EECS 16B Designing Information Devices and Systems II
Spring 2016 Anant Sahai and Michel Maharbiz

# Homework 3

**This homework is due February 15, 2016, at Noon.**

1. **Homework process and study group**

   Who else did you work with on this homework? List names and student ID's. (In case of hw party, you can also just describe the group.) How did you work on this homework?

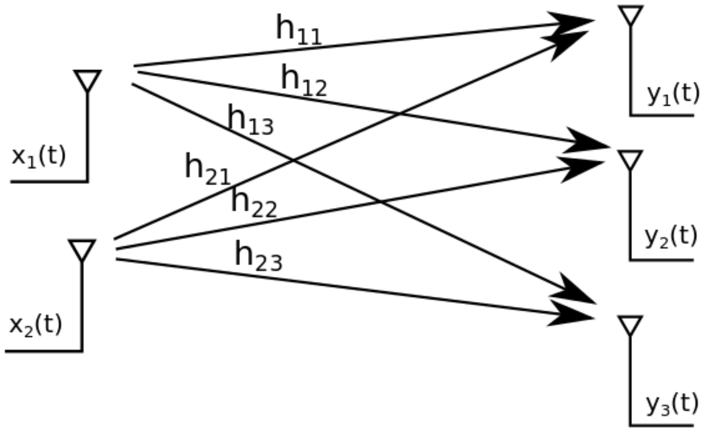
2. **MIMO wireless signals**

   Ever wonder why newer wifi routers and cellular base stations have 4 or sometimes even more antennas on them? New wireless technologies actually use multiple antennas that each send their own signal on the same frequency band. The key here is not only do we encode signals in frequency bands, but also in spatial ones using a technique known as "Spatial Multiplexing".

   We call this idea "MIMO" wireless, which stands for "multiple input multiple output". This technique is used in many standards including 802.11n/ac, 4G LTE, and WiMAX.

   In this problem, we will explore how signals are decoded on the output end.

   Consider the following:



   We have 2 transmit antennas and 3 receive antennas, each receive antenna gets some signal from each of the transmit antennas. We can model the input output relation of the system as follows:

$$\begin{bmatrix} h_{11} & h_{21} \\ h_{12} & h_{22} \\ h_{13} & h_{23} \end{bmatrix} \cdot \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} = \begin{bmatrix} y_1(t) \\ y_2(t) \\ y_3(t) \end{bmatrix}$$

or

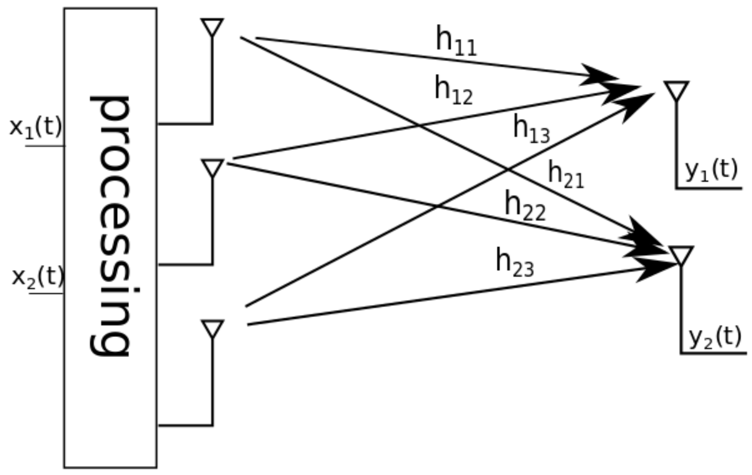
$$H\vec{x}(t) = \vec{y}(t)$$

Here, $H$ is the spatial-response matrix and it acts on the signals instantaneously at each time. For the purpose of this problem, we are going to pretend that there are no echoes across time.

(a) With our new MIMO wireless system, we want to recover the original $\vec{x}(t)$ signal after receiving the $\vec{y}(t)$. In order to do this, we will left multiply $\vec{y}(t)$ by some matrix $A$; ideally we should then exactly recover $\vec{x}(t)$ ($A\vec{y}(t) = \vec{x}(t)$). Using the SVD to decompose $H$, analytically write down what this matrix $A$ should be.

