Problem 1:

The first step for solving the problem is to determine the adjacency matrix. The adjacency matrix from the figure in the problem is:

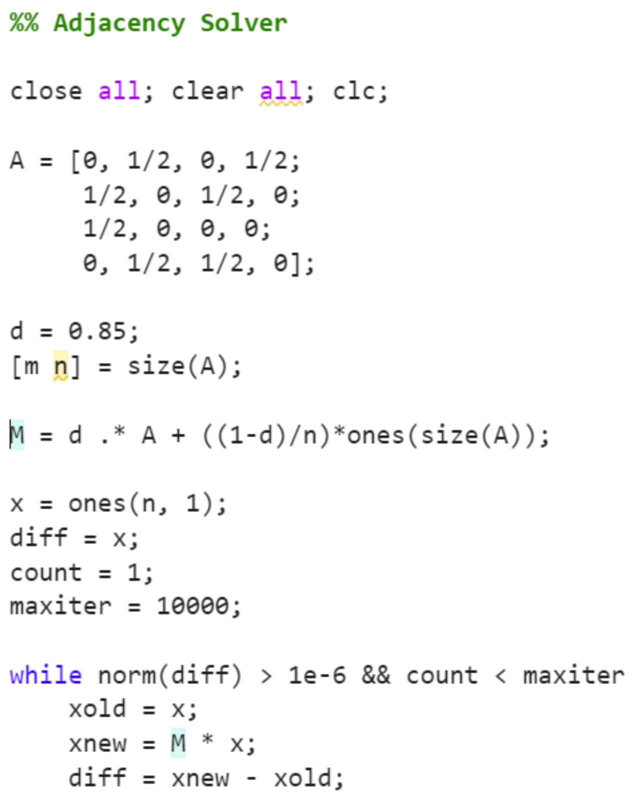
$$A = \begin{matrix} 0 & 1/2 & 0 & 1/2 \\ 1/2 & 0 & 1/2 & 0 \\ 1/2 & 0 & 0 & 0 \\ 0 & 1/2 & 1/2 & 0 \end{matrix}$$

The page rank $\vec{x}$ is calculated by using M.

$$M = dA + (1 - d)/4 * \begin{matrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{matrix}$$

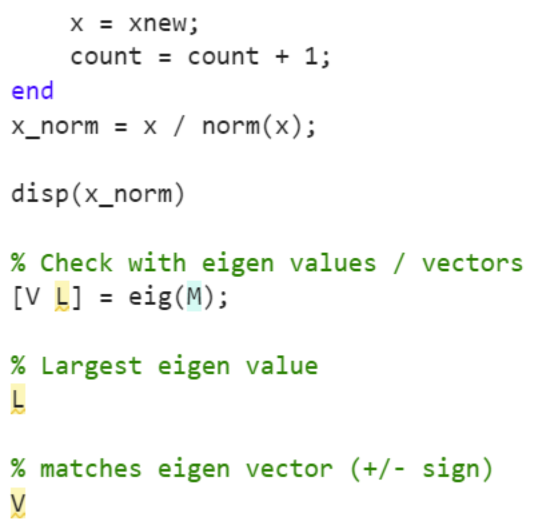
Where d = 0.85.

The result $\vec{x}$ is recalculated by being multiplied by M repeatedly until the difference in the vector L2-norm is less than 10^{-6}(error) or the maximum iterations have reached.



```
%% Adjacency Solver

close all; clear all; clc;

A = [0, 1/2, 0, 1/2;
     1/2, 0, 1/2, 0;
     1/2, 0, 0, 0;
     0, 1/2, 1/2, 0];

d = 0.85;
[m n] = size(A);

M = d .* A + ((1-d)/n)*ones(size(A));

x = ones(n, 1);
diff = x;
count = 1;
maxiter = 10000;

while norm(diff) > 1e-6 && count < maxiter
    xold = x;
    xnew = M * x;
    diff = xnew - xold;
```

Figure 1. Problem1_Using Iterative Method



```
    x = xnew;
    count = count + 1;
end
x_norm = x / norm(x);

disp(x_norm)

% Check with eigen values / vectors
[V L] = eig(M);

% Largest eigen value
L

% matches eigen vector (+/- sign)
V
```

