Problem 1

The graph in figure 1 shows the connection between four different web pages a, b, c and d. Rank the web pages using the approach based on the Perron-Frobenius eigenvector. Show the matrices used to solve the problem.

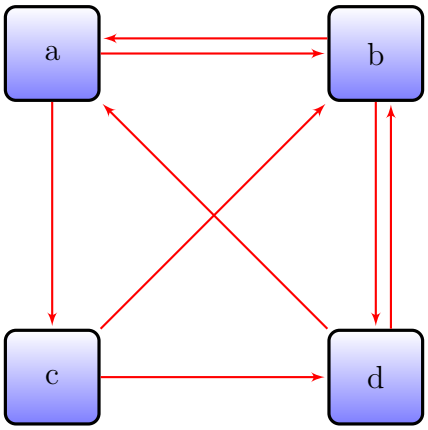

Figure 1: Links between four web pages

The adjacency matrix from the figure above is...

$$A = \begin{bmatrix} 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 \end{bmatrix}$$

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

$$M = dA + \frac{(1-d)}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Where $d = 0.85$. The resulting $\overrightarrow{x}$ is recalculated by being multiplied by M over and over again until the difference in the vector L2-norm is less than 10^{-6} or the iterative process has run more than 10000 times. Displayed in the code below...

```
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;
    x = xnew;
    count = count + 1;
end
```

Problem 1 Iterative Process

After the iterative process is done, $\overrightarrow{x}$ is divided by its resulting L2-norm, providing the answer...

$$\overrightarrow{x} = \begin{bmatrix} 0.5777 \\ 0.0298 \\ 0.3587 \\ 0.5069 \end{bmatrix}$$

This answer can be compared to the first column vector (which corresponds to the highest postivive eigenvalue from Λ, 0.8908, satisfying the Perron-Frobenius Theorem) from the resulting V from Singular Value Decomposition...

$$V_1 = \begin{bmatrix} 0.5777 + 0.0000i \\ 0.5298 + 0.0000i \\ 0.3587 + 0.0000i \\ 0.5069 + 0.0000i \end{bmatrix}$$

V_1 reaffirms $\overrightarrow{x}$ from the iterative process is correct, showing that rank from most to least important webpage would be in the order of a, d, c, and finally b.

Problem 2

Choose an image of your preference and analyze it using PCA. Describe your work step by step and show figures/plots to support your results. You can do either an RGB or greyscale image.

I chose the following greyscaled image of my boyfriend and me on his grandparent's farm:

