Table of Contents


```
clc; clear;
```

Part 1.1

Complete and turn in the function princomp.m. Do not use the built-in function for PCA (but do use, e.g., eigs or svd). Notes: This function removes the mean from the data—this makes computing the covariance matrix easier. princomp.m should then simply return the leading eigenvectors in W (and associated eigenvalues λ) of said covariance matrix. You can also easily project the zero-mean data onto these to get lower dimensional representations (the latent variables zi for each xi). To later reconstruct data (approximations) from latent data, use zi and principal components; do not forget to add the mean back, at this time!
****

I implemented PCA algorithm using eigs in princomp.m. To test that it works we compare to the built in function on the hald.ingredients dataset. The first two columns show my output, and the 2nd two columns show the output of the built in PCA function.

```
load hald
M = 2;
X = ingredients;
[W, Z, mu, lambda] = princomp(X,M);
Q = pca(X');
high_dim_case = [W Q(:, 1:M)]

X = X';
[W, Z, mu, lambda] = princomp(X,M);
Q = pca(X');
low_dim_case = [W Q(:, 1:M)]
clear;  % remove the dummy variables from the example
```

```
high_dim_case =
```

```
    0.1289    0.5651    0.1289    0.5651
    0.1453    0.4630    0.1453    0.4630
    0.3097   -0.0673    0.3097   -0.0673
    0.1418    0.3611    0.1418    0.3611
    0.3035    0.1325    0.3035    0.1325
    0.3000   -0.0418    0.3000   -0.0418
    0.4139   -0.2805    0.4139   -0.2805
    0.1384    0.3390    0.1384    0.3390
    0.3007   -0.0152    0.3007   -0.0152
    0.2236    0.0005    0.2236    0.0005
    0.1959    0.1787    0.1959    0.1787
    0.3738   -0.2076    0.3738   -0.2076
    0.3939   -0.2079    0.3939   -0.2079


low_dim_case =

   -0.0678    0.6460   -0.0678   -0.6460
   -0.6785    0.0200   -0.6785   -0.0200
    0.0290   -0.7553    0.0290    0.7553
    0.7309    0.1085    0.7309   -0.1085
```

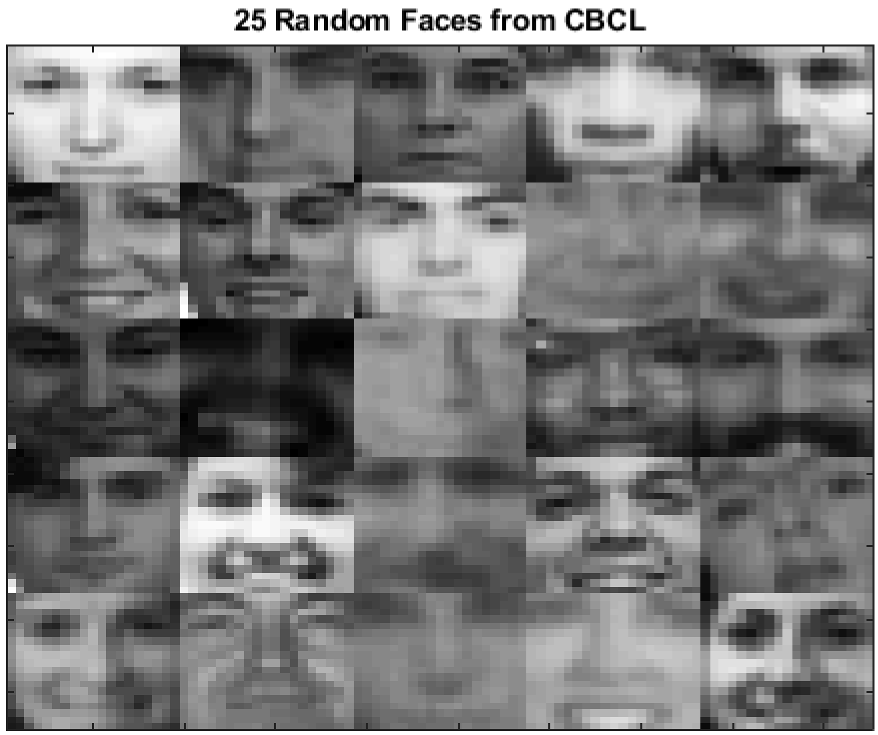
Part 1.2

Load and look at the CBCL faces data set to get a feel for it, for example by using the provided function imgrid.m with the raw data (e.g., visualize an array of 25 randomly selected faces).

```
cbcl = load('cbcl.mat');
faces = randperm(2900, 25)
display_data = cbcl.X(:, faces);

figure()
imgrid(display_data, cbcl.dims, [5, 5]);
title("25 Random Faces from CBCL");
```

```
faces =

  Columns 1 through 6

        2363         2626          369         2647         1832
 283

  Columns 7 through 12

         806         1583         2770         2790          456
 2805

  Columns 13 through 18

        2765         1402         2310          410         1217
 2641
```

```
Columns 19 through 24

      2284         2888         1889          103         2444
2688

Column 25

      1953
```



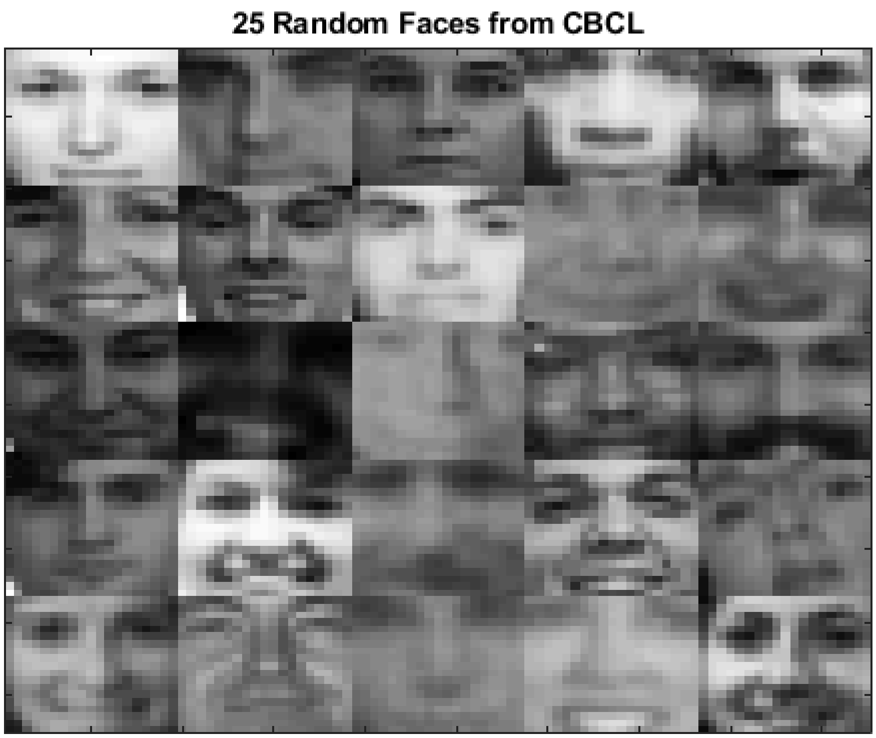
Part 1.3

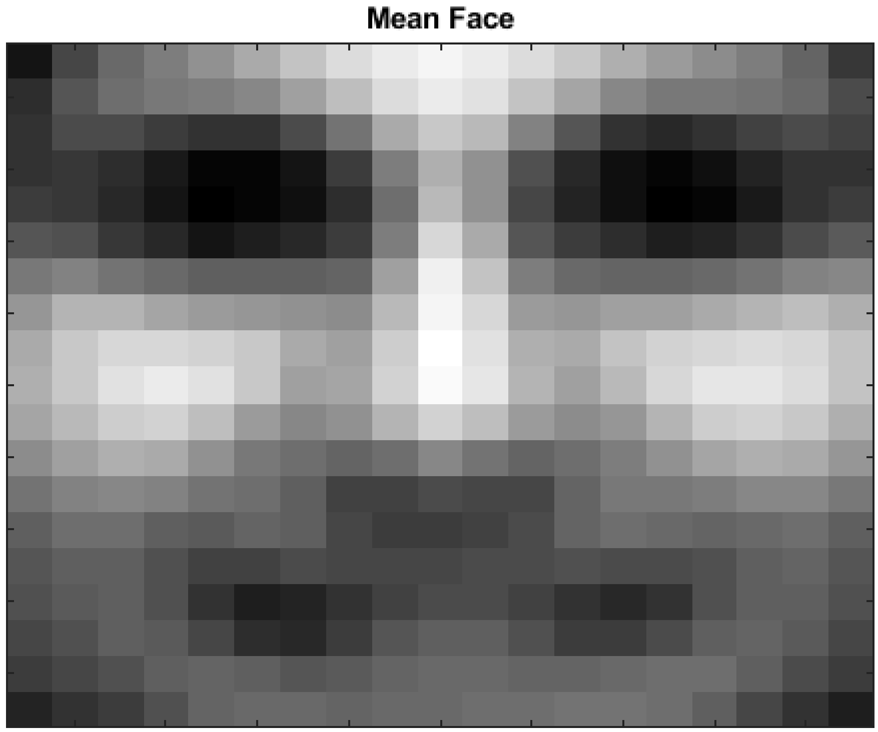
Run PCA on the CBCL faces data set and turn in a figure with the mean face and the first five principal components (leading eigenfaces). Explain why you think they look the way they do, and what sort of information in the data they are modeling. Note: for higher components this becomes more difficult. Explain as much as you can
****

