EECS 16B Designing Information Devices and Systems II
Fall 2021 UC Berkeley

Homework 11

This homework is due on Friday, November 12, 2021, at 11:59PM. Self-grades and HW Resubmission are due on Tuesday, November 16, 2021, at 11:59PM.

1. Reading Lecture Notes

Staying up to date with lectures is an important part of the learning process in this course. Here are links to the notes that you need to read for this week: Note 15 Note 16 Note 17

(a) If the outer product form of the SVD of A is $\sum_{i=1}^{r} \sigma_i \vec{u}_i \vec{v}_i^\top$, **what is the best rank k approximiation for A, which we'll call X, that minimizes the Frobenius norm $\|A - X\|_F$?** Assume that $k \leq r$.

(b) We have a data matrix X and its SVD $U\Sigma V^T$. When performing PCA, **what are the principal components if the columns of X are our data points? What are the principal components if the rows of X are our data points?**

2. SVD from the other side

In lecture, we thought about the SVD for a wide matrix M with n rows and $m > n$ columns by looking at the big $m \times m$ symmetric matrix $M^\top M$ and its eigenbasis. This question is about seeing what happens when we look at the small $n \times n$ symmetric matrix $Q = MM^\top$ and its orthonormal eigenbasis $U = [\vec{u}_1, \vec{u}_2, \cdots, \vec{u}_n]$ instead. Suppose we have sorted the eigenvalues so that the real eigenvalues $\widetilde{\lambda}_i$ are sorted in descending order where $\widetilde{\lambda}_i \vec{u}_i = Q\vec{u}_i$.

(a) **Show that $\widetilde{\lambda}_i \geq 0$.**
(HINT: You want to involve $\vec{u}_i^\top \vec{u}_i$ somehow.)

(b) Suppose that we define $\vec{w}_i = \frac{M^\top \vec{u}_i}{\sqrt{\widetilde{\lambda}_i}}$ for all i for which $\widetilde{\lambda}_i > 0$. Suppose that there are ℓ such nonzero eigenvalues. **Show that $W = [\vec{w}_1, \vec{w}_2, \cdots, \vec{w}_\ell]$ has orthonormal columns.**

(c) **Prove that $M = \sum_{i=1}^{\ell} \sqrt{\widetilde{\lambda}_i} \vec{u}_i \vec{w}_i^\top$.**
HINT: Plug in the definition of $\vec{w}_i$ from the previous part. Then show that left multiplying by an arbitrary $\vec{x}^\top$ where $\vec{x} \in \mathbb{R}^n$ produces the same result on both sides. Also remember that U is a basis for $\mathbb{R}^n$.

(d) Consider an arbitrary input $\vec{x} = \vec{x}_\perp + \sum_{i=1}^{\ell} \alpha_i \vec{w}_i$ where $\vec{x}_\perp$ is orthogonal to each of the $\vec{w}_i$. **Show that $M\vec{x}_\perp = \vec{0}$.** The previous part will be helpful.

3. Frobenius Norm

In this problem we will investigate the basic properties of the Frobenius norm.

Similar to how the norm of vector $\vec{x} \in \mathbb{R}^N$ is defined as $\|x\| = \sqrt{\sum_{i=1}^{N} x_i^2}$, the Frobenius norm of a matrix $A \in \mathbb{R}^{N \times M}$ is defined as

$$\|A\|_F = \sqrt{\sum_{i=1}^{N} \sum_{j=1}^{M} |A_{ij}|^2}.$$

A_{ij} is the entry in the i^{th} row and the j^{th} column. This is basically the norm that comes from treating a matrix like a big vector filled with numbers.

(a) With the above definitions, **show that for a 2×2 matrix A:**

$$\|A\|_F = \sqrt{\mathbf{Tr}\{A^\top A\}}. \tag{1}$$

Note: The trace of a matrix is the sum of its diagonal entries. For example, let $A \in \mathbb{R}^{N \times M}$, then,

$$\text{Tr}\{A\} = \sum_{i=1}^{\min(N,M)} A_{ii}$$

Think about how/whether this expression eq. (1) generalizes to general $n \times m$ matrices.

(b) **Show that if U and V are square orthonormal matrices, then**

$$\|UA\|_F = \|AV\|_F = \|A\|_F .$$

(HINT: Use the trace interpretation from part (a).)

(c) **Use the SVD decomposition to show that $\|A\|_F = \sqrt{\sum_{i=1}^{N} \sigma_i^2}$, where $\sigma_1, \ldots, \sigma_N$ are the singular values of A.**
(HINT: The previous part might be quite useful.)