Figure 2. Problem1_Using Iterative Method

After the iterative process concludes, the vector $\vec{x}$ is normalized by dividing it by its resulting L2-norm, yielding the answer:

$$\vec{x} = \begin{matrix} 0.5777 \\ 0.5298 \\ 0.3587 \\ 0.5069 \end{matrix}$$

This solution can be compared to the first column vector (corresponding to the highest positive eigenvalue, 0.8908, as per the Perron-Frobenius Theorem) obtained from the resulting V matrix through Singular Value Decomposition:

$$V = \begin{matrix} 0.5777 + 0.0000i & 0.0056 - 0.5499i & 0.0056 + 0.5499i & -0.1555 + 0.0000i \\ 0.5298 + 0.0000i & -0.0518 + 0.3789i & -0.0518 - 0.3789i & -0.6311 + 0.0000i \\ 0.3587 + 0.0000i & -0.3753 + 0.2306i & -0.3753 - 0.2306i & 0.3652 + 0.0000i \\ 0.5069 + 0.0000i & 0.5978 + 0.0000i & 0.5978 + 0.0000i & 0.6665 + 0.0000i \end{matrix}$$

$$\Rightarrow \; V1 = \begin{matrix} 0.5777 + 0.0000i \\ 0.5298 + 0.0000i \\ 0.3587 + 0.0000i \\ 0.5069 + 0.0000i \end{matrix}$$

The $\vec{x}$ and V1 are equals and this affirms the correctness of the iterative process, indicating that the rank from the most to the least important web page is in the order of a, d, c, and b.

Problem 2:

The first picture is just for fun :D

Figure 1: Problem 2: Original Picture

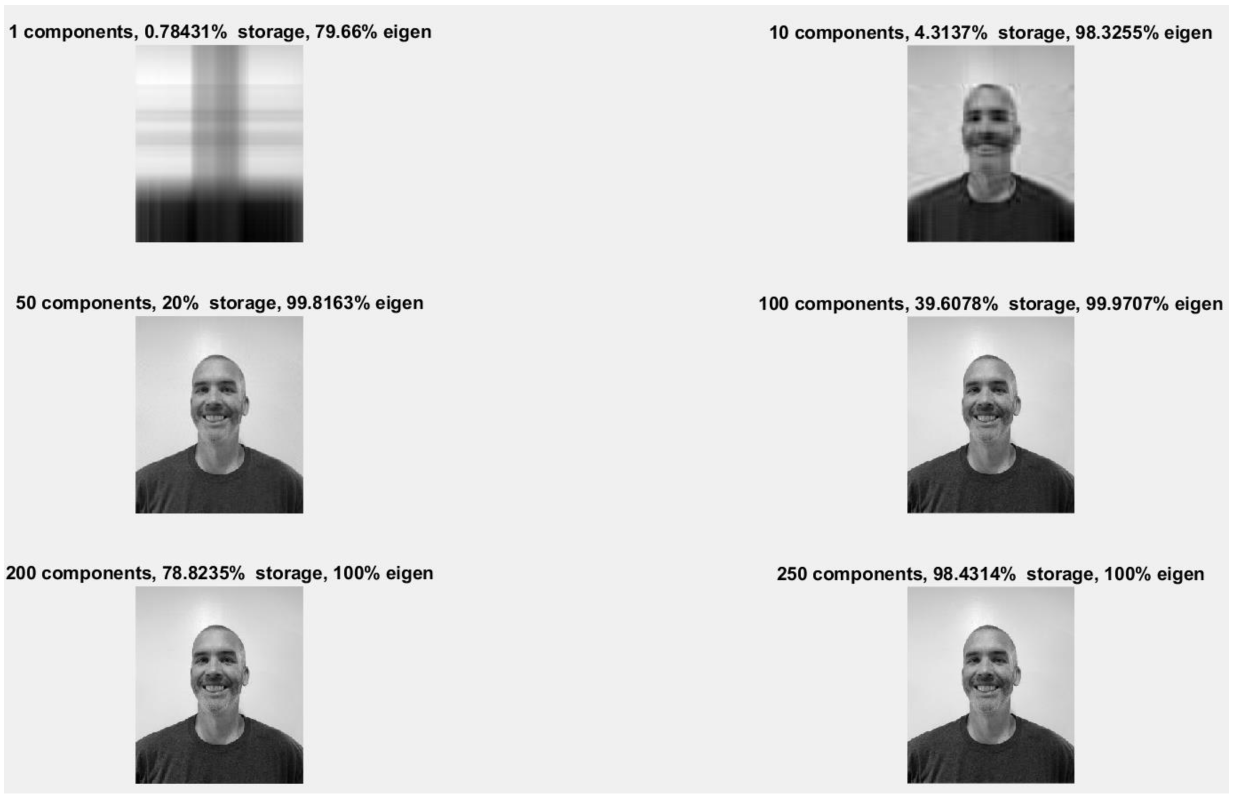

1 components, 0.78431% storage, 79.66% eigen
10 components, 4.3137% storage, 98.3255% eigen
50 components, 20% storage, 99.8163% eigen
100 components, 39.6078% storage, 99.9707% eigen
200 components, 78.8235% storage, 100% eigen
250 components, 98.4314% storage, 100% eigen