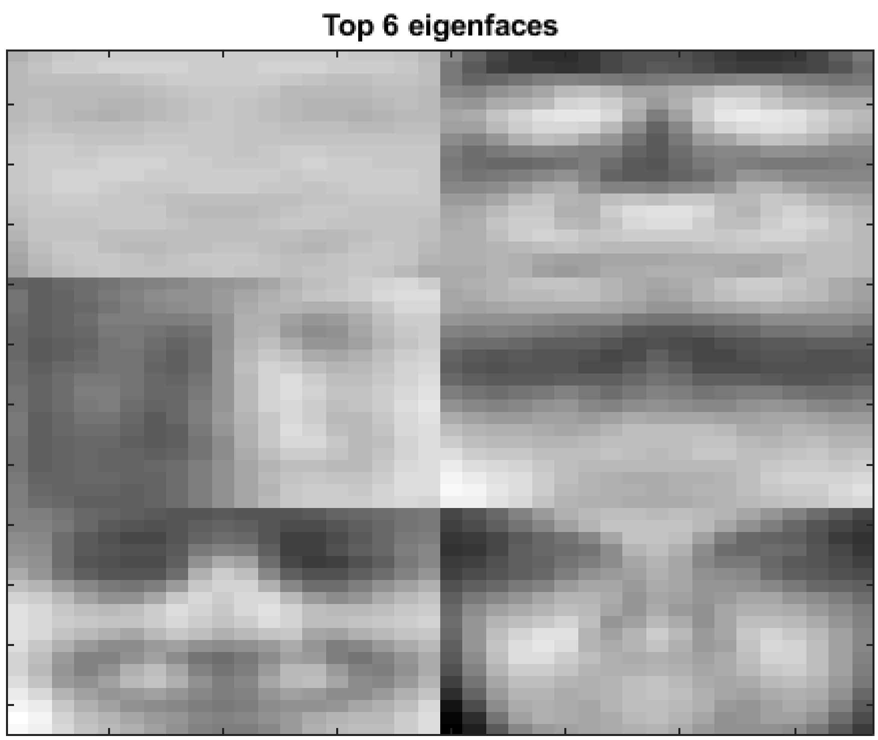
I notice that the mean face has very strong forehead, nose, and cheeks. Its eyes are very vague and black and also has a very dark mouth. This is probably because there is a lot of variation in these areas while the nose, forehead and cheeks are all typically quite well lit. The eigenfaces are a bit harder to interpret. The first one (top left) is just a broad outline of a face, while the 2nd one (top right) is the eyes and mouth. I have no idea what the third one is. The 4th one is just the eyes. And the fith looks pretty normal, I would say it is more on the mouth structure. Then the 6th we can see some eyebrows and the full face is not in the frame, so perhaps capturing more facial hair or people with small heads.

```
X = cbcl.X;
```

```
M = 6;
[W, Z, mu, lambda] = princomp(X,M);
figure()
imgrid(mu, cbcl.dims, [1, 1]);  % Mean face (kind of insulting)
title("Mean Face");
figure()
imgrid(W, cbcl.dims, [3, 2]);  % 5 eigenfaces
title("Top 6 eigenfaces");
```




Mean Face

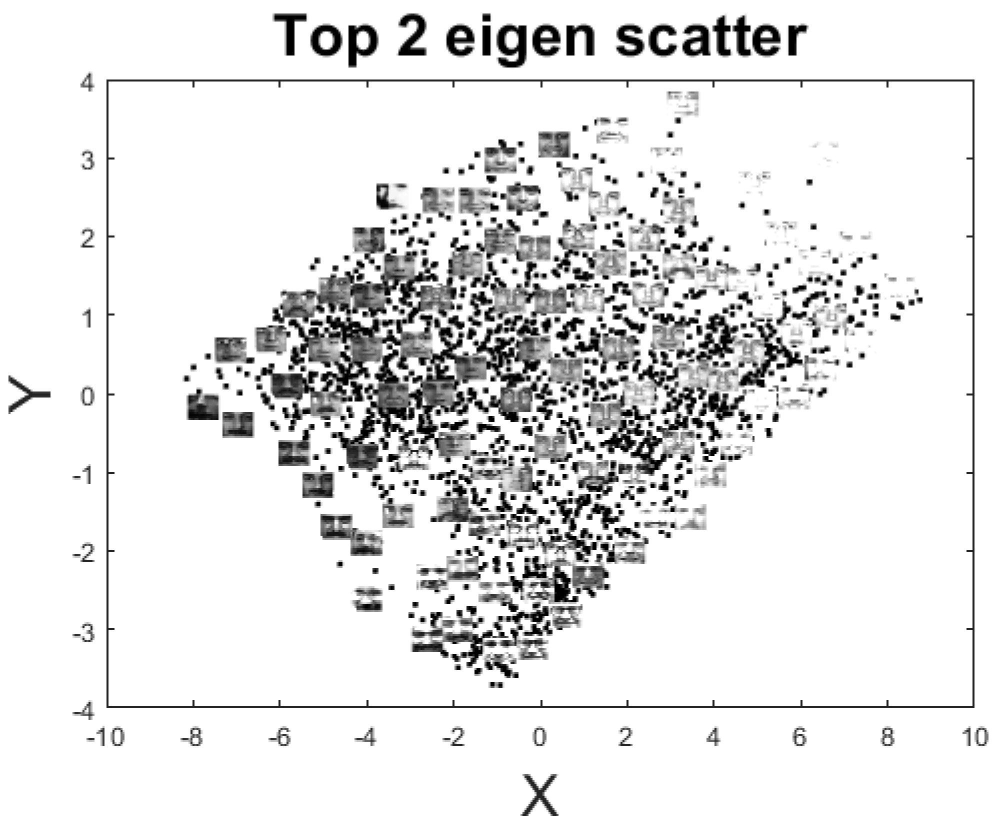

Top 6 eigenfaces

Part 1.4

Project the faces onto the first two principal components and plot/turn in the result as a point cloud in 2D (the best representation of the data in two dimensions), using either scatter or imcloud.m. **

I notice that from left to right the pictures get brighter and darker (from the first principal component), then in the second component (vertical) we see more variation in the brightness of the eyes and mouth.

```
M = 2;
[W, Z, mu, lambda] = princomp(X,M);
figure()
imcloud(Z, X, cbcl.dims)
title("Top 2 eigen scatter");
```



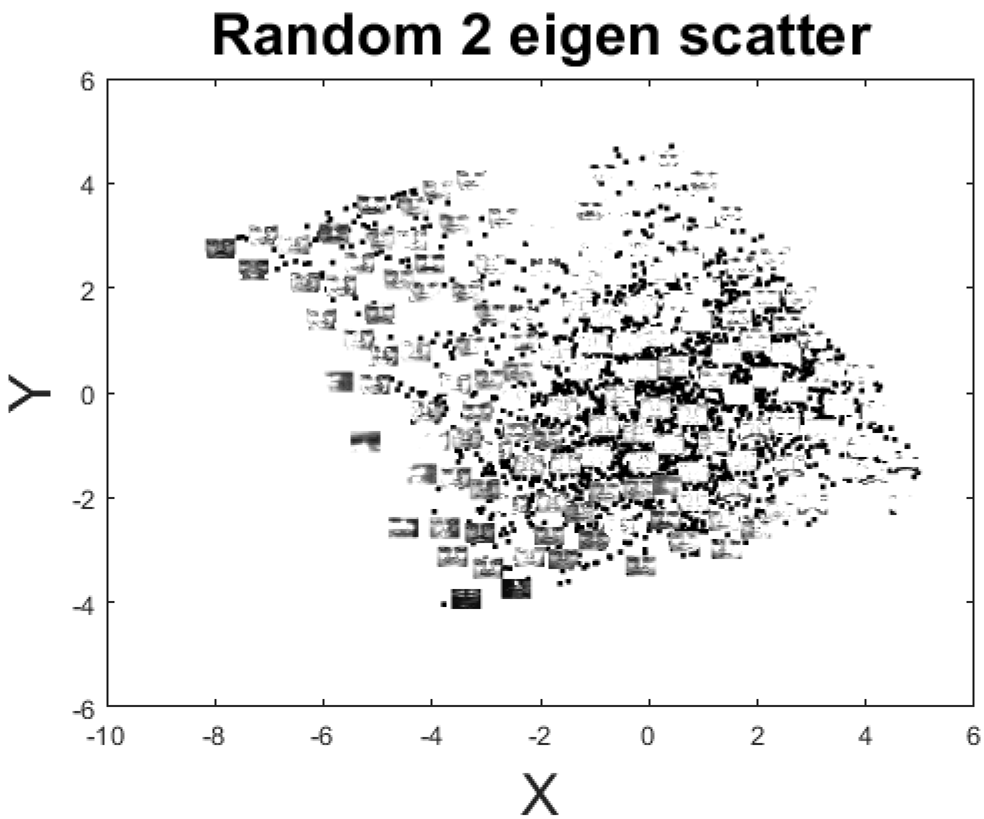
Part 1.5

Use the imcloud.m function to create this plot for projections onto other significant PCs, and explain the results you see as they compare to your response from looking at the PC vectors. How might you change or improve your description of the information captured by the principal components?

```
M = 6;
[W, Z, mu, lambda] = princomp(X,M);
comps = randperm(6, 2)
W2 = W(:, comps);
P = W2' \ (W2' * W2);  % projection matrix formula
```

```
Z2 = P' * (X - mu);
figure()
imcloud(Z2, X, cbcl.dims)
title("Random 2 eigen scatter")
```

```
comps =

     3     5
```



