Lab 6
April 4, 2016
Vector-Space Signal Processing

INSTRUCTIONS:
All lab submissions include a written report and source code in the form of an m-file. The report contains all plots, images, and figures specified within the lab. All figures should be labeled appropriately. Answers to questions given in the lab document should be answered in the written report. ***The written report must be in PDF format***. Submissions are done electronically through **my.ECE**.

1 Convolution Revisited

If a system satisfies linearity and shift-invariance, its output $y(n)$ can be expressed as a convolution between $x(n)$ and the system impulse response $h(n)$.

$$y(n) = \sum_{m=0}^{N-1} x(m)h(n-m) \tag{1}$$

Since convolution is a linear operation, it can be written as a matrix equation of the form $\mathbf{y} = \mathbf{C} \cdot \mathbf{x}$ where $\mathbf{C}$ is a convolution matrix.

Report Item: Given

$$x(n) = \{1, 4, -4, -3, 2, 5, -6\}$$
$$h(n) = \{1, 2, 3, 4, 5\}$$

Find the convolution between $x(n)$ and $h(n)$ by generating the convolution matrix using **convmtx**. Plot the resulting matrix using **imagesc**. What do you notice about the structure of the **convmtx**? Verify that $\mathbf{C} \cdot \mathbf{x}$ produces the same result as **conv**. Plot the results for both using **stem** on the same plot.

2 The DFT Matrix

As we have learned in Lab 1, the DFT can be expressed as a matrix equation.

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-i2\pi kn/N} \longrightarrow \mathbf{X} = \mathbf{F} \cdot \mathbf{x}$$

where the elements of $\mathbf{F}$ are

$$[\mathbf{F}]_{kn} = e^{-i2\pi kn/N} \tag{2}$$

Furthermore, it can be shown that the IDFT is just $\frac{1}{N}\mathbf{F}^\dagger$ where $\dagger$ is the conjugate transpose of a matrix. Naturally, it can be shown that $\mathbf{I} = \frac{1}{N}\mathbf{F}^\dagger \cdot \mathbf{F}$ where $\mathbf{I}$ is the identity matrix. Provided that $\mathbf{F}$ is square, it can be shown that

$$\frac{1}{\sqrt{N}}\mathbf{F}^\dagger = \frac{1}{\sqrt{N}}\mathbf{F}^{-1} \tag{3}$$

Matrices which satisfy are unitary.

Report Item: Given

$$x(n) = \{1, 4, -4, -3, 2, 5, -6\}$$

Generate the DFT matrix $\mathbf{F}$ corresponding to an input $\mathbf{x}$ using **dftmtx**. Plot the real and imaginary parts of this matrix using **subplot** and **imagesc**. Comment on what you see. In particular, what waveform do you notice in each row and how does the frequency vary with higher row numbers? Use the term *dot product* to describe the relationship between DFT coefficients and $\mathbf{x}$.

The DFT can compute circular convolution in the frequency domain, which is cheaper than doing so in the time domain when using FFT. The circular convolution matrix $\tilde{\mathbf{C}}$ is slightly different compared to the convolution matrix $\mathbf{C}$. It can be shown that

$$\tilde{\mathbf{C}} \cdot \mathbf{x} = \frac{1}{N}\mathbf{F}^\dagger \cdot \text{diag}(\mathbf{F} \cdot \mathbf{h}) \cdot \mathbf{F} \cdot \mathbf{x}$$

Therefore,

$$\tilde{\mathbf{C}} = \frac{1}{N}\mathbf{F}^\dagger \cdot \text{diag}(\mathbf{F} \cdot \mathbf{h}) \cdot \mathbf{F}$$

Report Item: Given

$$x(n) = \{1, 4, -4, -3, 2, 5, -6\}$$
$$h(n) = \{1, 2, 3, 4, 5\}$$

Find the *circular* convolution between $x(n)$ and $h(n)$ by generating the circular convolution matrix $\tilde{\mathbf{C}}$ using **cconvmtx**. Plot the resulting matrix using **imagesc**. What differences do you notice between $\mathbf{C}$ and $\tilde{\mathbf{C}}$? Given that $x(n)$ is length L and $h(n)$ is length M, express the number of rows and columns in $\tilde{\mathbf{C}}$ in terms of L and M. Verify that $\tilde{\mathbf{C}} = \frac{1}{N}\mathbf{F}^\dagger \cdot \text{diag}(\mathbf{F} \cdot \mathbf{h}) \cdot \mathbf{F}$ (*HINT: Remember that x and h must be zero-padded to a particular length in order to perform circular convolution using DFT*).