4. Image Compression With SVD

Open the **image_compression.ipynb** Jupyter Notebook. Make sure to read through all the sections to understand how we will use what we know about PCA and SVD in order to compress images! You should write all your answers in the written submission and you don't need to submit the notebook.

(a) Read and run through the cells in Part 1. **Express C as a linear combination of outer products.**

(b) **Express S as a linear combination of outer products.**

(c) Run all the cells in Part 2. Thoroughly read through the 'rank_k_approx' function. **Since we are now only using the top k singular values, in terms of $k, m,$ and n, how many real numbers do we require to describe the rank k approximation of A?**

(d) Run the cells in Part 3 of the notebook. Notice there are two images: One compressed, one uncompressed. **Can you easily visually detect any difference between the two images even though we are using only about 40% of the data?**

(e) Run the cells that plot the singular values. **Given the plots above, why did the image look so good even with only about 40% of the data?**

(f) Play around with the slider for the kitten picture. **What is the lowest acceptable value of k that you can go without losing too much image quality? What were the memory savings?**
Note: Use your best judgment since this depends on eyesight, screen resolution, etc.

(g) Plot the singular values for the image of the US flag. **What do you notice about the singular values here in comparison to the kitten's singular values? What does this mean for our low rank compression?**

(h) Play around with the interactive slider for the US flag. **What is the lowest acceptable value of k here? What is the memory saving?**

5. Movie Ratings and PCA

Recall from the lecture on PCA that we can think of movie ratings as a structured set of data. For every person i and movie j, we have that person's rating $R_{i,j}$ (thought of as a real number).

Suppose that there are m movies and n people. Let's think about arranging this data into a big $n \times m$ matrix R with people corresponding to rows and movies corresponding to columns. So the entry in the i-th row and j-th column should be $R_{i,j}$ above. Note that this is organized differently from how it was in lecture. Each row corresponds to a unique person and each column to a unique movie.

$$R = \begin{bmatrix} R_{11} & R_{12} & \cdots & R_{1m} \\ R_{21} & R_{22} & \cdots & R_{2m} \\ \vdots & \vdots & \cdots & \vdots \\ R_{n1} & R_{n2} & \cdots & R_{nm} \end{bmatrix}$$

(a) Suppose we believe that there is actually an underlying pattern to this rating data and that a separate study has revealed that every movie is characterized by a set of characteristics: say action and comedy. This means that every movie j has a pair of numbers $a[j], c[j]$ that define it. At the same time, every person i has a pair of sensitivities $s_a[i], s_c[i]$ that essentially define that person's preferences. A person i will rate the movie j as $R_{i,j} = s_a[i]a[j] + s_c[i]c[j]$.

If we arrange the sensitivities into a pair of n-dimensional vectors $\vec{s_a}, \vec{s_c}$, and the movie characteristics into a pair of m-dimensional vectors $\vec{a}, \vec{c}$, **use outer products to express the rating matrix R in terms of these vectors $\vec{s_a}, \vec{s_c}, \vec{a}, \vec{c}$.**

(b) Now suppose that we want to discover the underlying nature of movies from the data R itself.

Suppose for this part, that you have four observed rating data vectors (corresponding to four different movies being rated by six individuals).

All of the movie data vectors just happened to be multiples of the following 6-dimensional vector $\vec{w} = \begin{bmatrix} 2 \\ -2 \\ 3 \\ -4 \\ 4 \\ 0 \end{bmatrix}$. (For your convenience, note that $\|\vec{w}\| = 7$.)

You arrange the movie data vectors as the columns of a matrix R given by:

$$R = \begin{bmatrix} | & | & | & | \\ -\vec{w} & -2\vec{w} & 2\vec{w} & 4\vec{w} \\ | & | & | & | \end{bmatrix} \tag{2}$$

You want to perform PCA (for movies) using the SVD of the matrix R to better understand the pattern in your data.

The first "principal component vector" is a *unit vector* that tells which direction we would want to project the columns of R onto to get the best rank-1 approximation for R.

Find this first principal component vector of the columns of R to explain the nature of your movie data points.

(HINT: You don't need to actually compute any SVDs in this simple case. Also, be sure to think about what size vector you want as the answer. Don't forget that you want a unit vector!)