Part 1.6

Plot the spectrum of your PCA: trace the eigenvalues of the covariance matrix in descending order. Does this maybe tell you how many principal components should be selected for “optimal” dimensionality reduction?
****
We see that the eigenvalues steadily decrease. I don't see a super major drop-off in the eigenvalues. If there was, it would be best to take all the values before the drop-off for good dimensionality reduction.

```
lambda
```

```
lambda =

   1.0e+04 *
```

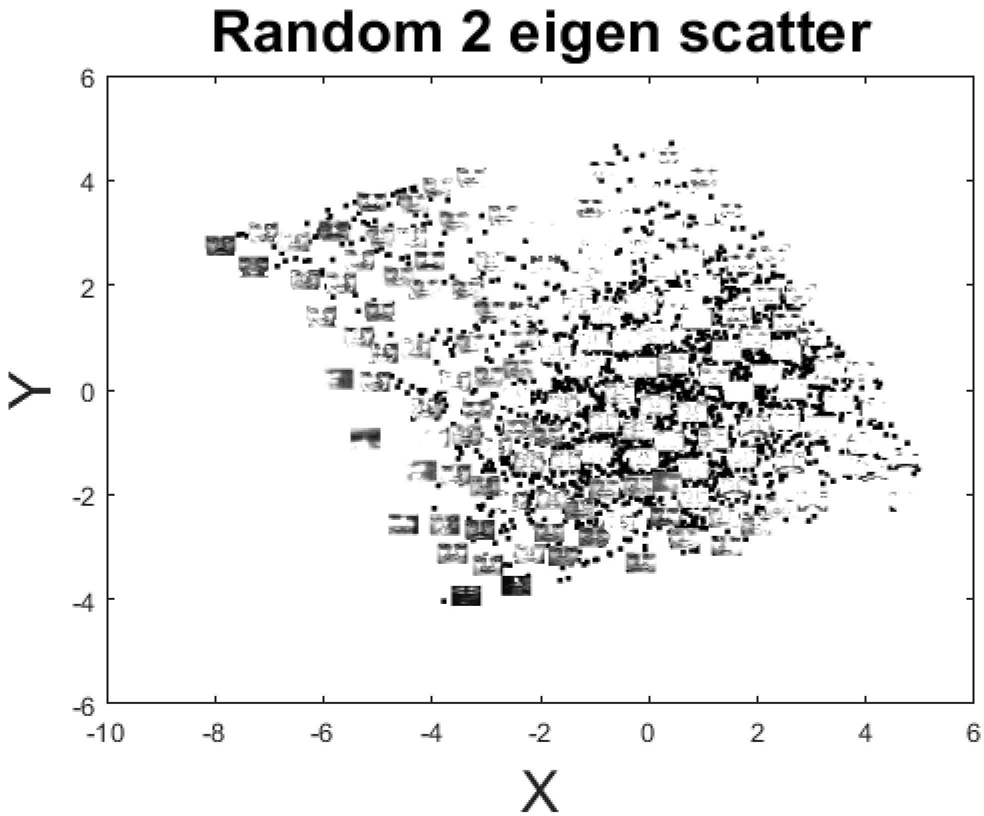
```
2.2697
0.4478
0.2512
0.1322
0.1107
0.0978
```

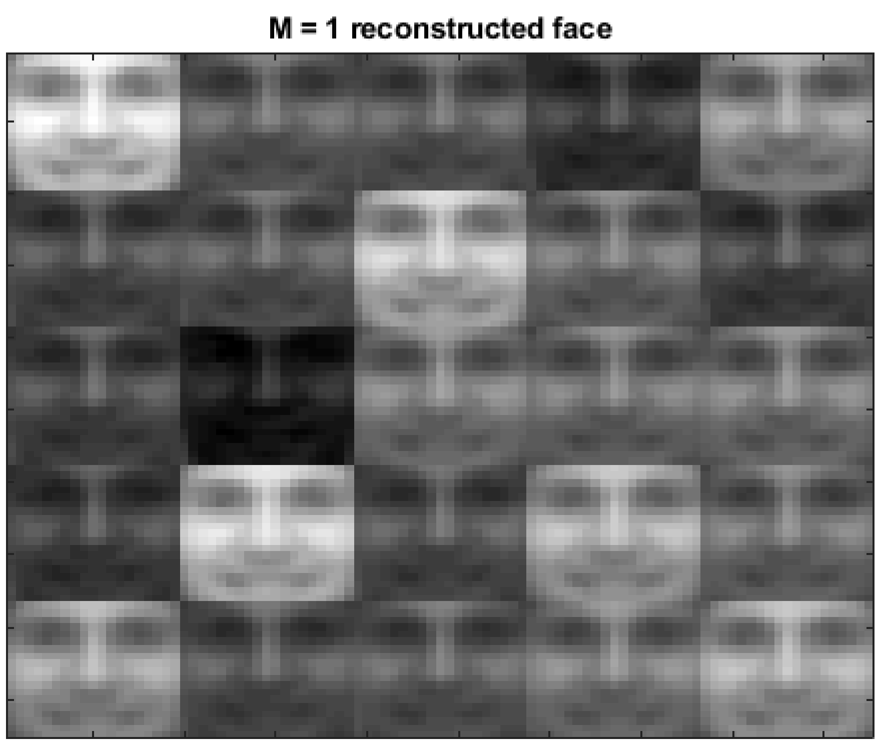
Part 1.7

Reconstruct faces from varying number of principal components. For example, try M = 1,2,5,10,25, and compare the reconstructions against original faces (you can chose the same 25 faces as under item 1, for example, and use imgrid.m with the reconstructed data). What do you observe? ***

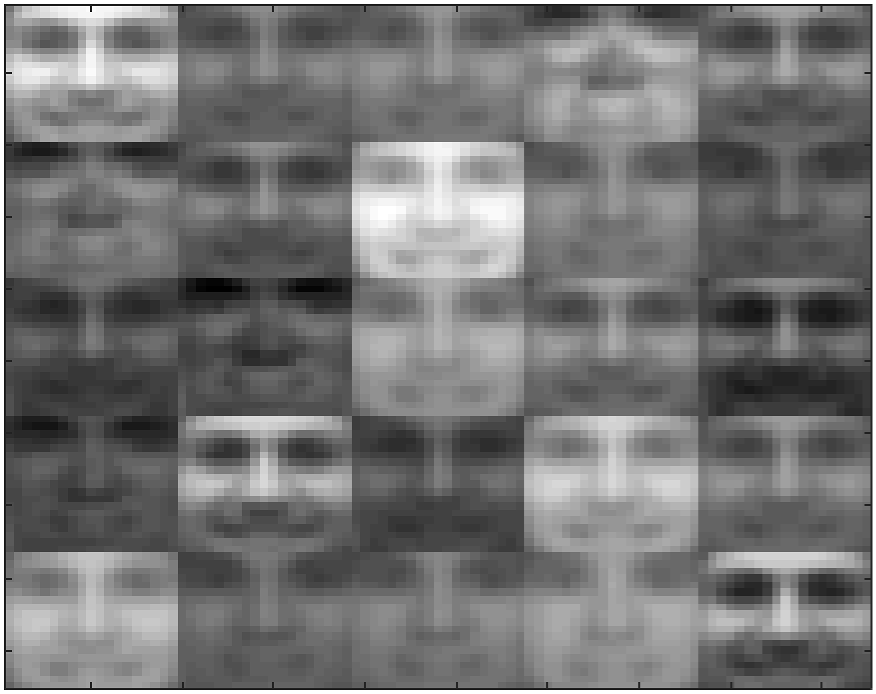
It does a pretty good job of reconstructing the faces. At M=25 they are quite recognizable as the same face. M=1 is jsut lighter and darker versions of the same (mean) face. But as M increases, more defining characteristics become visible. It is interesting how some (presumably) more rare characteristics, like glasses are still not present at M = 25.

```
for m = [1, 2, 5, 10, 25]
    [W, Z, mu, lambda] = princomp(X,m);
    Re = W * Z + mu;

    display_data = Re(:, faces);
    figure()
    imgrid(display_data, cbcl.dims, [5, 5]);
    title("M = " + m + " reconstructed face");
end
```

