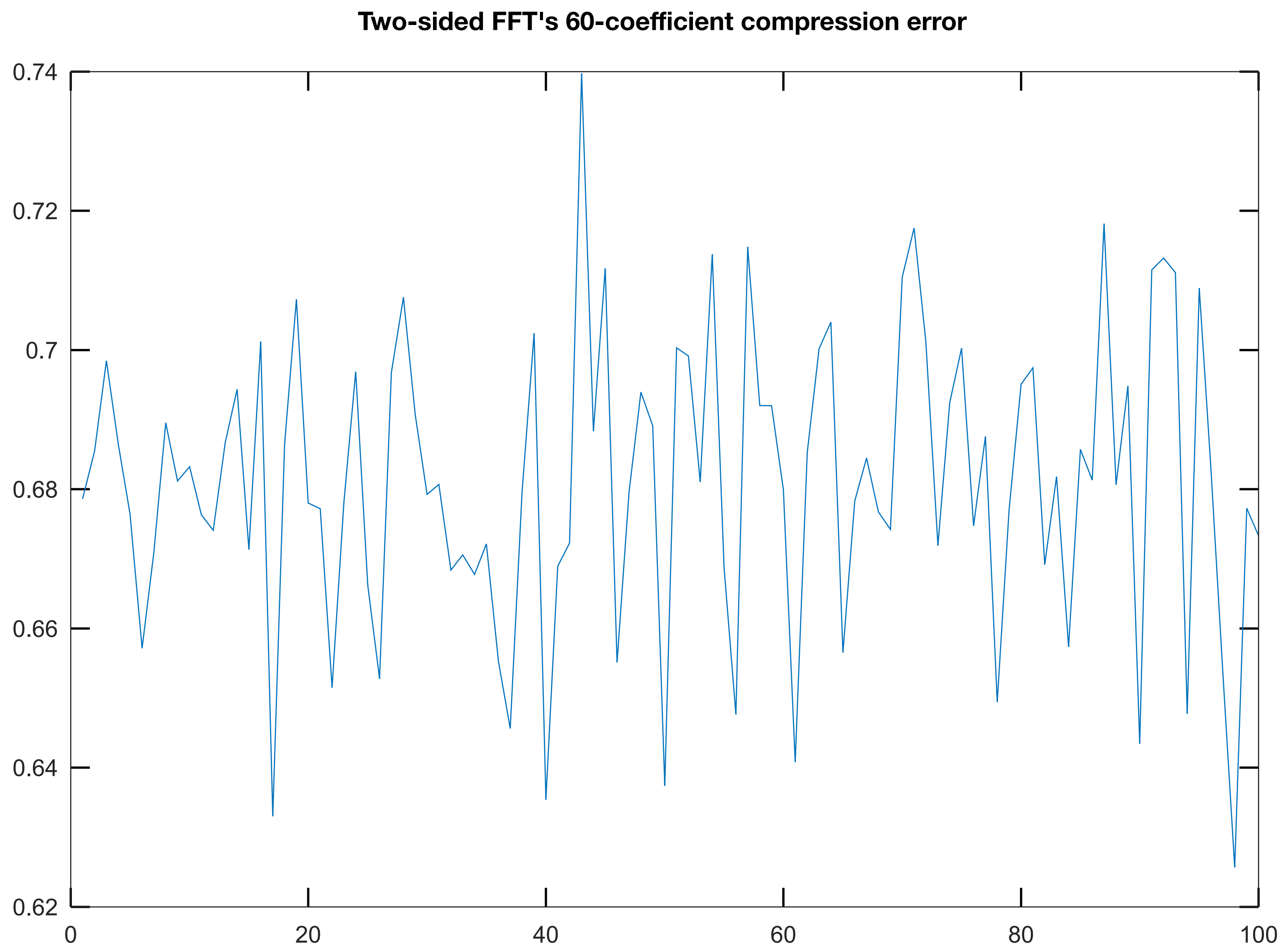
In [68]:

```
%  plot of the Eigenface's k-coefficient compression error   (for k <= 60)

for i = 1:100
    Image_File = test_smiling(i).name;
    Face_Matrix = imread(fullfile("new_faces/smiling_faces",Image_File));
    Face_Matrix = padarray(imresize(Face_Matrix, [64 54]),[0,5]);
    test_smiling_vectors(i,:) = double(reshape(Face_Matrix,1,64*64));
    test_smiling_vectors(i,:) = test_smiling_vectors(i,:) - meanSmiling;
end

[USmiling, singular_values_smiling] = more_efficient_pca( test_smiling_vectors,
100);

for i= 1:100
    image_f = imread(fullfile("new_faces/smiling_faces",test_smiling(i).name));
    image_f = padarray(imresize(image_f, [64 54]),[0,5]);
    vector_f = double(reshape(image_f,1,64*64));
    t = vector_f - meanSmiling;
    c = t * USmiling;
    c = sort(abs(c), 'descend');    % sorted first 60 eigenface coefficients
    compression_error_eface(i) = sum(c(61:end))/sum(c);
end

plot(1:100, compression_error_eface)
title("Eigenface's 60-coefficient compression error")
```