(c) Suppose that now, we have two more data points (corresponding to two more movies being rated by the same set of six people, i.e. we added two columns to our matrix) that are multiples of a different vector $\vec{p}$ where:

$\vec{p} = \begin{bmatrix} 6 \\ 3 \\ -2 \\ 0 \\ 0 \\ 0 \end{bmatrix}$. (For your convenience, note that $\|\vec{p}\| = 7$ and that $\vec{p}^\top \vec{w} = 0$.)

We augment our ratings data matrix with these two new data points to get:

$$R = \begin{bmatrix} | & | & | & | & | & | \\ -\vec{w} & -2\vec{w} & 2\vec{w} & 4\vec{w} & -3\vec{p} & 3\vec{p} \\ | & | & | & | & | & | \end{bmatrix} \tag{3}$$

Find the first two principal components corresponding to the nonzero singular values of R. This is what we would use to best project the movie data points onto a two-dimensional subspace.

What is the first principal component vector? What is the second principal component vector? Justify your answer.

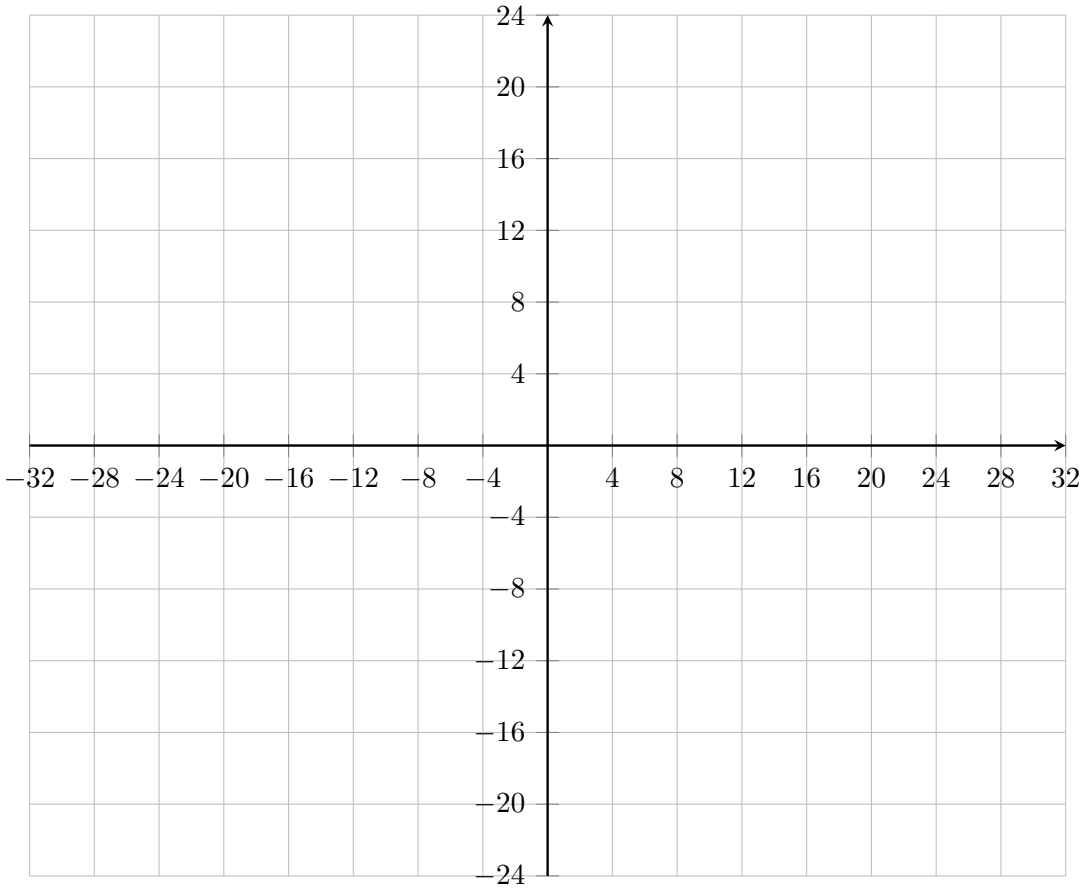
(Hint: Think about the inner product of $\vec{w}$ and $\vec{p}$ and what that implies for being able to appropriately decompose R. Again, very little computation is required here.)

(d) In the previous part, you had

$$R = \begin{bmatrix} | & | & | & | & | & | \\ -\vec{w} & -2\vec{w} & 2\vec{w} & 4\vec{w} & -3\vec{p} & 3\vec{p} \\ | & | & | & | & | & | \end{bmatrix}$$

with $\|\vec{w}\| = 7$ and $\|\vec{p}\| = 7$, satisfying $\vec{p}^\top \vec{w} = 0$.