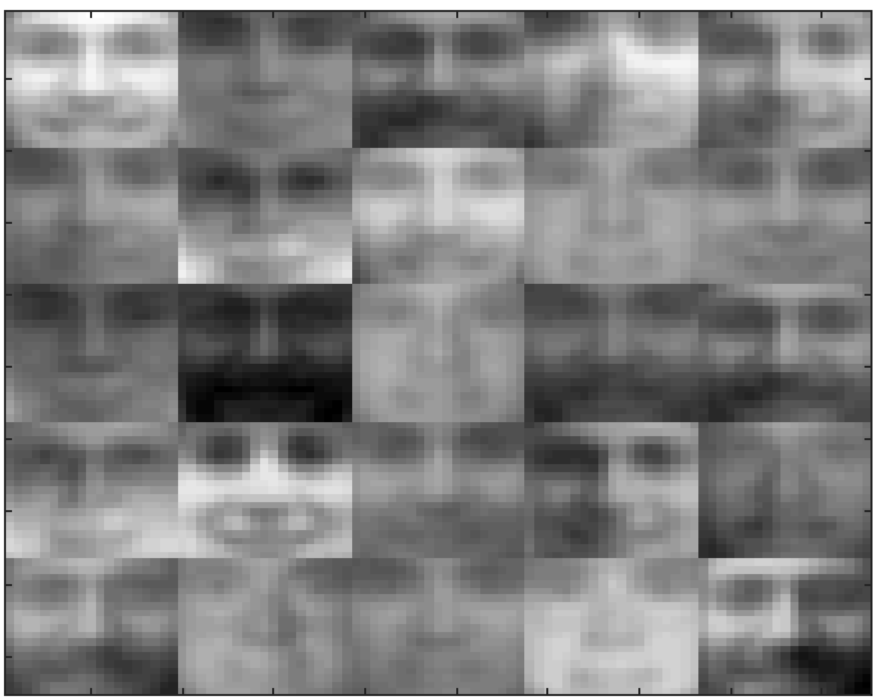

Random 2 eigen scatter
6
4
2
0
-2
-4
-6
Y
-10
-8
-6
-4
-2
0
2
4
6
X


M = 1 reconstructed face

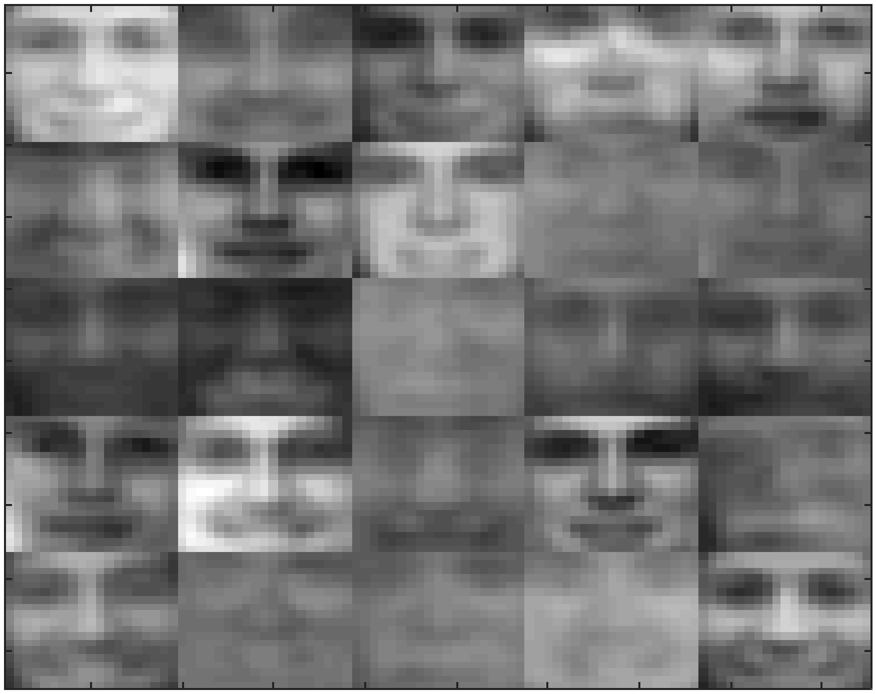
M = 2 reconstructed face



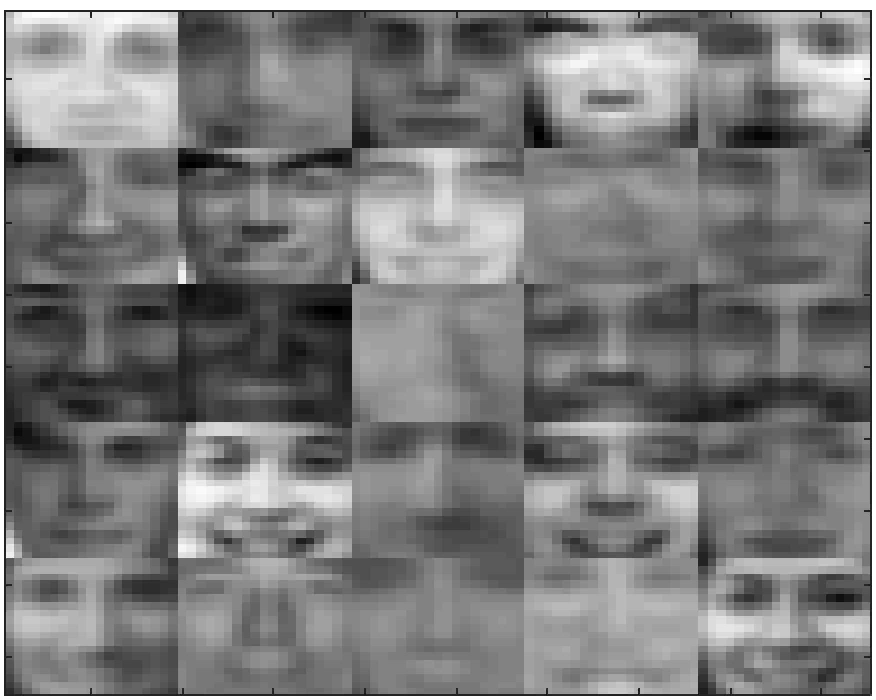
M = 5 reconstructed face



M = 10 reconstructed face



M = 25 reconstructed face



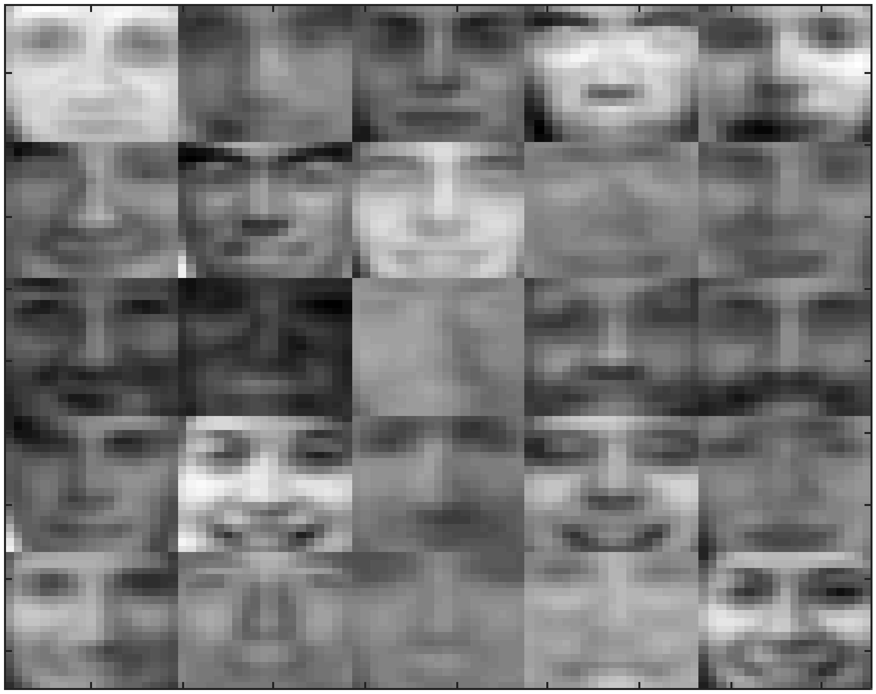
Part 2.2

Load and look at the MNIST numbers data set to get a feel for it, for example by using the provided function imgrid.m with the raw data (e.g., visualize an array of 25 randomly selected numbers).

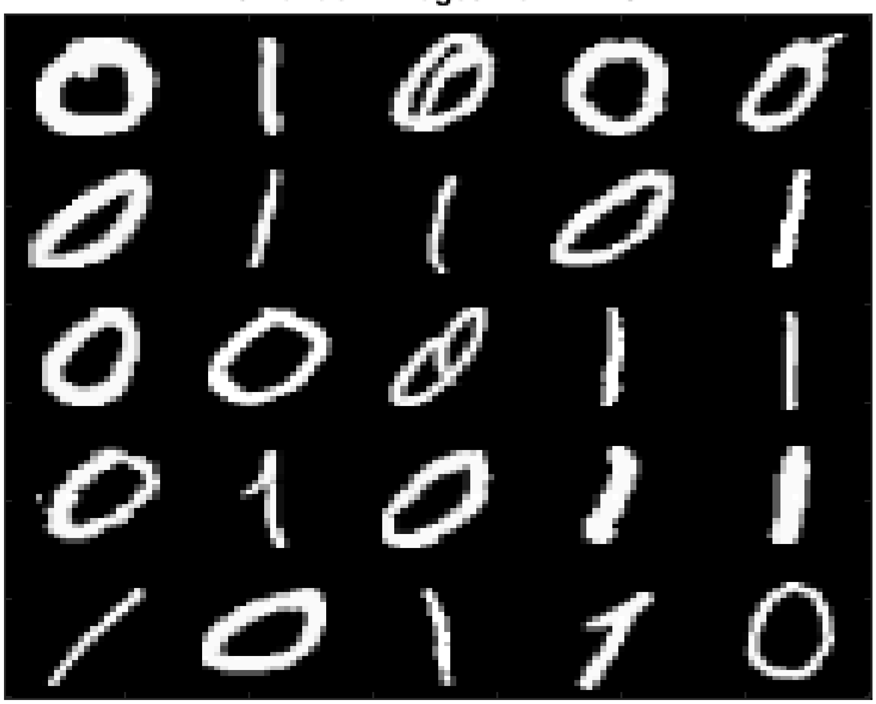
```
mnist = load("mnist.mat");
faces = randperm(12665, 25)
display_data = mnist.X(:, faces);

figure()
imgrid(display_data, mnist.dims, [5, 5]);
title("25 Random Images from MNIST");
```

```
faces =

  Columns 1 through 6

        2169        8942         404        3507         585
 1230

  Columns 7 through 12

       10425        8796        4014       12027         436
 5552

  Columns 13 through 18

        4828        9686       10061        2364        6196
 5636

  Columns 19 through 24

        8174        8971        9544        3491        8594
 8282

  Column 25

        2056
```