(b) There might have been some noise on each receive antenna. So how is the solution you found in part (a) related to the least squares solution of this problem?

(c) What we just did is referred to as "post processing" or "post coding", and involves the receive end having more antennas than the send side. Many times this is not the case (eg. a wireless cell tower having many more antennas than a phone). What if we wanted to send 2 streams on 3 antennas and receive precisely those 2 streams back on the other end?



The channel is very similar to the one we had made in part (a). In fact, the original channel modelled with spatial response matrix $H$ is precisely the transpose of this channel! Thus, we can say the spatial response matrix for this channel, let's call it $H'$, is simply the following:

$$H' = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \end{bmatrix} = H^T$$

Using the SVD of $H$ and its relation to $H'$, show how you can pre-process $x_1(t)$ and $x_2(t)$ so that you recover them precisely after they have been transmitted across the channel. To be more explicit, after the processing and transmission has been done $y_1(t) = x_1(t), y_2(t) = x_2(t)$.

(d) (Optional) Why do you think that we are using the SVD here? Is there a unique solution of what to transmit that will achieve the desired goal in the previous part? Why choose this approach?

3. **Hermitian Matrices** A Hemitian matrix $T$ is a square matrix, which is equal to its conjugate transpose, i.e. $T^* = T$. Hermitian matrices have nice properties! In this problem, we will walk through one of them. Let $\lambda$ be an eigenvalue of $T$ with associated eigenvector $\vec{v}$. That is to say,

$$T\vec{v} = \lambda\vec{v} \tag{1}$$

(a) Multiply both sides of Equation (1) with $\vec{v}^*$. Call this Equation (2). *(Hint: Where should we put $\vec{v}^*$ at each side of the equation? The right or left side of the original formula?)*

(b) Compute the conjugate transpose of each side of Equation (1). What do you get? Simplify and call this Equation (3). *(Hint: use the fact that $T^* = T$ )*

(c) Multiply both sides of Equation (3) with $\vec{v}$. Let that be Equation (4).

(d) Compare Equations (2) and (4). What can you conclude about the eigenvalues of Hermitian matrices?

## 4. Symmetric Matrices

A real symmetric matrix is an important special case of Hermitian matrices. Here we want to show every real symmetric matrix is diagonalized by a matrix of its orthonormal eigenvectors. In other words, a symmetric matrix has full complement of eigenvectors that are all orthogonal to each other.

In discussion section, you have seen a recursive derivation of this fact. Formally however, such recursive derivations are usually turned into proofs by using induction. This problem serves to both freshen your mind regarding induction as well as to give you a chance to prove for yourself this very important theorem.

(a) You will start by proving a basic lemma about real symmetric matrices under an orthonormal change of basis. Prove that if $S$ is a symmetric matrix $S = S^T$ that if $U$ is a matrix whose columns are orthonormal, then $S$ in the basis $U$, namely $U^T SU$ is also symmetric.

(b) Another useful lemma is that real symmetric matrices have real eigenvalues. Argue why this is true. In lecture, this was proved by leaning on the fact that real symmetic matrices can be diagonalized by an orthonormal basis. However, since that is what we are trying to prove here, it is good to have an alternative proof. *(Hint: Does another problem on the homework help us here?)*

(c) A third useful lemma is one about finding orthonormal bases. Show that given a nonzero vector $\vec{u}_0$ of length $n$, that it is possible to find an orthonormal set of $n$ vectors, $\vec{v}_0, \ldots, \vec{v}_{n-1}$ such that $\vec{v}_0 = \alpha \vec{u}_0$ for some scalar $\alpha$.

*(Hint: Use the Gram-Schmidt process somehow.)*

(d) For the main proof, we will proceed by formal induction. Recall that for a proof by induction, we have to start with a base case - this is also the base case in a recursive derivation.