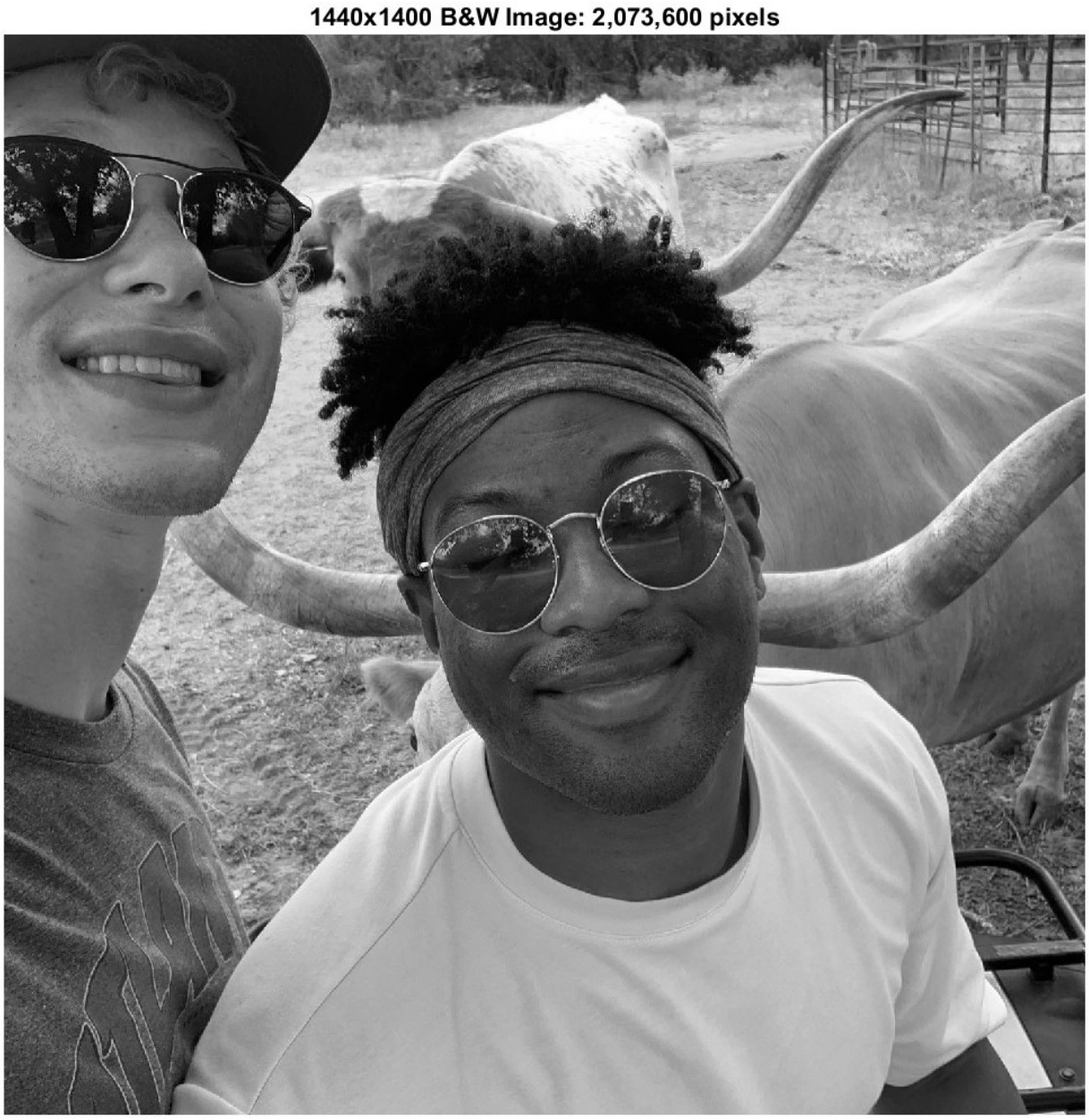

Figure 2: Problem 2: Original Picture

I felt that there was a lot of complexity to this image and wanted to see how many principal components would need to be used to recreate the image without any noticable changes.

After some tinkering with the principal components array, I found that 300 principal com-

ponents created no noticible changes, which will be shown and discussed later.

Step-By-Step Process Applied to the Image

The following steps describe the process applied the the image in order to find the principal components and recreate an indistinguishable image with only 300 principal components...

1. Read the image and convert the image to grey values

   The image is read by the function, *imread*, and converted to grayscale values (ranging from 0 to 255) by the function, *rgb2gray*. The process is displayed by the following code...

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

Read and Convert to Grayscale Values

2. Center the data

   The greyscale matrix is subtracted by 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. Compute the covariance matrix

   This process is done by the function, *cov*

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

Computing Covariance Matrix

4. Find the eigenvectors and eigenvalues of the covariance matrix

   This process is done by the function, *eig*, which is then flipped to have the eigenvectors/values go 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 the image using different numbers of principal components

   For this analysis, I chose six principal component thresholds: $[1, 10, 50, 100, 200, 300]$

   The process for recreating the image by multiplying the principal component matrix with the number of principal component vectors equal to that of the principal component threshold with the average being added back to it and then readjusted is shown below...

```
% 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']);
```

Reconstructing the Image

The resulting images show a gradual increase in clarity. Using only 200 principal components was nearly indistinguishable from the original image, however, the next 100 principal components struggled to accurately depict the stubble on our faces.