M = 25 reconstructed face



25 Random Images from MNIST



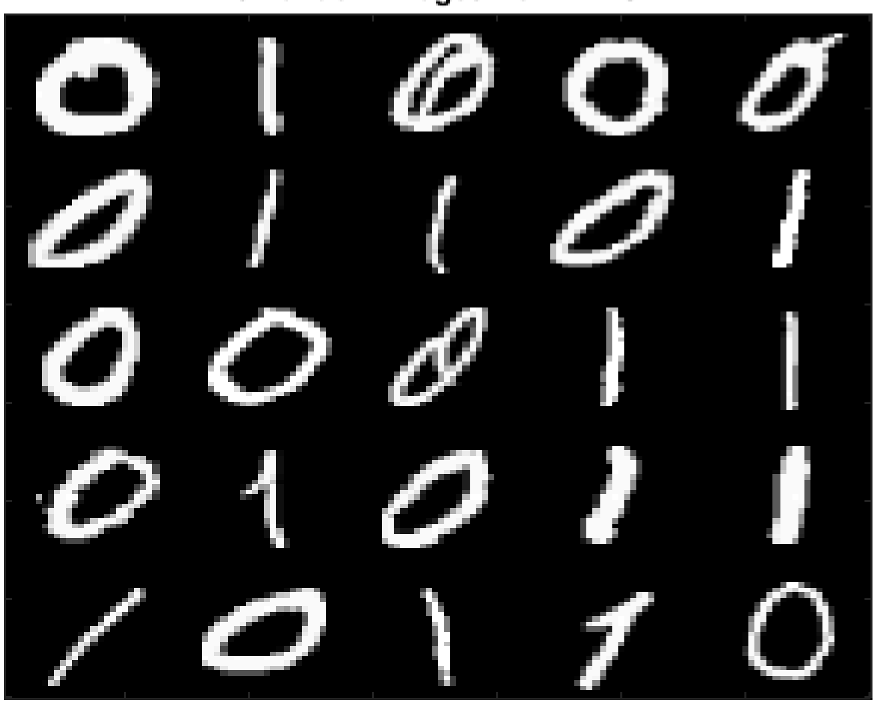
Part 2.3

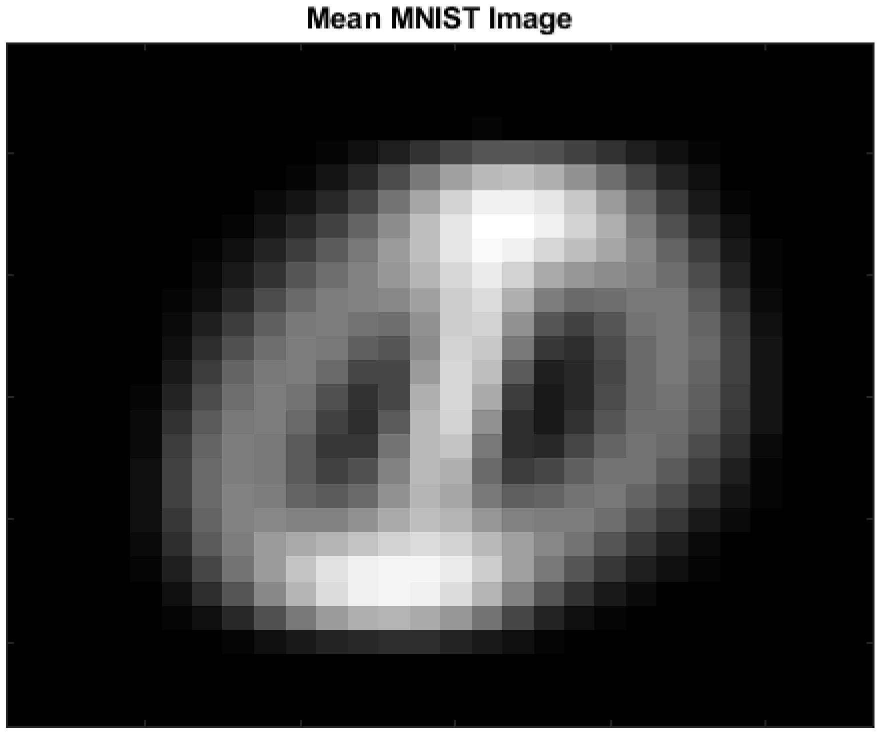
Run PCA on the MNIST numbers data set and turn in a figure with the mean number and the first five principal components (leading eigennumbers). Explain why you think they look the way they do, and what sort of information in the data they are modeling. Note: for higher components this becomes more difficult. Explain as much as you can
****

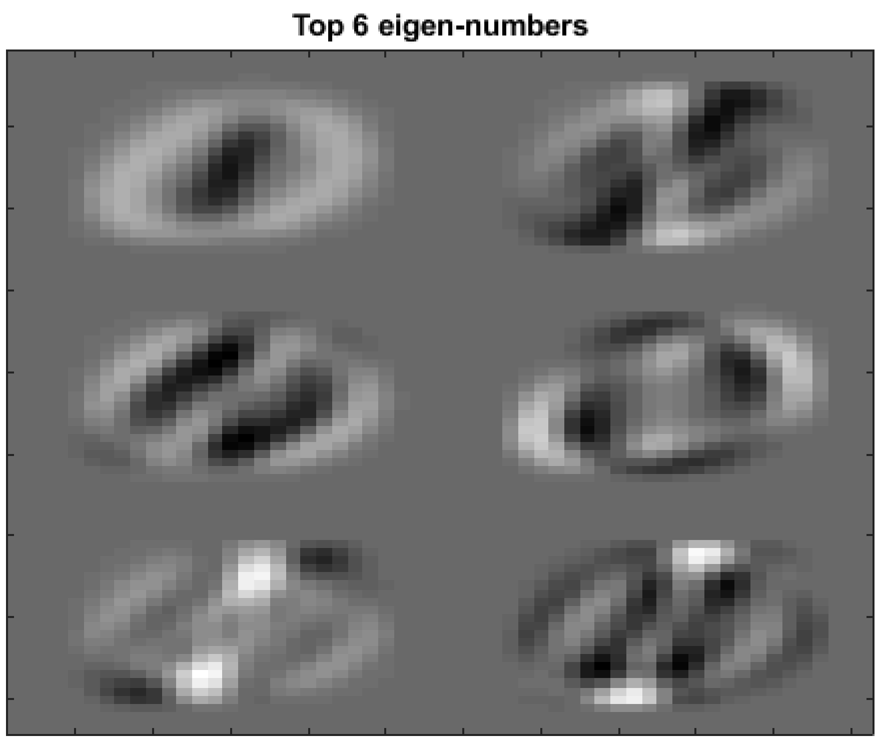
The mean image looks like a 1 with a 0 around it. This makes a ton of sense as this is what is in the dataset, just 1 and 0. So the mean would be the average of these, which is a circle with a line through it. The first two eigenvalues are pretty distinctive. The first one is mostly a zero, while the second one is a one.

```
X = mnist.X;
M = 6;
[W, Z, mu, lambda] = princomp(X,M);
figure()
imgrid(mu, mnist.dims, [1, 1]);  % Mean face (kind of insulting)
title("Mean MNIST Image");
figure()
imgrid(W, mnist.dims, [3, 2]);  % 5 eigenfaces
title("Top 6 eigen-numbers");
```