Consider the trivial case of $S$ having dimensions $[1 \times 1]$ ($n = 1$). Does $S$ have an eigenvector? Can this eigenbasis be made orthonormal? Is the matrix diagonal in this basis? Are the entries real?

(e) After the base case, we do an inductive stage. The first step in the inductive stage is to write down the induction hypothesis. Assume this property holds for all symmetric matrices with size $[(n-1) \times (n-1)]$. Write down the statement of this fact in your own words using mathematical notation. *(Hint: In general for proofs by induction, you want to start with the strongest version of what you want to prove. This gives you the most powerful inductive hypothesis.)*

(f) Now think about a symmetric matrix $S$ with size $[n \times n]$: consider a real eigenvalue $\lambda_0$ of $S$, and the corresponding eigenvector $\vec{u}_0$ (a column vector with size $n$). Use an appropriate orthonormal change of basis $V$ to show that $S = V \begin{bmatrix} \lambda_0 & \cdots & \cdot \\ \vec{0} & \ddots & \cdot \end{bmatrix} V^T$.

(g) Continue the previous part to show that actually

$$S = V \begin{bmatrix} \lambda_0 & \vec{0}^T \\ \vec{0} & Q \end{bmatrix} V^T$$

where $Q$ is an $n-1$ by $n-1$ symmetric matrix.

(h) According to our induction hypothesis, we can write $Q$ as $U\Lambda U^T$ where $U$ is an orthonormal $n-1$ sized square matrix and $\Lambda$ is a diagonal matrix filled with real entries. Use this fact to show that indeed there must exist an orthonormal $n$-sized square matrix $W$ so

$$S = W \begin{bmatrix} \lambda_0 & \vec{0}^T \\ \vec{0} & \Lambda \end{bmatrix} W^T$$

*(Hint: What is the product of orthonormal matrices?)*

By induction, we are now done since we have proved that having the desired property for $n-1$ implies that we have the property for $n$ and we also have a valid base case at $n = 1$.

According to the base and inductive steps we just proved, the statement, "every real symmetric matrix is diagonalized by a matrix of its orthonormal eigenvectors" is proved by induction.

**5. Eigenfaces**

In this problem, we will be be exploring the use of PCA to compress and visualize pictures of human faces. We use the images from the data set Labeled Faces in the Wild. Specifically, we use the set with all images aligned using deep funneling to ensure that the faces are centered in each photo. Each image is a 250x250 image with the face aligned in the center. To turn the image into a vector, we stack each column of pixels in the image on top of each other, and we normalize each pixel value to be between 0 and 1. Thus, a single image of a face is represented by a 625,000 dimensional vector, but a vector this size is a bit challenging to work with directly. We combine the vectors from each image into a single matrix and run PCA on it. For this problem, we will provide you with the first 250 principal components, but you can explore how well the images are compressed with fewer components. Please refer to the IPython notebook to answer the following questions.

(a) We provide you with a set of faces from the training set and compress them using the first 100 principal components. You can adjust the number of principal components used to do the compression. What changes do you see in the compressed images when you used a small number of components and what changes do you see when you use a large number?

(b) You can visualize each principal component to see what each dimension “adds” to the high-dimensional image. What visual differences do you see in the first few components compared to the last few components?

(c) By using PCA on the face images, we obtain orthogonal vectors that point in directions of high variance in the original images. We can use these vectors to transform the data into a lower dimensional space and plot the data points. In the notebook, we provide you with code to plot a subset of 400 images using the first two principal comonents. Try plotting other components of the data, and see how the shape of the points change. What difference do you see in the plot when you use the first two principal components compared with the last two principal components? What do you think is the cause of this difference?

(d) We can use the principal components to generate new faces randomly. We accomplish this by picking a random point in the low-dimensional space and then multiplying it by the matrix of principal components. In the notebook, we provide you with code to generate faces using the first 250 principal components. You can adjust the number of components used. How does this affect the resulting images?

6. **Image Processing by Clustering** In this homework problem, you will learn how to use the k-means algorithm to solve two image processing problems: (1) color quantization and (2) image segmentation.