In [69]:

```
%  plot of the two-sided FFT's k-coefficient compression error  (for k <= 60)

for i = 1:100
    X = imread(fullfile("new_faces/smiling_faces",test_smiling(i).name));
    X = padarray(imresize(X, [64 54]),[0,5]);
    fftX = fft2(X);
    FourierCoefficients = sort(abs(fftX(:)), 'descend');
    compression_error_fft(i) = sum(FourierCoefficients(61:end))/sum(FourierCoeff
icients);
end

plot(1:100, compression_error_fft)
title("Two-sided FFT's 60-coefficient compression error")

%%%   You might do something like this:
%   TwoSidedFFTofX = fft2(X);
%   SortedAbsoluteValuesOfFourierCoefficients = sort(abs(TwoSidedFFTofX(:)), 'de
scend');
%   figure
%   plot( SortedAbsoluteValuesOfFourierCoefficients(1:60) )
```

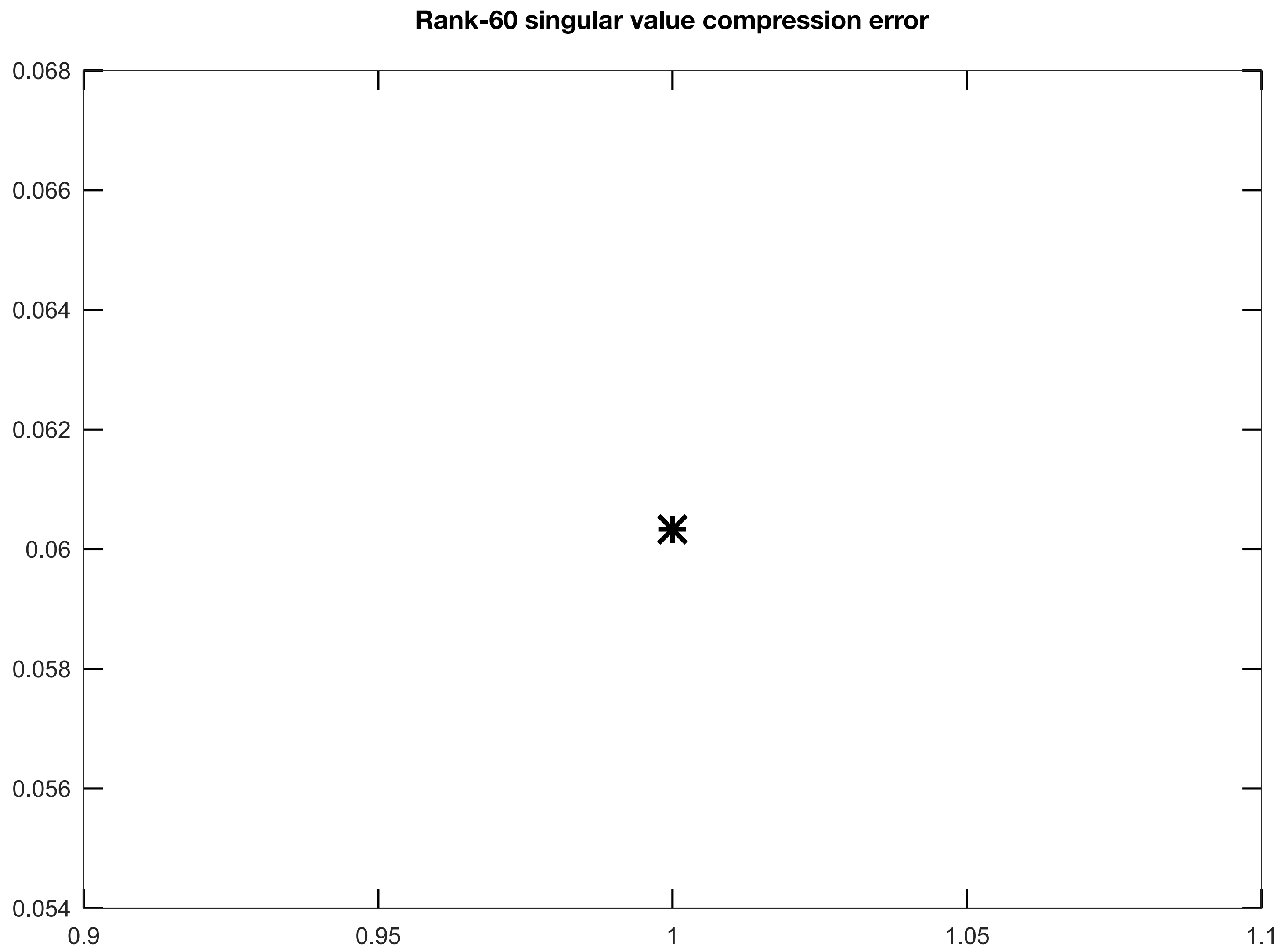


In [ ]:

```
%  plot of the two-sided DCT's k-coefficient compression error  (for k <= 60)

for i = 1:100
    X = imread(fullfile("new_faces/smiling_faces",test_smiling(i).name));
    X = padarray(imresize(X, [64 54]),[0,5]);
    dctX = dct(X);    % dct cannot run successfully on Octave
    DCTCoefficients = sort(abs(dctX(:)), 'descend');
    compression_error_dct(i) = sum(DCTCoefficients(61:end))/sum(DCTCoefficients)
;
end

plot(1:100, compression_error_dct)
title("Two-sided DCT's 60-coefficient compression error")
```

In [81]:

```
%  plot of the rank-k singular value compression error (for k <= 60)

compression_error_sv = sum(singular_values_smiling(61:end))/sum(abs(singular_val
ues_smiling));

plot(compression_error_sv, '*')
title("Rank-60 singular value compression error")
```



In [80]:

```
%  which of the 4 compression schemes has lowest average compression error?

avg_error_eface = sum(compression_error_eface)/numel(compression_error_eface);
avg_error_fft = sum(compression_error_fft)/numel(compression_error_fft);
% avg_error_dct = sum(compression_error_dct)/numel(compression_error_dct);
avg_error_sv = compression_error_sv;

% min_error = min([avg_error_eface avg_error_fft avg_error_dct avg_error_sv]);
min_error = min([avg_error_eface avg_error_fft avg_error_sv]);

if (min_error == avg_error_eface)
    printf("Eigenface\n");
elseif (min_error == avg_error_fft)
    printf("FFT\n");
% elseif (min_error == avg_error_dct)
%     printf("DCT\n");
elseif (min_error == avg_error_sv)
    printf("Singular values\n");
endif
```

```
Singular values
```