25 Random Images from MNIST




Mean MNIST Image

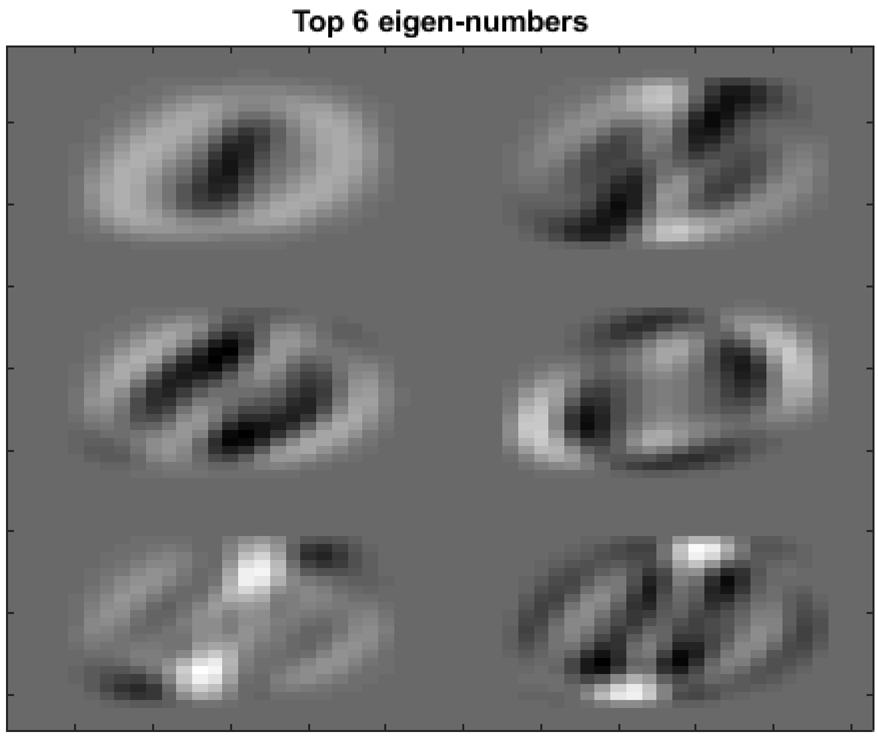

Top 6 eigen-numbers

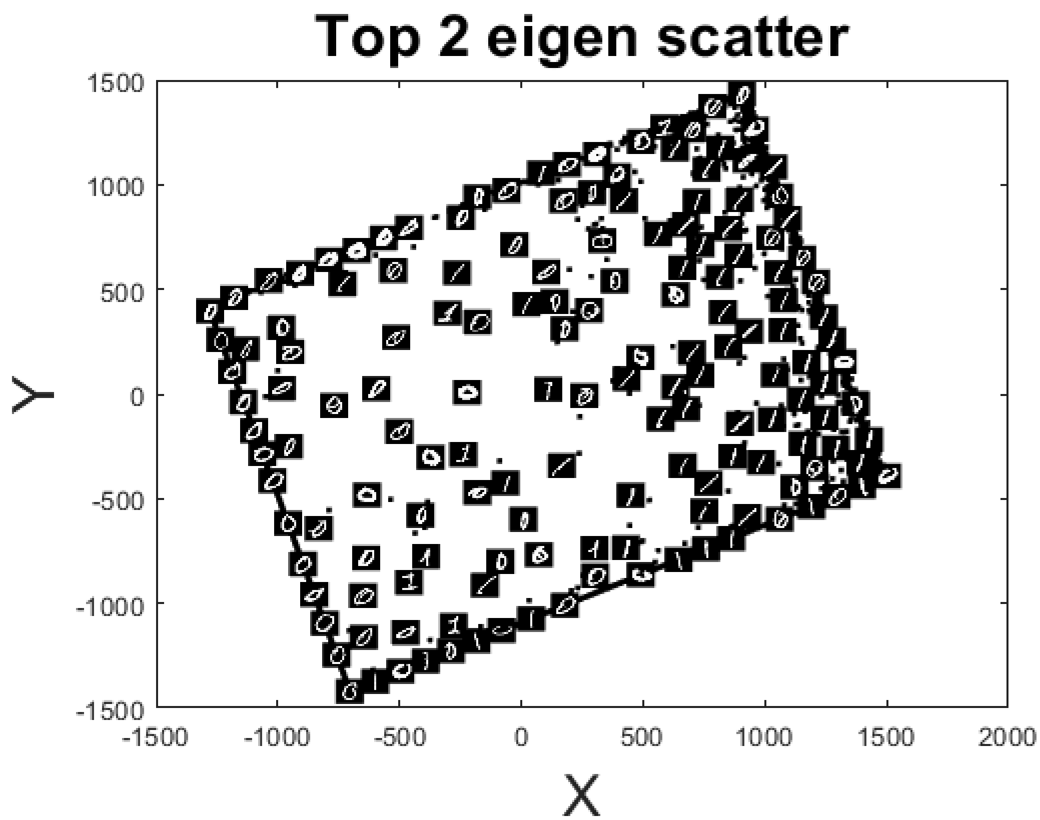
Part 2.4

Project the faces onto the first two principal components and plot/turn in the result as a point cloud in 2D (the best representation of the data in two dimensions), using either scatter or imcloud.m. **

I notice that from left to right there are more ones on the right than on the left. So the X eigenvector does a pretty good job of distinguishing between the two numbers. I am not sure what difference there is top to bottom, perhaps slantyness?

```
M = 2;
[W, Z, mu, lambda] = princomp(X,M);
figure()
imcloud(Z, X, mnist.dims)
title("Top 2 eigen scatter");
```



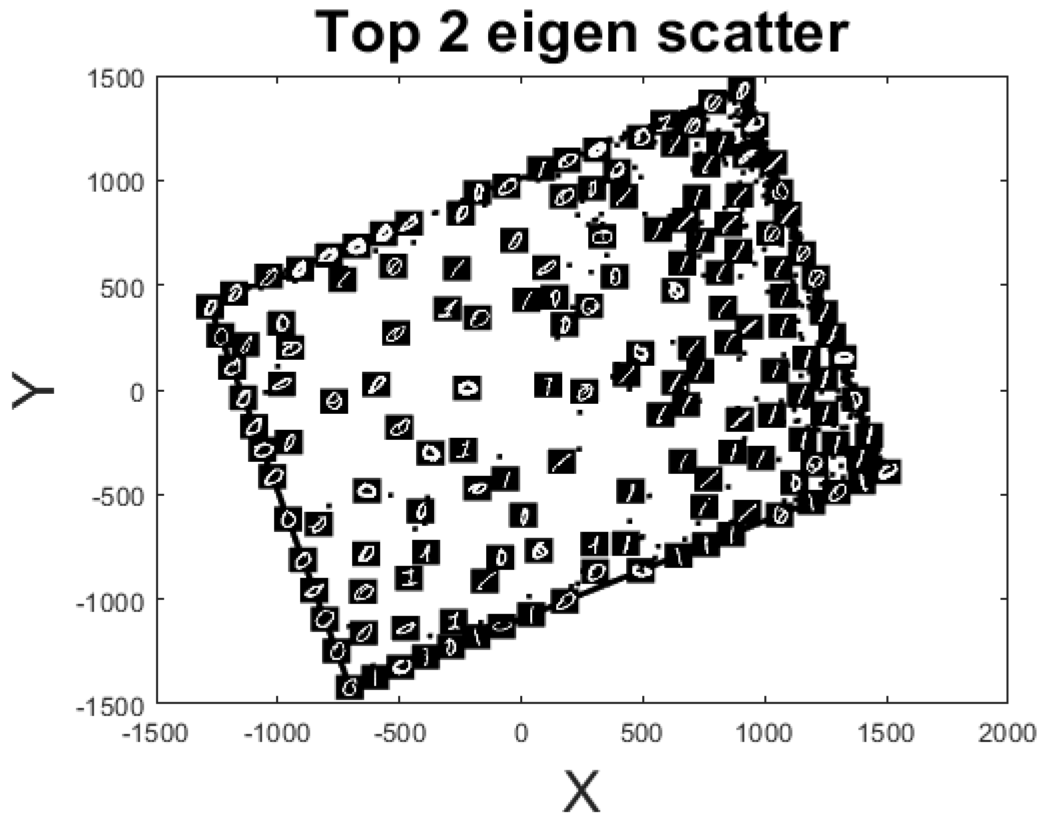


Part 2.5

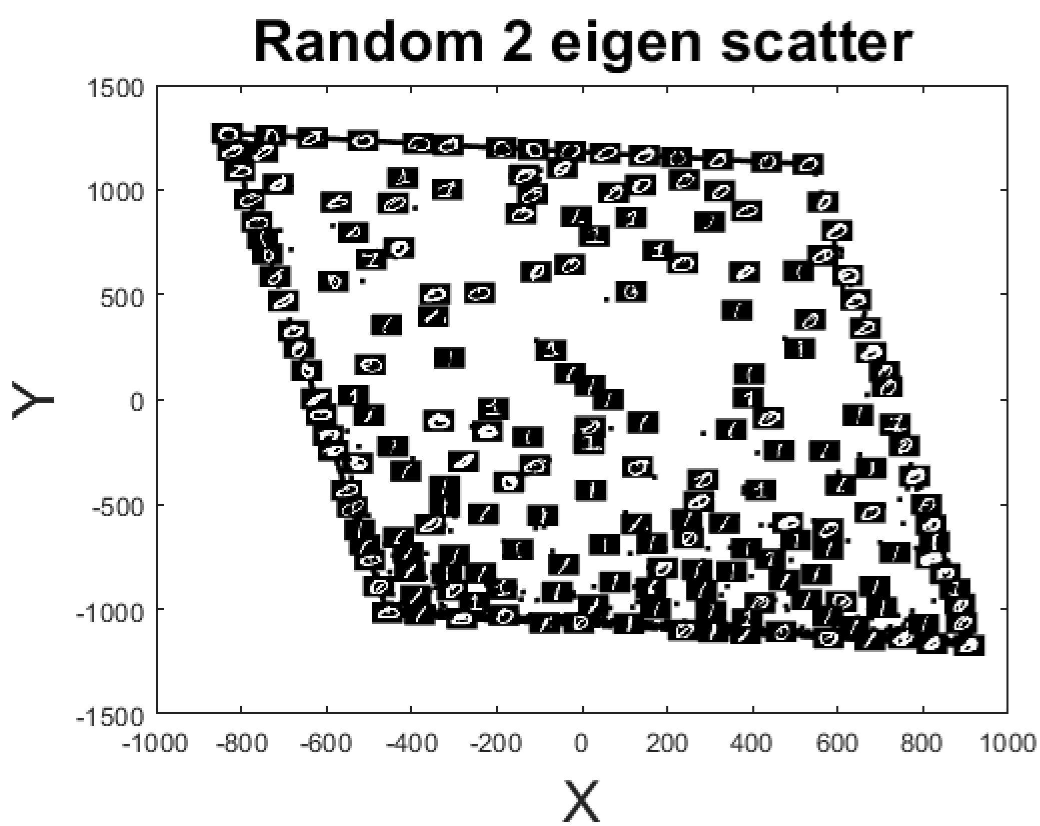
Use the imcloud.m function to create this plot for projections onto other significant PCs, and explain the results you see as they compare to your response from looking at the PC vectors. How might you change or improve your description of the information captured by the principal components?

```
M = 6;
[W, Z, mu, lambda] = princomp(X,M);
comps = randperm(6, 2)
W2 = W(:, comps);
P = W2' \ (W2' * W2);  % projection matrix formula
Z2 = P' * (X - mu);
figure()
imcloud(Z2, X, mnist.dims)
title("Random 2 eigen scatter")
```

```
comps =

     6     1
```