3 Singular-Value Decomposition

The singular-value decomposition (SVD) is a matrix factorization of the form

$$\mathbf{A}_{M\times N} = \mathbf{U}_{M\times M} \cdot \mathbf{\Sigma}_{M\times N} \cdot \mathbf{V}^{\dagger}_{N\times N} \tag{4}$$

where $\mathbf{A}$ is an arbitrary $M \times N$ matrix and $\mathbf{U}$, $\mathbf{V}$ and Σ represent a canonical singular value decomposition. The matrices $\mathbf{U}$ and $\mathbf{V}$ are orthonormal, meaning they satisfy the property

$$\mathbf{U}^{\dagger} \cdot \mathbf{U} = \mathbf{U} \cdot \mathbf{U}^{\dagger} = \mathbf{I} \tag{5}$$

which also implies that for square $\mathbf{U}$,

$$\mathbf{U}^{-1} = \mathbf{U}^{\dagger} \tag{6}$$

The Σ matrix is a diagonal (but not necessarily square) matrix which contains the singular values of $\mathbf{A}$. The singular values are related to the eigenvalues of $\mathbf{A}$. This can easily be shown by first stating the definition of eigenvectors. Given a matrix $\mathbf{A}$, the eigenvector $\mathbf{x}_n$ and its associated eigenvalue λ_n form the relationship shown by (7).

$$\mathbf{A} \cdot \mathbf{x}_n = \lambda_n \mathbf{x}_n \tag{7}$$

Since every $N \times N$ matrix has N eigenvectors, we can write a matrix equation that encompasses all N eigenvalues/eigenctors as

$$\mathbf{A} \cdot \mathbf{X} = \mathbf{X} \cdot \mathbf{D} \tag{8}$$

where $\mathbf{D}$ is a diagonal matrix of eigenvalues and $\mathbf{X}$ is an eigenvector matrix where each column is an eigenvector.

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \dots & \mathbf{x}_N \end{bmatrix} \tag{9}$$

Taking (4) and multiplying it with its conjugate transpose,

$$\mathbf{A} \cdot \mathbf{A}^{\dagger} = \mathbf{U} \cdot \mathbf{\Sigma} \cdot \mathbf{V}^{\dagger} \cdot \mathbf{V} \cdot \mathbf{\Sigma}^{\dagger} \cdot \mathbf{U}^{\dagger} \tag{10}$$

Using the fact that $\mathbf{U}$ is orthonormal,

$$\mathbf{A} \cdot \mathbf{A}^{\dagger} = \mathbf{U} \cdot \mathbf{\Sigma} \cdot \mathbf{\Sigma}^{\dagger} \cdot \mathbf{U}^{\dagger} \tag{11}$$

Multiplying both sides by $\mathbf{U}$ yields

$$\mathbf{A} \cdot \mathbf{A}^{\dagger} \cdot \mathbf{U} = \mathbf{U} \cdot \mathbf{\Sigma} \cdot \mathbf{\Sigma}^{\dagger} \tag{12}$$

Comparing (12) and (8), one can see that $\mathbf{U}$ is the eigenvector matrix of $\mathbf{A} \cdot \mathbf{A}^{\dagger}$ and $\mathbf{\Sigma} \cdot \mathbf{\Sigma}^{\dagger}$. Using a similar procedure, it can be shown that $\mathbf{V}$ and $\mathbf{\Sigma}^{\dagger} \cdot \mathbf{\Sigma}$ are the eigenvector and eigenvalue matrices for $\mathbf{A}^{\dagger} \cdot \mathbf{A}$.