Figure 2: Problem 2: Pictures from PCA Analysis

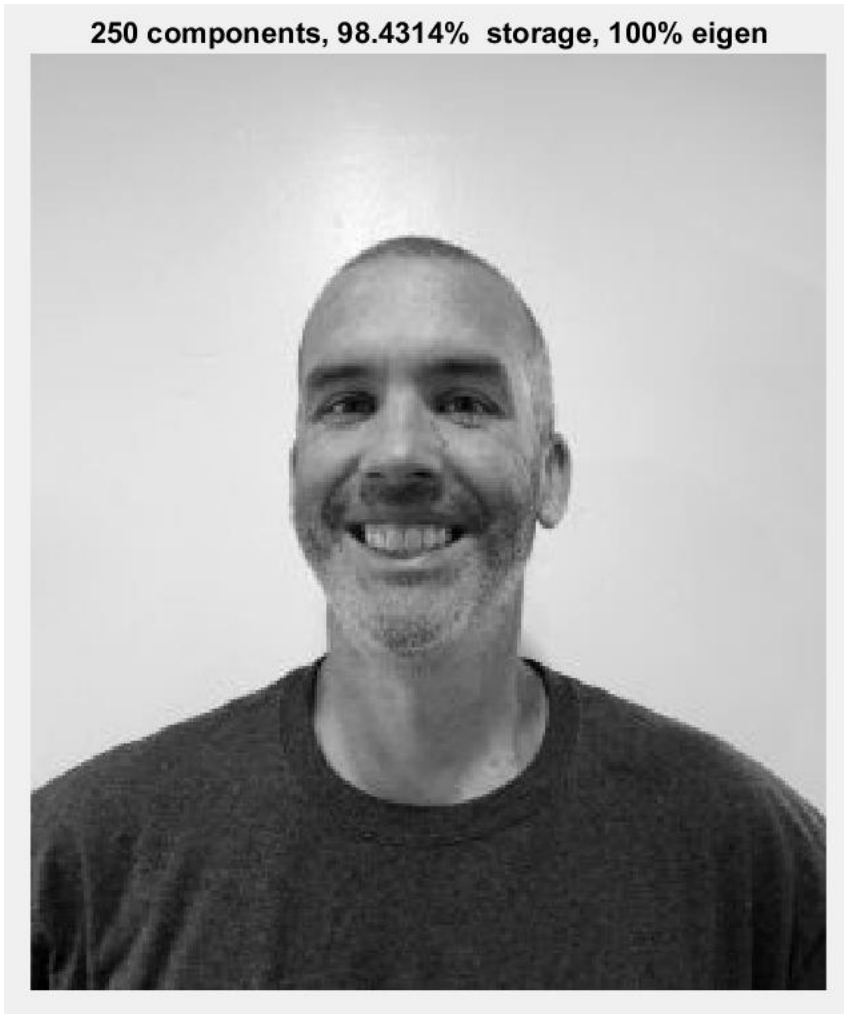


Figure 3: Problem 2: Resulting Image Using Only 250 Principal Components

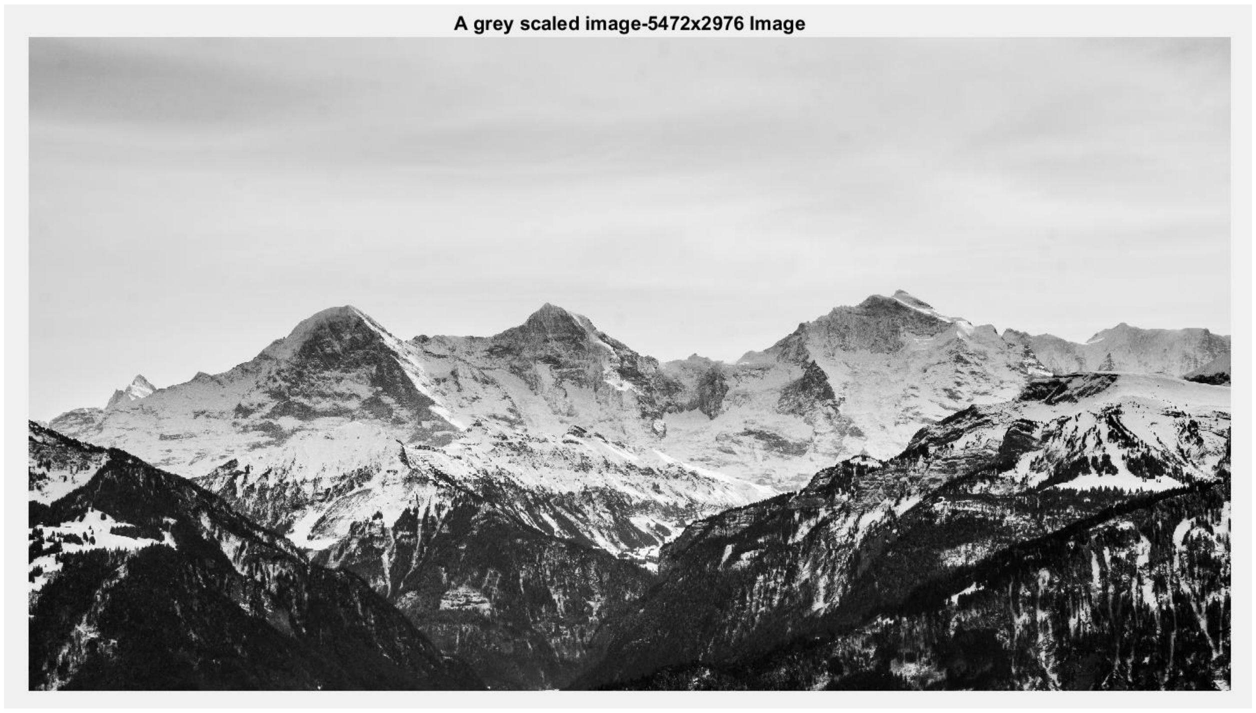


Figure 4: Problem 2: Original Picture

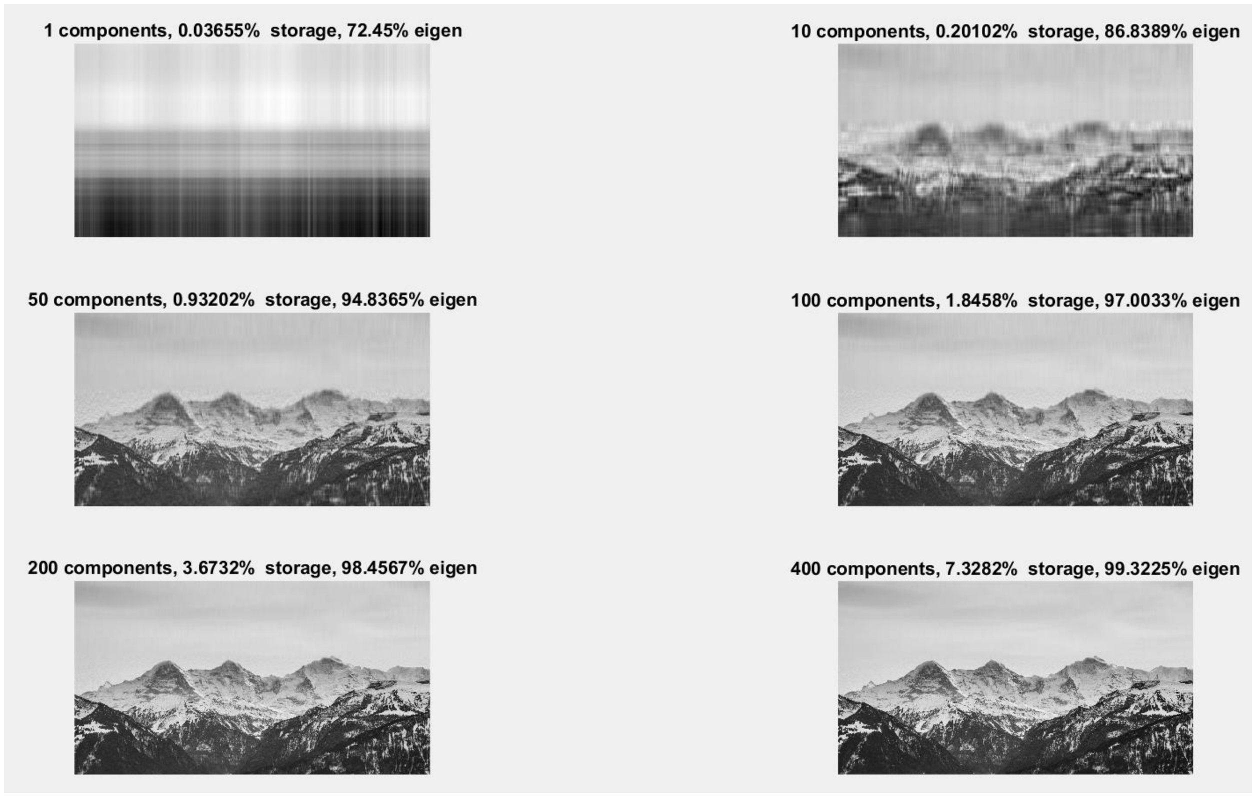

1 components, 0.03655% storage, 72.45% eigen
10 components, 0.20102% storage, 86.8389% eigen
50 components, 0.93202% storage, 94.8365% eigen
100 components, 1.8458% storage, 97.0033% eigen
200 components, 3.6732% storage, 98.4567% eigen
400 components, 7.3282% storage, 99.3225% eigen