Top 2 eigen scatter
1500
1000
500
0
-500
-1000
-1500
Y
-1500
-1000
-500
0
500
1000
1500
2000
X



Random 2 eigen scatter
1500
1000
500
0
-500
-1000
-1500
Y
-1000
-800
-600
-400
-200
0
200
400
600
800
1000
X

Part 2.6

Plot the spectrum of your PCA: trace the eigenvalues of the covariance matrix in descending order. Does this maybe tell you how many principal components should be selected for "optimal" dimensionality reduction?
****

The vast majority is contained within the first eigenvector as it is 3 times that of even the second vector. This also helps explain the difference we see between zero and one left to right as it captures most of our variation.

```
lambda
```

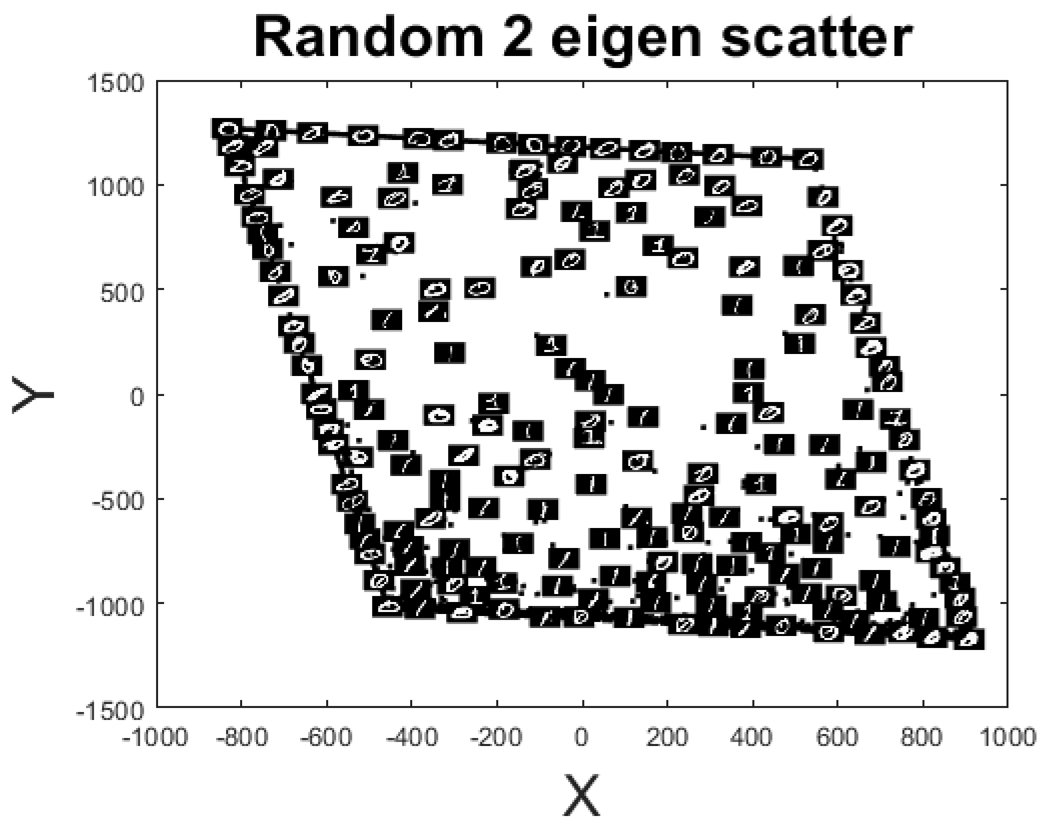
```
lambda =

   1.0e+10 *

    1.3090
    0.3697
    0.3298
    0.2265
    0.1594
    0.1383
```

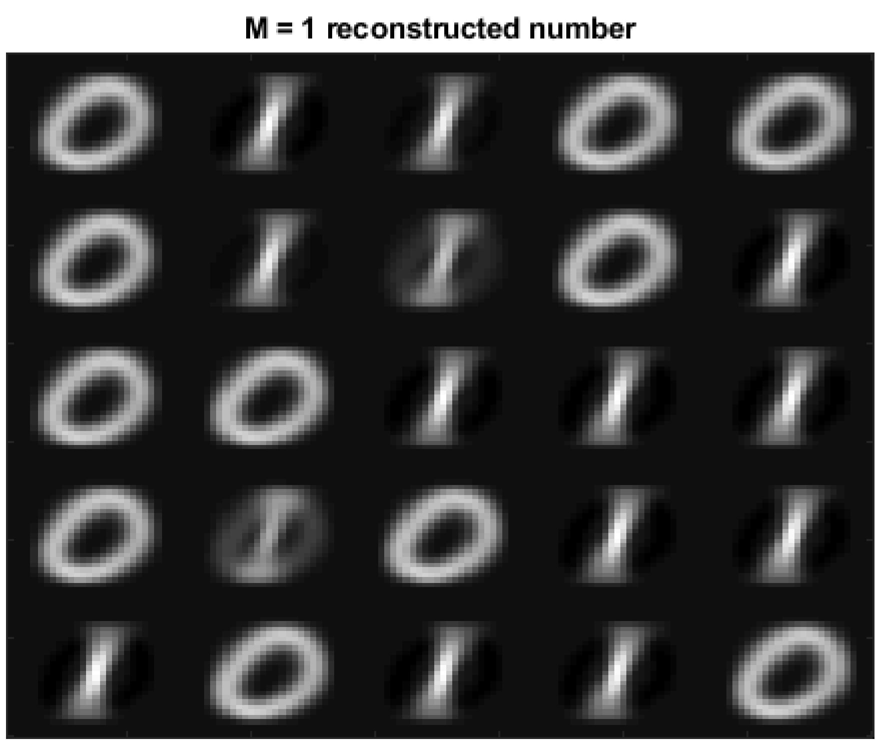
Part 2.7

Reconstruct faces from varying number of principal components. For example, try M = 1,2,5,10,25, and compare the reconstructions against original numbers (you can chose the same 25 faces as under item 1, for example, and use imgrid.m with the reconstructed data). What do you observe?
****

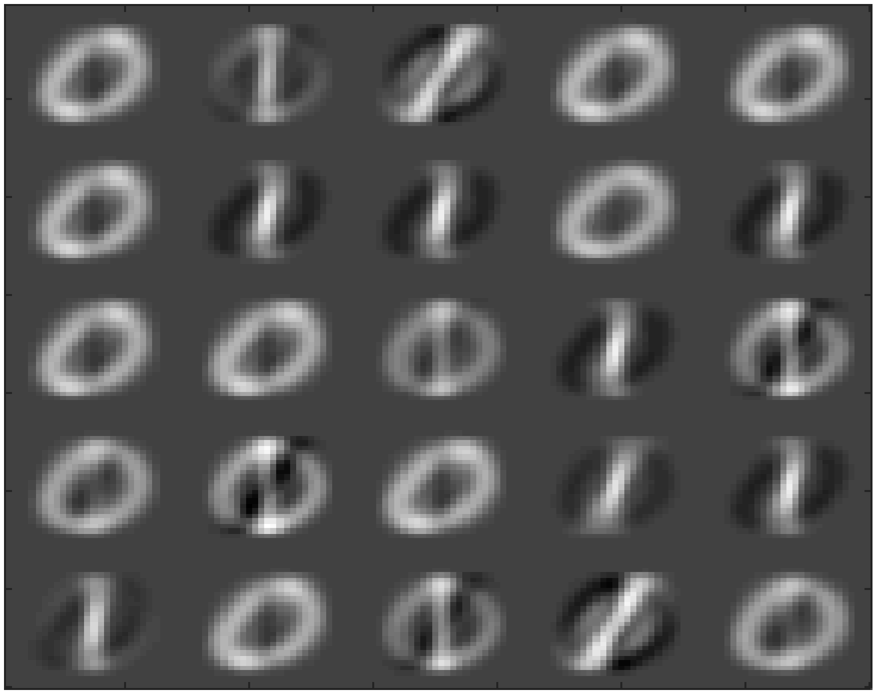
It does a pretty good job of reconstructing the numbers. It is very very clean at M=1, which makes sense from 2.6.

```
for m = [1, 2, 5, 10, 25]
    [W, Z, mu, lambda] = princomp(X,m);
    Re = W * Z + mu;

    display_data = Re(:, faces);
    figure()
    imgrid(display_data, mnist.dims, [5, 5]);
    title("M = " + m + " reconstructed number");
end
```

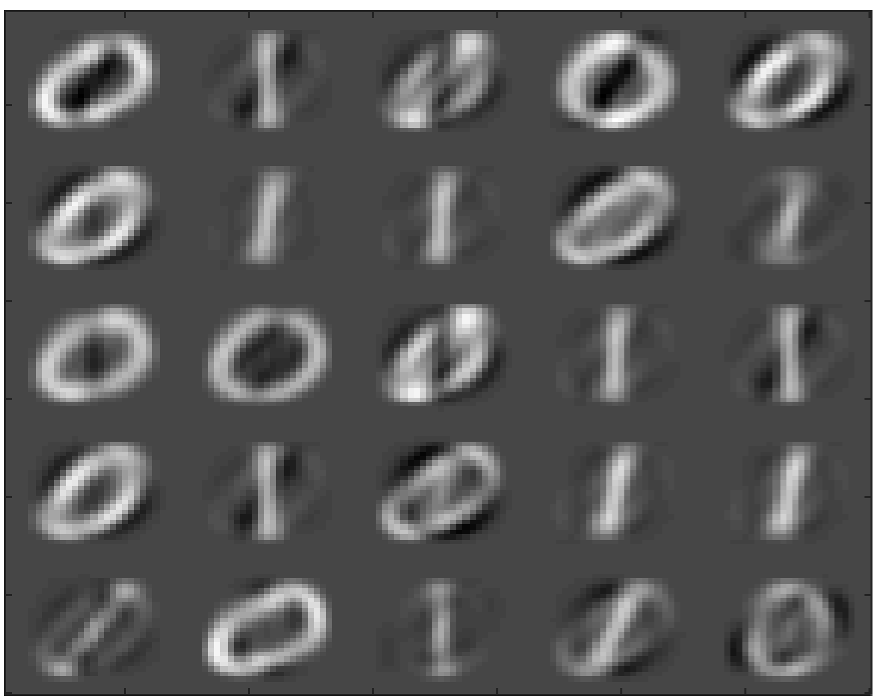

Random 2 eigen scatter
1500
1000
500
0
-500
-1000
-1500
-1000
-800
-600
-400
-200
0
200
400
600
800
1000
Y
X