If we use $\vec{r}_i$ to denote the i-th column of R, **plot the movie data points $\vec{r}_i$ projected onto the first and second principal component vectors along the columns of R.** The coordinate along the first principal component should be represented by horizontal axis and the coordinate along the second principal component should be the vertical axis. **Label each point, and the axes. Remember that principal component vectors are normalized**.



6. Brain-Machine Interface and Neuron Sorting

The iPython notebook `pca_BMI.ipynb` will guide you through the process of analyzing brain machine interface data leveraging the SVD. This will help you to prepare for the project.

For SIXT33N (your robot car), you will need to use the SVD and PCA to classify your sound inputs so that your car does what you tell it to do.

Brain-Machine interfaces (BMIs) are a way for a brain to directly communicate to an external device. They're often used for research, mapping, assisting or repairing human cognitive or sensory-motor function.

Data was collected that shows waveform traces of (simulated) brain waves of a subject whose arm is pointing in certain directions over time. These waveform traces are gathered from electrodes inserted into the brain, but the electrodes are generally recording more than 1 neuron at the same time — the brain is crowded after all. To make predictions based on neuron firing rates, we need to be able to distinguish waveforms that come from different neurons.

Each neuron has its own waveform "signature" shape unique to that neuron. This is due to the physical characteristics of neurons, such as their physical shape and structure. It is impossible to know beforehand how many neurons an electrode measures or what each neuron's waveform looks like, so we must first separate the waveforms from the different neurons near the electrode.

The goal of this problem is to see how we can use the SVD (in its PCA manifestation) and clustering to decide which neuron fired. We will first look at a case where there are only two neurons, then a case where there are three neurons. The neurons have also been presorted using a professional software package, so we can check our model against presorted data.

Please complete the notebook by following the instructions given.

Task 1: Two Neuron Waveform Sorting

(a) The data you have are traces of length 32 timesteps. The data is arranged along the rows of the matrix, each row is one trace. Import the data sets and see the average waveform for each presorted neuron by running the corresponding cells in the .ipynb. **What is the shape of the training data matrix `three_neurons_training`?**

(b) **What do the columns and rows of the training data matrix: `three_neurons_training` represent?**

We can approximate each waveform in a lower dimensional space using PCA. In your implementation of PCA you will decide how to choose these components.

(c) You will be using the SVD in your PCA function. **What matrices does the SVD return? What are the dimensions of these matrices for the SVD of `three_neurons_training`?**

(d) To represent each waveform in a lower dimensional space we want a basis for the time signals of the waveforms of the neurons. To construct this basis we start by taking the SVD (of `three_neurons_training`). **Which of the U and V matrices can we use to construct our desired basis?**

(e) **Read through `PCA_train` and implement `PCA_project`**. We will use these functions throughout the rest of the notebook, so make sure you understand them.

(f) **Call `PCA_train` on `two_neurons_training` and plot the 2 principal components.** Note that since the dataset is randomized, you might get different plots every time you run the second code cell of this notebook (the `_make_training_set` function). Note that these components have no real "physical" relationship to the actual shape of the neuron plots. They are a part of a basis, not the representation of exemplar traces.

(g) Before we project onto our principal components, let us project our data onto 2 random 32 length vectors. **Run the corresponding part in the .ipynb file and comment on the separation of the 2 neuron's distinctive trace shapes in this projected basis.** Do you think that it would be easy to tell them apart just by looking at their projection into these two random dimensions?

(h) **Project `two_neurons_test` onto the two principal components found using the SVD and produce a 2D scatter plot in the new basis.** We will also try projecting the presorted data containing 2 neurons so we can see how the model behaves on the 2 neurons. **Comment on the difference between the projections here and part (g) where you projected onto random directions.**

(i) Now we will repeat this process for three principal components. **Do you see a significant improvement with 3 principal components?**

Task 2: Three Neuron Sorting

(j) In the previous part we sorted 2 neurons using two or three principal components. In this part we are now dealing with 3 neurons. **Replicate what we did in the previous parts by finding and projecting our data on the first two principal components of this three neuron dataset in the .ipynb.**

(k) Now **call `PCA_train` on `three_neurons_training` and plot the 3 principal components. Do the same classification process as the 2 neuron data, but now with the 3 neuron data. Compare your model's behavior with that of the presorted data.** The `plot_3D` function will be useful to view the results.