Figure 5: Problem 2: Pictures from PCA Analysis

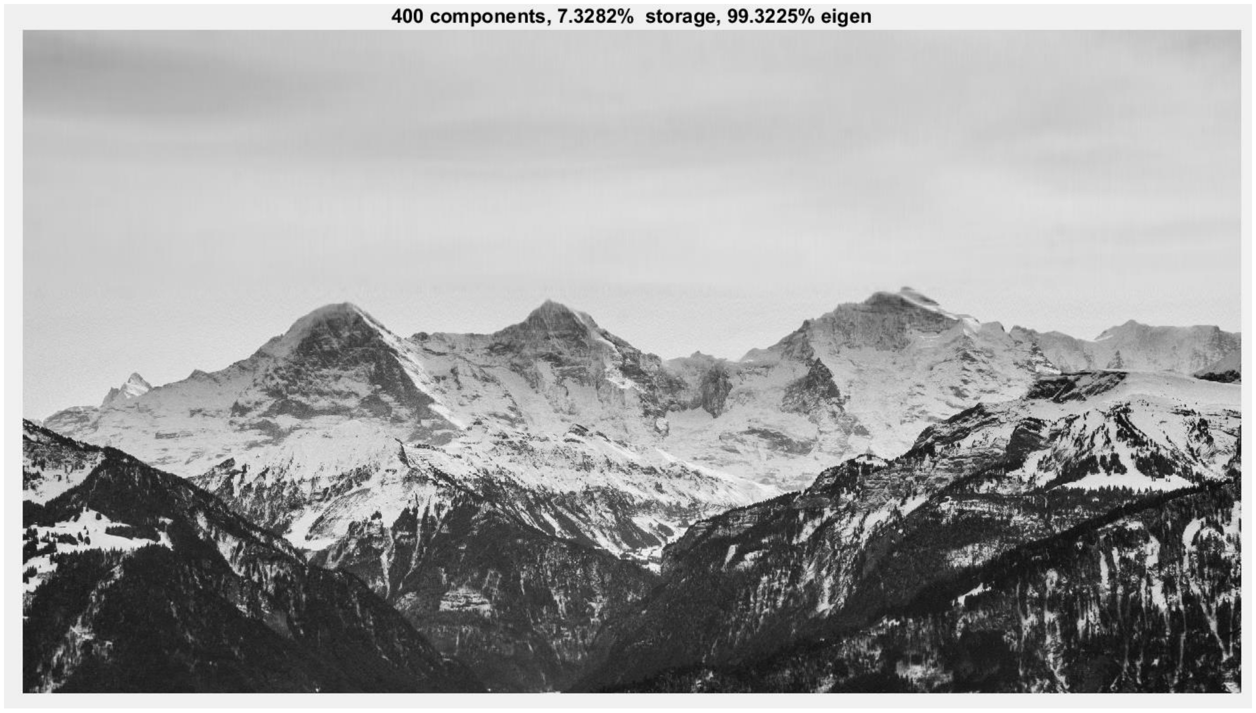


Figure 6: Problem 2: Resulting Image Using Only 400 Principal Components

The image processing procedure outlined below demonstrates the steps taken to extract principal components and recreate an image while retaining only 400 principal components. I believe even 200 components were enough to construct the image with 98.4%.

The steps of reconstruction of the image:

1. Read the image and convert it to Greyscale:

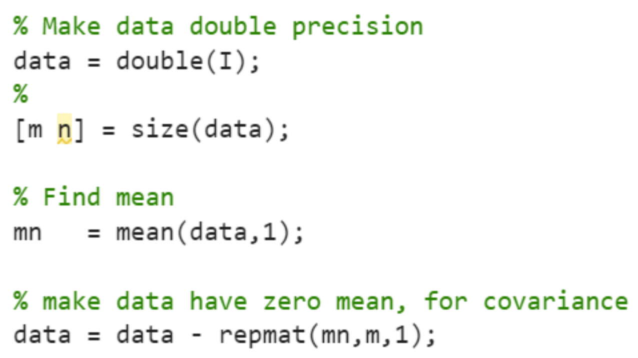
- The FH.jpg image is read using the "imread" function and then converted to greyscale values (ranging from 0 to 255) using the “rgb2gray” function. (The second image is already grey.)

```
% Read in image
I = imread('FH.jpg');
% and convert to grayscale
I = rgb2gray(I);
```

Image Read and Convert to Grayscale Values

2. Center the Data:

- The greyscale matrix is centered by subtracting the mean of the values in each column of the matrix.

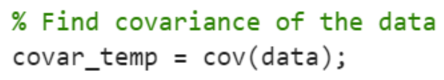


```
% Make data double precision
data = double(I);
%
[m n] = size(data);

% Find mean
mn   = mean(data,1);

% make data have zero mean, for covariance
data = data - repmat(mn,m,1);
```

Centering the Data

3. Covariance Matrix Computation

- The covariance matrix is computed using the “cov” function.

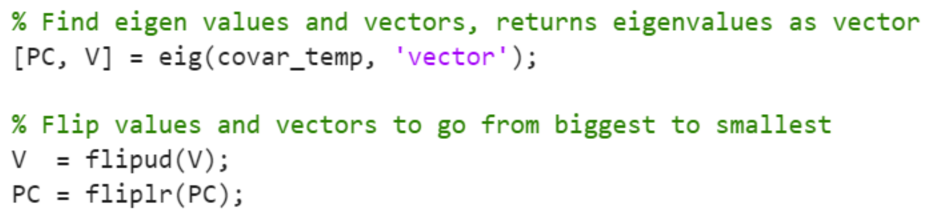


```
% Find covariance of the data
covar_temp = cov(data);
```

Covariance Matrix Computation

4. Find Eigenvectors and Eigenvalues:

- The eigenvectors and eigenvalues of the covariance matrix are calculated using the "eig" function. The resulting vectors are then flipped to have the eigenvectors/values arranged from largest to smallest.