Report Item: Given

$$\mathbf{A} = \begin{bmatrix} 1 & 4 & -2 \\ 3 & 11 & 5 \\ 7 & 7 & 7 \end{bmatrix}$$

Determine the eigenvalues and eigenvector of $\mathbf{A} \cdot \mathbf{A}^\dagger$ and $\mathbf{A}^\dagger \cdot \mathbf{A}$ using **eig**. Then, take the singular value decomposition of $\mathbf{A}$ using **svd** and verify that $\mathbf{U}$ are the eigenvectors of $\mathbf{A} \cdot \mathbf{A}^\dagger$ and $\mathbf{V}$ are the eigenvectors of $\mathbf{A}^\dagger \cdot \mathbf{A}$.

4 Principal Component Analysis

Principal component analysis (PCA) is the projection of data onto a subspace where the subspace can be defined by a set of basis vectors. In PCA, the eigenvectors of a viable set of basis vectors are referred to as principal components. If the eigenvalue associated with a principal component is small, then it can be omitted without having a large effect on the data projected onto it. Keeping important components (ones associated with larger eigenvalues) reduces the computation time of image recognition algorithms based on PCA while preserving important features of the subspace. The principal components can be derived using SVD.

Assume that an $M \times N$ matrix $\mathbf{X}$ contains M row vectors of length N such that

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_M^T \end{bmatrix} \tag{13}$$

where $\mathbf{x}_m^T$ are row vectors (or the transpose of column vectors) which may represent voltages measured from acquisitions on an oscilloscope, an image, or any set of measured numbers. We consider each row in $\mathbf{X}$ to be a sample. The sample mean is a vector which can be defined as

$$\boldsymbol{\mu}_X = \frac{1}{M} \sum_{n=1}^{M} \mathbf{x}_n^T \tag{14}$$

From (14), we can define a matrix $\hat{\mathbf{X}} = \mathbf{X} - \text{ones}(M, 1) \cdot \boldsymbol{\mu}_X$ that subtracts the sample mean from each row vector in $\mathbf{X}$. The result is a set of row vectors in $\hat{\mathbf{X}}$ whose origin is at $\boldsymbol{\mu}_X$. Subtracting the mean permits us to write a simple expression for the co-variance matrix of $\hat{\mathbf{X}}$.

The co-variance matrix $\mathbf{R}$ can be written as

$$\mathbf{R} = \hat{\mathbf{X}}^\dagger \cdot \hat{\mathbf{X}} \tag{15}$$

Note that the co-variance matrix is necessarily square. Using SVD, the eigenvectors of the co-variance matrix can be expressed as

$$\mathbf{R} = \mathbf{V} \cdot \mathbf{D} \cdot \mathbf{V}^\dagger \tag{16}$$

or equivalently

$$\mathbf{R} \cdot \mathbf{V} = \mathbf{V} \cdot \mathbf{D} \tag{17}$$

where $\mathbf{V}$ is the eigenvector matrix of $\mathbf{R}$ and $\mathbf{D}$ is a diagonal matrix of singular values. The PCA coefficients can be determined via projection. The PCA matrix can be expressed as

$$\mathbf{Z} = \hat{\mathbf{X}} \cdot \mathbf{V} \tag{18}$$

The inverse transform can be derived from orthogonality as

$$\hat{\mathbf{X}} = \mathbf{Z} \cdot \mathbf{V}^{\dagger} \tag{19}$$

5 Image Recognition

PCA can be used to perform image recognition. First, the principal components are extracted from a set of training images. In the *training_images* folder are a set of training images. Use **dir** to extract the file names of every image in the folder and read each image using **imread**. Load the images into a cell array called *faces*. Use the provided **imresize** function to resize each image to $M \times N$ prior to storage in the cell array.