Digital images are composed of pixels (you could think of pixels as small points shown on your screen.) Each pixel is a data point (sample) of an original image. The intensity of each pixel is its feature. If we use 8-bits integer $Var_i$ to present the intensity of one pixel, $(Var_i = 0)$ means black, while $(Var_i = 255)$ means white. Images expressed only by pixel intensities are called *grayscale* images.

In color image systems, the color of a pixel is typically represented by three component intensities (features) such as Red, Green, and Blue. The features of one pixel is a list of three 8-bit integers:$[Var_r, Var_g, Var_b]$. Here $[0,0,0]$ means black, while $[255,255,255]$ presents white. You could play with this website, http:/www.colorschemer.com/online.html, to see how values of $[Var_r, Var_g, Var_b]$ influence the color.

Now think about your own experience: Have you needed to delete some photos on your cell phones because you ran out of the memory? Have you felt "why does it take forever to upload my photos"? We need ways to reduce the memory size of each photo by image compression.

In computer graphics, there are two types of image compression techniques: lossless and lossy image compression. The former one is preferred for archival purposes, which aim at maintaining all features but reducing the required memory of one image, while the latter one tries to remove some irrelevant and redundant features without destroying the main features of this image. In this problem, you will learn how to use the k-means algorithm to perform lossy image compression by color quantization.

Color quantization is one way to reduce the memory size of an image. In real schemes, it complements the DCT/DFT/Haar-based techniques you saw on the last homework, and is applied there to the coefficients in that basis, but here we do it directly to pixels for simplicity. The target of color quantization is to to reduce the number of colors used in an image, while trying to make the new image visually similar to the original image. An image stored in Graphics Interchange Format (GIF) is an example.

For example, consider an image of the size $800 \times 600$, where each pixel takes 24 bits (3 bytes) to store its color intensities. This raw image takes $800 \times 600 \times 3$ bytes, about 1.4MB to store. If we use only 8 colors to represent all pixels in this image, we could include a color map, which stores the RBG values for these representation colors, and then use 3 bits for each pixel to indicate which one is its representation color. In that case, the compressed image will take $8 \times 24 + 800 \times 600 \times 3$ bits, about 0.2MB, to store it. We save memory.

There are two main tasks in color quantization: (1) decide the representation colors, and (2) determine which representation color each pixel should be assigned. Both tasks can be done by the k-means algorithm: It groups data points (pixels) into $k$ different clusters (representation colors), and the centroids of the clusters will be the representation colors for those pixels inside the cluster.

Here is another thing we can do with this flow: image segmentation. Image segmentation partitions a digital image into multiple segments (sets of pixels.) It is typically used to locate boundaries of objects in images. Once we isolate objects from images, we can perform object detection and recognization, which play essential roles in artificial intelligence. This can be done by clustering pixels with similar features and labeling them by an indicating color of each cluster (object).

(a) Please look at the ipython notebook file, there is a 4 by 4 grayscale image. Perform the k-means algorithm on the 16 data points with $k = 3$. What are the representation colors (centroids)? Show the image after color quantization.

(b) See ipython notebook. Apply the k-means algorithm to the grayscale image with different values of k. Observe the distortion. Choose an ideal value for k, which should be the minimum value for keeping the compressed image visually similar to the original image. Calculate the memory we need for the compressed image.

(c) See ipython notebook. Apply the k-means algorithm to the color image with different values of k. Observe the distortion. Choose an ideal value for k, calculate the memory we need for the compressed image.

(d) See ipython notebook. Here we combine three features of each pixel into one feature as the intensity in grayscale images. Use the k-means algorithm to locate the boundaries of objects. How many objects do you expect from this image? Adjust the value of k and describe your observations. If you are interested in this, you could learn more algorithms for image segmentation from EECS courses in image processing and computer vision.

## 7. Your Own Problem

Write your own problem related to this week's material and solve it. You may still work in groups to brainstorm problems, but each student must submit a unique problem. What is the problem? How to formulate it? How to solve it? What is the solution?

**Contributors:**

- Saavan Patel.
- Siddharth Iyer.
- Yu-Yun Dai.
- Stephen Bailey.