(l) **How many principal components do you actually need to cluster the 3 neurons? (Hint: Think about whether there was an increase in separability between the last two parts)**

Task 3: Determining Neurons

Now that we know how to project our data onto a basis where it is clearly separable we can classify our points and determine which neuron fired. (This is what we need to know if we want to use the firings of different neurons to build a BMI.)

To classify our points we use the following algorithm:

i. Find centroids: points that represent the average waveform of a particular neuron firing, in the basis of principal components.

ii. Classify each incoming point by assigning it the value of the centroid closest to it (i.e., declare that the neuron waveform that a point represents is the same as the neuron waveform of the centroid the point is closest to).

(m) Since we already have some presorted data, we can use this information to aid in classification. We can calculate our centroids by taking the empirical mean of the presorted data corresponding to a particular neuron firing. **Use this method to find centroids in the corresponding section of the .ipynb file. Then use `which_neuron` on the `two_classified` data set and count the number of times each neuron fired.**

When we are doing such problems in practice, we will not have a presorted data set. In such cases we can use algorithms like k-means clustering to determine our centroids (some of you have seen this algorithm in CS61A during the Maps project, but since Covid, it has not been reliably seen in CS61A). As a refresher, k-means will do the same thing as in (l)i. but with a couple modifications. A step is added at the beginning to randomly assign each data point to some centroid and a step is added to the end such that if the classifications remained the same as last time, exit. Else, go back to step (i) and recompute the centroids based on what they were last classified as. (Essentially, k-means is a technique that just figures out clusters by just seeing which points are more near each other than near other points. It works reasonably well as a heuristic in low dimensional spaces.)

Scipy happens to have a built in function to compute k-means given clustered data, which we have conveniently formatted for you.

Run the code that we have provided to do k-means for you and find the centroids. **Then use `which_neuron` on the `two_classified` data set and count the number of times each neuron fired. Are these values the same as when using the centroids from averages?**

In later courses, you will build on the basic techniques taught here. But in reality, the ideas that you have learned in 16AB so far already form a conceptual core for machine learning. You can go very far with just the ideas you have already learned. Especially when you combine them with what you are about to learn in the next few weeks.

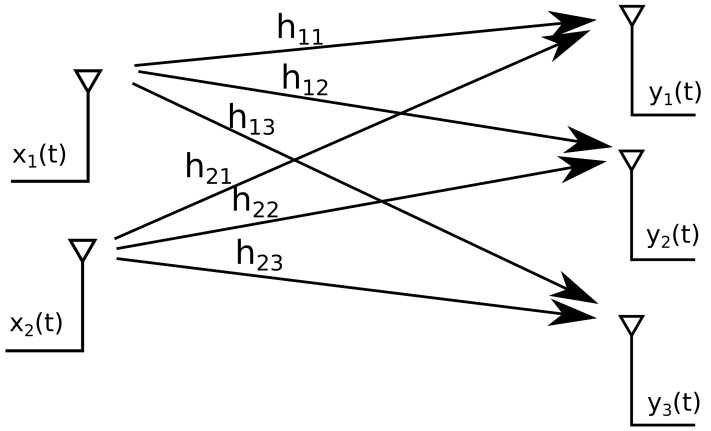
7. MIMO wireless signals

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, 5G, and LTE. Even more importantly, it is one of the core technologies that will underly “6G” communication, where engineers are considering using tens to hundreds of antennas (so called “Massive MIMO”) simultaneously to help support the simultaneous multi-gigabit data rates that will be needed to enable the kind of ubiquitous “metaverse” AR/VR applications that many companies are betting on.

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:

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

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

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.

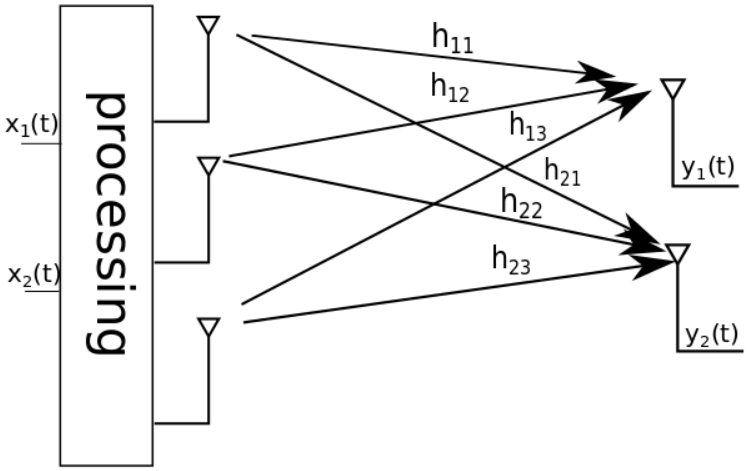
(a) With our new MIMO wireless system, we want to recover the original $\vec{x}(t)$ signal after receiving the $\vec{y}(t)$. But we have three observations and only want to recover two inputs. In order to do this, we will use least squares so that any noise in the measurements is also dealt with. **Write down the least-squares solution for $H\vec{x} \approx \vec{y}$ in terms of the SVD of H ($H = U\Sigma V^T$).**