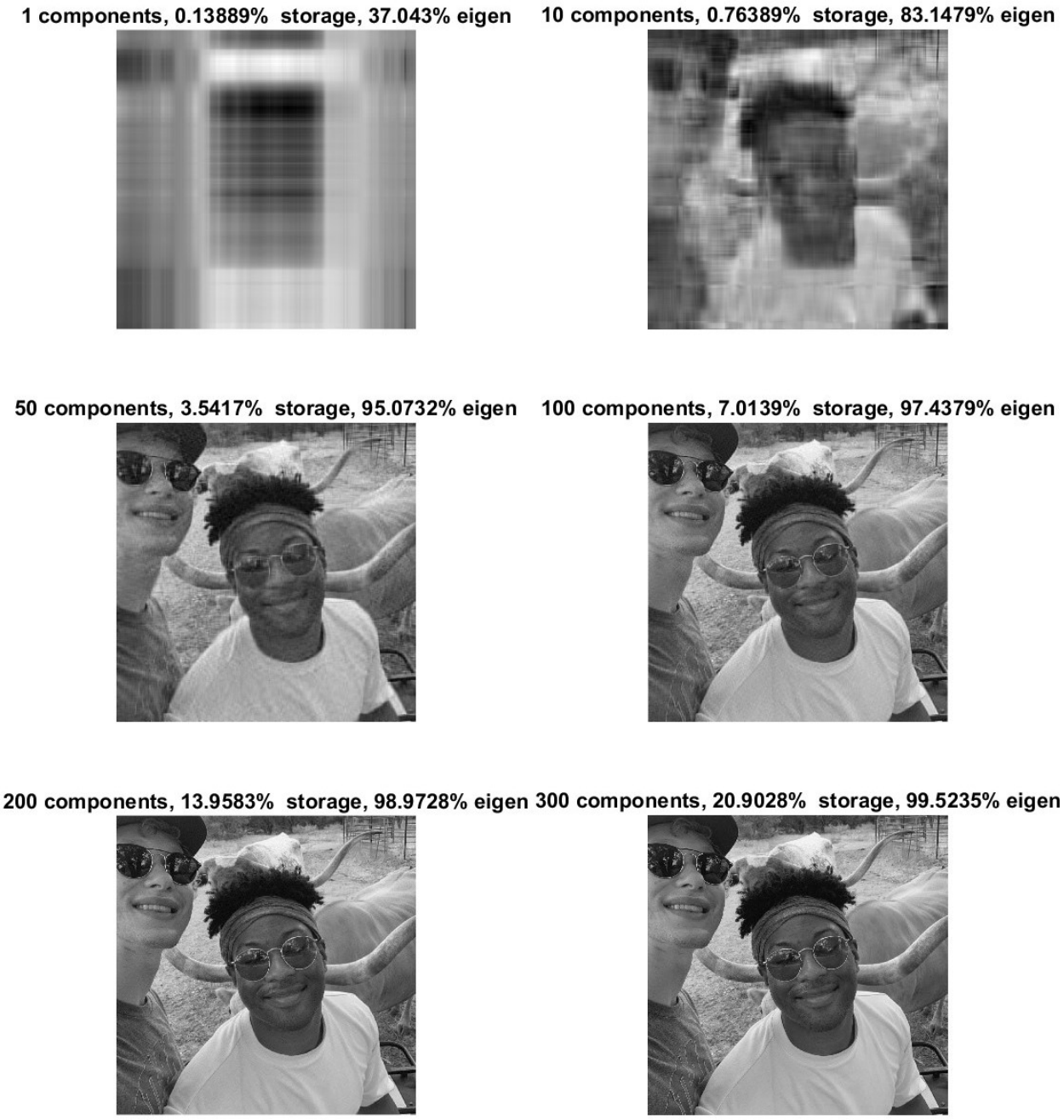

Figure 3: Problem 2: Pictures from PCA Analysis

The final image uses only 300 of the 1440 principal components, however, those factors for more than 99.5% of the sum of all of the eigenvalues.