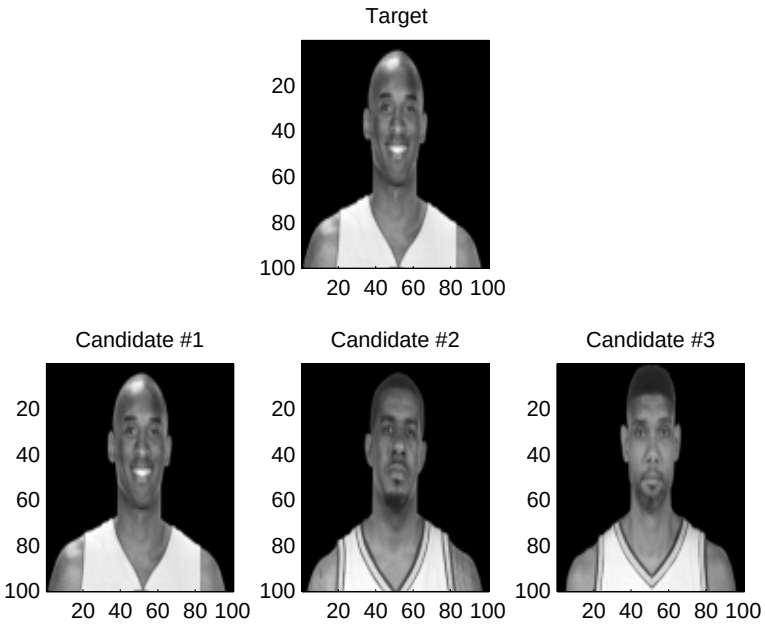

Figure 1: The top 3 candidates for the target image are shown for $K = 21$.

Create a function called *eigenfaces* which will extract the eigenfaces of the image. The input to this function are *faces* and K where K is the number of principal components to keep. The output of *eigenfaces* is *avgface*, and *eigfaces*. *avgface* is the average of all the faces stored in *faces*. *eigface* is an $M \times N \times K$ matrix that

stores the first K principal components in the face matrix $\mathbf{X}$. To create the face matrix $\mathbf{X}$, store each face in *faces* as a row in $\mathbf{X}$. To do this, each face needs to be reshaped from a 2-D matrix into a 1-D vector. This can be done using **reshape**. The face matrix $\mathbf{X}$ should have rows equal to the number of faces and columns equal to MN. Once this is done, subtract the mean face from each row in the face matrix to create $\hat{\mathbf{X}}$. Construct the co-variance matrix using (15). Then, using **svds**, the first K principal components can be found. Store these components in *eigfaces*.

Report Item: From the set of training images, use the *eigenfaces* function to generate the *averageface* and *eigenfaces* matrix. Use **imresize** to resize the images to 100×100. Plot the average face and the first 4 eigenfaces using **imagesc** with **colormap('gray')**. To read the images from the training set, you should use **rgb2gray** on the image returned from **imread** to obtain a black and white image. Additionally, plot the singular values of the co-variance matrix. What is a good value for K that can be used to truncate the number of eigenfaces?

Create a function called *projectface* whose inputs are *avgface*, *eigfaces*, and *face*. The output of this function is a vector *coeffs* of length K. The purpose of this function is to compute the projection of an image onto the subspace defined by *eigfaces*. Create a function called *constructface*. The inputs to this function are *avgface*, *eigfaces*, and *coeffs*. This function uses *coeffs* to construct a face from the subspace defined by *eigfaces*. If the image resemble the training images, then the *coeffs* derived from its projection onto *eigenfaces* will have some large values. Create another function called *comparefaces* which accepts *avgface*, *eigfaces*, *face1*, *face2*, and returns *mse*. This function should compute the coefficients of *face1* and *face2* and compute the mean squared error (MSE) between their projection coefficients. The equation for MSE is

$$\text{MSE} = \frac{1}{K}\sum_{k=1}^{K}(c_k - \hat{c}_k)^2 \tag{20}$$

where K is the length of $\mathbf{c}$ and $\hat{\mathbf{c}}$.

Finally, construct a function called *recognizeface* which accepts *avgface*, *eigfaces*, *usercoeffs*, *face*, and returns a vector called *order*. *usercoeffs* is a matrix where each row is a set of coefficients from a different image. This function uses *constructface* to construct a face from each row of coefficients in *usercoeffs* and compares the constructed face to *face* using *comparefaces*. The MSE returned from *comparefaces* can be stored row-wise in a matrix along with the index of the image. Using **sortrows**, the matrix can be sorted by MSE in ascending order where a small MSE indicates a good match. The order of the images which closely match *face* are stored in the second column of *order*. After applying **sortrows** to this matrix, the indices of the images in ascending MSE order can be found in the second column.

Report Item: Create a matrix of PCA coefficients from the training images (resized to 100×100. Read *face1.png* using **imread** and use *recognizeimage* to find the 3 closest matches. Use *subplot* to show *face1.png* along with the 3 images that match closely to it. What is the minimum K you need to use? Repeat for *face2.png*.