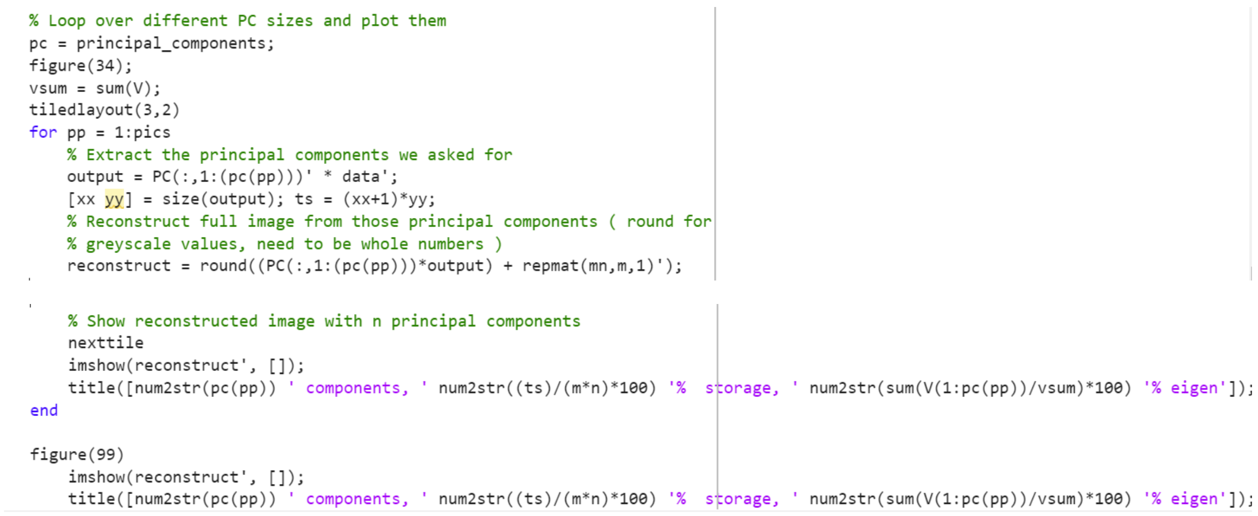
```
% Find eigen values and vectors, returns eigenvalues as vector
[PC, V] = eig(covar_temp, 'vector');

% Flip values and vectors to go from biggest to smallest
V  = flipud(V);
PC = fliplr(PC);
```

Finding Eigenvectors/values of the Covariance Matrix

5. Reconstruct Image with Principal Components:

- The image is reconstructed using different numbers of principal components. Six principal component thresholds [1 10 50 100 200 400] are chosen for analysis.

- The reconstruction involves multiplying the principal component matrix with the number of principal component vectors equal to the chosen threshold. The average is then added back to the reconstructed image, and adjustments are made.



```
% Loop over different PC sizes and plot them
pc = principal_components;
figure(34);
vsum = sum(V);
tiledlayout(3,2)
for pp = 1:pics
    % Extract the principal components we asked for
    output = PC(:,1:(pc(pp)))' * data';
    [xx yy] = size(output); ts = (xx+1)*yy;
    % Reconstruct full image from those principal components ( round for
    % greyscale values, need to be whole numbers )
    reconstruct = round((PC(:,1:(pc(pp)))*output) + repmat(mn,m,1)');

    % Show reconstructed image with n principal components
    nexttile
    imshow(reconstruct', []);
    title([num2str(pc(pp)) ' components, ' num2str((ts)/(m*n)*100) '%  storage, ' num2str(sum(V(1:pc(pp))/vsum)*100) '% eigen']);
end

figure(99)
    imshow(reconstruct', []);
    title([num2str(pc(pp)) ' components, ' num2str((ts)/(m*n)*100) '%  storage, ' num2str(sum(V(1:pc(pp))/vsum)*100) '% eigen']);
```

Image Reconstruction

The resulting images exhibit a gradual increase in clarity. Using only 200 principal components yields an image nearly indistinguishable from the original. However, the next 200 principal components may struggle to accurately depict finer details. The final image utilizes only 400 of the 5470 principal components, accounting for more than 99.3% (image 2) of the sum of all eigenvalues.