(b) You might notice something familiar in the form of the solution you found above in relation to the minimum-norm solution you found using the Moore-Penrose pseudoinverse in the last homework. **For the matrix G that pre-multiplies $\vec{y}$ to get the least-squares solution (i.e. $\hat{\vec{x}}(t) = G\vec{y}$), how does it look compared to the pseudoinverse of H?**

(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 $x_2(t)$ to get three numbers $w_1(t), w_2(t), w_3(t)$ that are actually transmitted over the three antennas.** They need to be such that you recover them precisely after they have been transmitted across the channel without doing any further processing. To be more explicit, after the processing and transmission has been done $y_1(t) = x_1(t), y_2(t) = x_2(t)$. **By what matrix should you multiply $\vec{x}$ to get $\vec{w}$?**

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

To actually support this kind of preprocessing for Massive MIMO systems at the data rates that people are envisioning, it turns out that doing it only using digital computers and having a DAC for every antenna burns far too much power to run the digital electronics. What is required is the development of combined hardware/software solutions that combine analog circuits (of the type you can learn about in classes like 142) with digital circuits (of the type you can learn about in classes like 151) and advanced mathematical algorithms.

8. Write Your Own Question And Provide a Thorough Solution.

Writing your own problems is a very important way to really learn material. The famous "Bloom's Taxonomy" that lists the levels of learning (from the bottom up) is: Remember, Understand, Apply, Analyze, Evaluate, and Create. Using what you know to create is the top level. We rarely ask you any homework questions about the lowest level of straight-up remembering, expecting you to be able to do that yourself (e.g. making flashcards). But we don't want the same to be true about the highest level. As a practical matter, having some practice at trying to create problems helps you study for exams much better than simply counting on solving existing practice problems. This is because thinking about how to create an interesting problem forces you to really look at the material from the perspective of those who are going to create the exams. Besides, this is fun. If you want to make a boring problem, go ahead. That is your prerogative. But it is more fun to really engage with the material, discover something interesting, and then come up with a problem that walks others down a journey that lets them share your discovery. You don't have to achieve this every week. But unless you try every week, it probably won't ever happen.

You need to write your own question and provide a thorough solution to it. The scope of your question should roughly overlap with the scope of this entire problem set. This is because we want you to exercise your understanding of this material, and not earlier material in the course. However, feel free to combine material here with earlier material, and clearly, you don't have to engage with everything all at once. A problem that just hits one aspect is also fine.

Note: One of the easiest ways to make your own problem is to modify an existing one. Ordinarily, we do not ask you to cite official course materials themselves as you solve problems. This is an exception. Because the problem making process involves creative inputs, you should be citing those here. It is a part of professionalism to give appropriate attribution.

Just FYI: Another easy way to make your own question is to create a Jupyter part for a problem that had no Jupyter part given, or to add additional Jupyter parts to an existing problem with Jupyter parts. This often helps you learn, especially in case you have a programming bent.

9. Homework Process and Study Group

Citing sources and collaborators are an important part of life, including being a student!
We also want to understand what resources you find helpful and how much time homework is taking, so we can change things in the future if possible.

(a) **What sources (if any) did you use as you worked through the homework?**

(b) **If you worked with someone on this homework, who did you work with?**
List names and student ID's. (In case of homework party, you can also just describe the group.)

(c) **Roughly how many total hours did you work on this homework? Write it down here where you'll need to remember it for the self-grade form.**

Contributors:

- Anant Sahai.
- Ayan Biswas.
- Aditya Arun.
- Kourosh Hakhamaneshi.
- Daniel Abraham.
- Ashwin Vangipuram.
- Nikhil Shinde.
- Nathan Lambert.
- Saavan Patel.