M = 1 reconstructed number

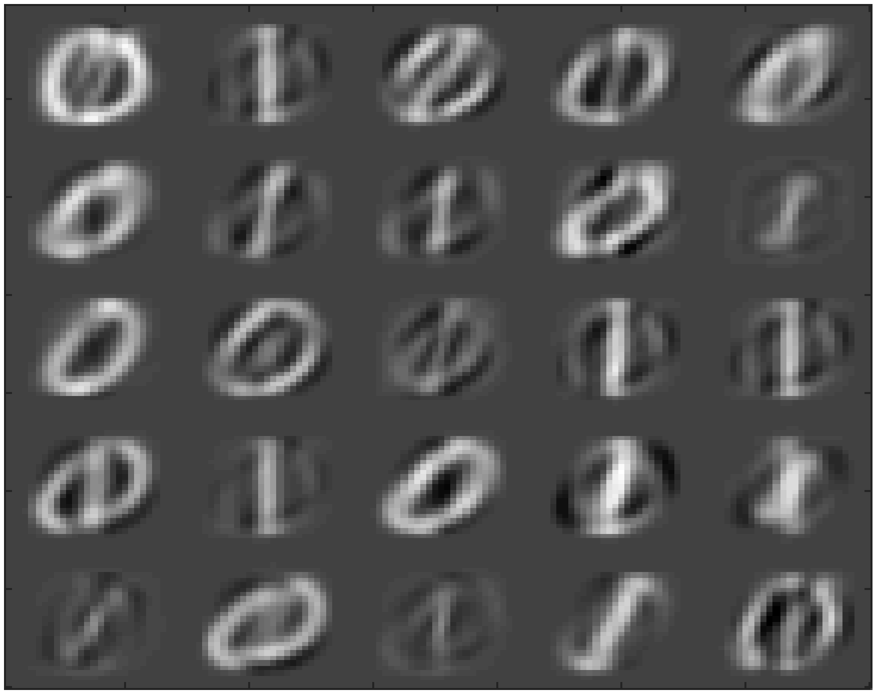
M = 2 reconstructed number



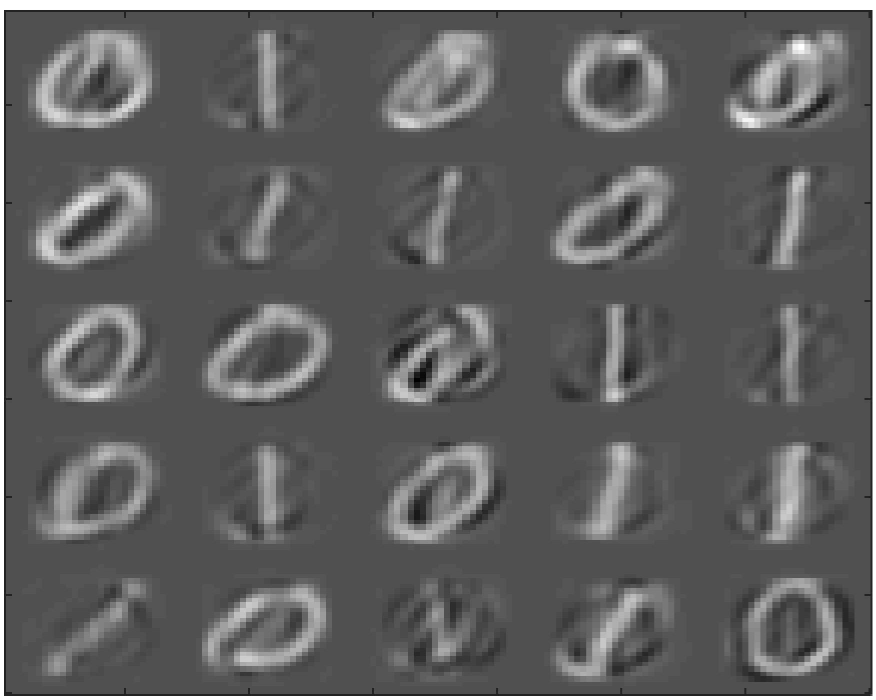
M = 5 reconstructed number



M = 10 reconstructed number



M = 25 reconstructed number



PrinComp.m

```
% PRINcipal COMPonent calculator
%   Calculates the principal components of a collection of points.
% Input:
%   X - D-by-N data matrix of N points in D dimensions.
% Output:
%   W - A D-by-M matrix containing the M principal components of the
 data.
%   Z - A M-by-N matrix containing the latent variables of the data.
%   mu - A D-by-1 vector containing the mean of the data.
%   lambda - A vector containing the eigenvalues associated with the
 above principal components.
function [W, Z, mu, lambda] = princomp(X,M)
[D, N] = size(X);

mu = mean(X, 2);
Y = X-mu;

% Compute PCA Vectors
if D > N  % high dimensional case
    [W2, V] = eigs(Y' * Y, M);
    W = Y * W2;
    W = W ./ vecnorm(W);
else  % normal case
    [W, V] = eigs(Y * Y', M);
end

% Now we need to project X onto span(W)
P = W' \ (W' * W);  % projection matrix formula
Z = P' * Y;

lambda = diag(V);  % get vector of eigenvalues
lambda = lambda(1:M);  % chop to first M values
end
```

imcloud.m

```
% IMage CLOUD generator
%   To visualize a point cloud in 2D where each point corresponds to
 an image,
%  this function generates a scatter plot with example images rendered
 over the
%  points.
% Input:
%   H - 2-by-N matrix with the location for each observation in the
 plane.
```

```
%   I - dims(1)*dims(2)-by-N matrix with image data for each point in
 the plane.
%   dims - 2D vector with the height and width of each image.
 Reshaping the columns of
%    I to this size should produce each image.
% Output:
%   None (creates a figure).

function [] = imcloud(H, I, dims)
    if size(H,2) == 2 && size(H,1) > 2
        H = H';
    end
    if dims(1) * dims(2) ~= size(I,1)
        error('Columns of I must have the same number of elements as
 given by dims.');
    end

    SUBIMG_PC1_HALF_SIZE = (max(H(1,:)) - min(H(1,:)))/50;
    SUBIMG_PC2_HALF_SIZE = (max(H(2,:)) - min(H(2,:)))/50;
    SUBIMG_SPACING = 2*SUBIMG_PC1_HALF_SIZE;

    plot(H(1,:),H(2,:),'.k');
    title('Cloud','FontSize',22);
    xlabel('X','FontSize',22);
    ylabel('Y','FontSize',22);
    hold on;
    DONE = [1e80,1e80];
    for i = 1:size(I,2)
        CENTER = [H(1,i),H(2,i)];
        if sqrt(min(sum((repmat(CENTER,[size(DONE,1),1]) -
 DONE).^2,2))) > SUBIMG_SPACING
            imagesc(...
                CENTER(1) + SUBIMG_PC1_HALF_SIZE*[-1,1],...
                CENTER(2) + SUBIMG_PC2_HALF_SIZE*[-1,1],...
                flipud(reshape(I(:,i),dims)),[0,max(max(I(:,i)))]);
            DONE = [DONE;CENTER];
        end
    end
    colormap gray;
    hold off;
    set(gcf,'Color','w');
end
```

imgrid.m

```
% IMage GRID generator
%   Given a bunch of little images, this generates a single figure
 which shows
%  a lot of them.
% Input:
```

```
%   I - dims(1)*dims(2)-by-N matrix with image data for each of N
 images.
%   dims - 2D vector with the height and width of each image.
 Reshaping the columns of
%    I to this size should produce each image.
%   gridsz - The size of the grid to show (for example, [2,3] will
 show six images in total
%    laid out in a grid with two rows).
% Output:
%   None (creates a figure).

function [] = imgrid(I, dims, gridsz)
    if dims(1) * dims(2) ~= size(I,1)
        error('Columns of I must have the same number of elements as
 given by dims.');
    end

    M = zeros(dims(1) * gridsz(1), dims(2) * gridsz(2));
    k = 1;
    for i = 1:gridsz(1)
        for j = 1:gridsz(2)
            is = (1 + (i-1)*dims(1)):((i)*dims(1));
            js = (1 + (j-1)*dims(2)):((j)*dims(2));
            M(is, js) = reshape(I(:,k), dims);
            k = k + 1;
            if k > size(I,2)
                break;
            end
        end
        if k > size(I,2)
         break;
        end
    end
    imagesc(M);
    colormap gray;
    set(gcf,'Color','w');
 set(gca,'XTickLabel',[]);
 set(gca,'YTickLabel',[]);
end
```

Published with MATLAB® R2021a