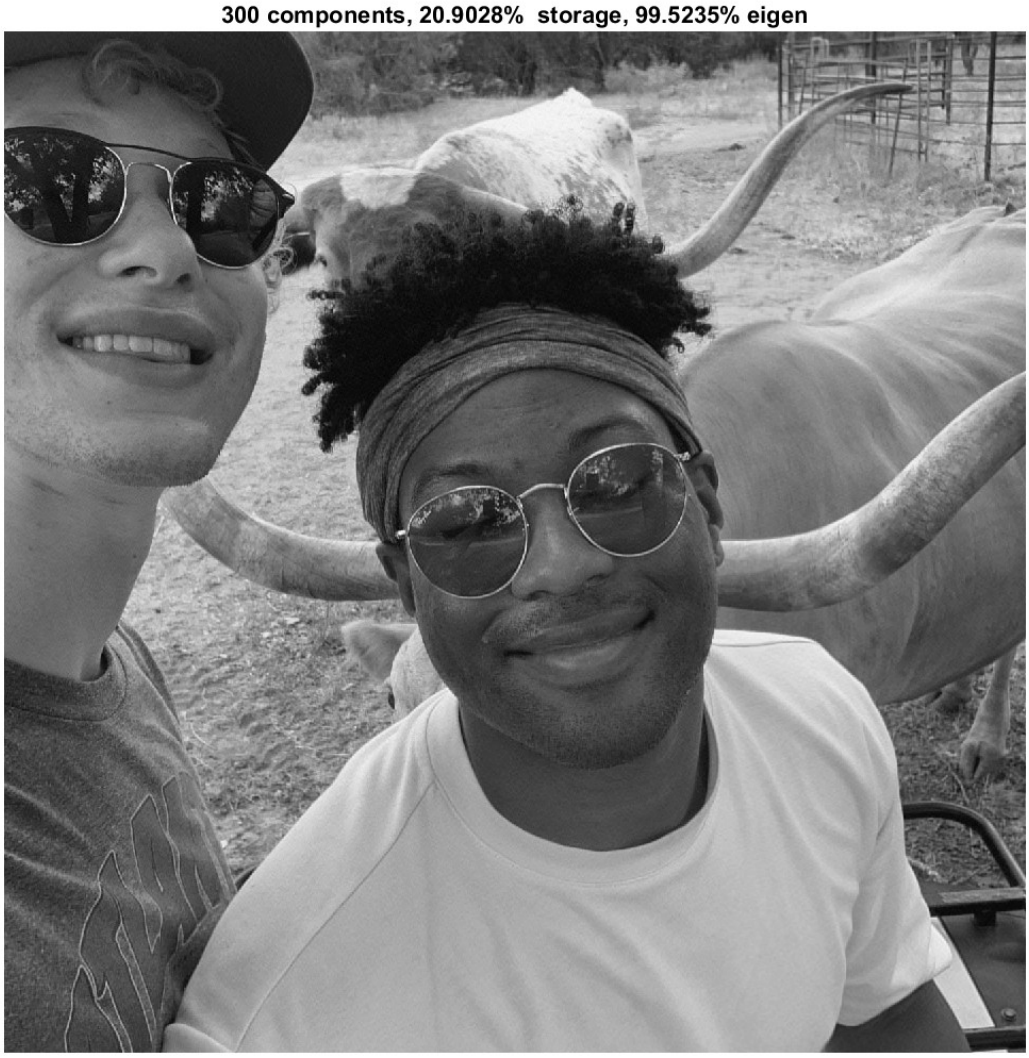

Figure 4: Problem 2: Resulting Image Using Only 300 Principal Components

MATLAB Code

```
%% 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;
    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
```

Problem 1

```
%% Greyscale PCA
%
% Jared Brzenski

close all;
clear all;
clc
% Setup principal components to use. Pick 4.
principal_components = [1 10 50 100 200 300]
pics = length(principal_components);

```

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

% Show it
figure(1)
imshow(I, []);title('1440x1400 B&W Image: 2,073,600 pixels');

% 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);
fprintf('Now data has zero mean: %4.5f\n', (mean(mean(data))) );

% Find covariance of the data
covar_temp = cov(data);
%
% Or do it by hand....
% for y=1:n
%     for in=1:n
%             covar_temp(y,in)=1/((m*n)-1)*(data(:,y)' * data(:,in))  ;
%     end
% end
%
% 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);

% OLD!!!: Rank and order the values and vectors
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Retain only the eigen values ( stored on the diagonal )
%V = diag(V);

% Rank the eigenvalues - backwards from given values
%[holder rank_indices] = sort(-1*V);

% Sort the eigen vectors based on the sorted eigenvalues
%V = V(rank_indices);
%PC = PC(:,rank_indices);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% 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']);
```

Problem 2