EECS 16B Designing Information Devices and Systems II
Fall 2021 UC Berkeley Homework 10

This homework is due on Friday, November 5, 2021, at 11:59PM. Self-grades and HW Resubmission are due on Tuesday, November 9, 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

   (a) Consider two vectors $\vec{x} \in \mathbb{R}^m$ and $\vec{y} \in \mathbb{R}^n$, **what are the dimensions of the matrix $\vec{x}\vec{y}^\top$ and what is the rank of $\vec{x}\vec{y}^\top$?**

   (b) Consider a matrix $A \in \mathbb{R}^{m\times n}$ and the rank of A is r. Suppose its SVD is $A = U\Sigma V^\top$ where $U \in \mathbb{R}^{m\times m}$, $\Sigma \in \mathbb{R}^{m\times n}$, and $V \in \mathbb{R}^{n\times n}$. **Write A in terms of the singular values of A and outer products of the columns of U and V?**

2. Symmetric Matrices

We want to show that every real symmetric matrix can be diagonalized by a matrix of its orthonormal eigenvectors. In other words, a symmetric matrix has an orthonormal eigenbasis. This is called the spectral theorem for real symmetric matrices.

In discussion section, you have seen a recursive derivation of a related 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. (This is the same essential proof as that of Schur upper-triangularization. So understanding this problem will help solidify your understanding of that proof as well.)

(a) You will start by proving a basic lemma about real symmetric matrices under an orthonormal change of basis. **Prove that if S is an $m \times m$ symmetric matrix ($S = S^\top$) and U is any $m \times n$ matrix, then $U^\top SU$ is also symmetric**.

(b) Another useful lemma is one about finding orthonormal bases. **Show that given a single nonzero vector $\vec{u}_0$ of dimension n, 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 α.**

(Hint: Use the Gram-Schmidt process on the list of $n+1$ vectors obtained by starting with the given vector and appending the standard basis — i.e. the columns of the identity matrix.)

(c) For the main proof that every real symmetric matrix is diagonalized by a matrix of its orthonormal real eigenvectors, 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 a real eigenvector? Does S have a real eigenbasis? Can this eigenbasis be made orthonormal? Is the matrix S diagonal in this basis? Are the entries of S in this basis real?**

(d) After the base case, we do an inductive stage of the main proof. The first step in the inductive stage is to write down the induction hypothesis. Assume that the property that every real symmetric matrix is diagonalized by a matrix of its orthonormal eigenvectors holds for all symmetric matrices with size $[(n-1) \times (n-1)]$. **Complete the proof template below by filling in the blanks**. *(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.)*

Goal: We want to show that any $[n \times n]$ real symmetric matrix S can be diagonalized by a matrix of its orthonormal real eigenvectors.
Base case: When $n = 1$, this holds for $[1 \times 1]$ matrix $S = [s]$ as you showed in Part (d).
Inductive hypothesis: Let Q be an $[(n-1) \times (n-1)]$ real symmetric matrix with eigenvectors $\vec{u}_0, \ldots, \vec{u}_{n-2}$, then we can express Q as $U\Lambda_Q U^\top$, where Λ_Q is a ________ matrix with ________ eigenvalues of Q along its ________, and U is an $[(n-1) \times (n-1)]$ matrix, where the columns are ________ of Q.

(e) Now think about a symmetric matrix S with size $[n \times n]$. Consider a real eigenvalue λ_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 $S = VXV^\top$, where X is of the form**

$$X = \begin{bmatrix} \lambda_0 & x_{1,2} & \cdots & x_{1,n} \\ 0 & x_{2,2} & \cdots & x_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & x_{n,2} & \cdots & x_{n,n} \end{bmatrix}. \tag{1}$$

That is, the first column of X is $\begin{bmatrix} \lambda_0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$.

(Hint: Follow the proof strategy from lecture. What should the first column of V be? How can you fill out the rest?)

(f) Continue the previous part **to show that in fact S can be written as**

$$S = V \begin{bmatrix} \lambda_0 & \vec{0}^\top \\ \vec{0} & Q \end{bmatrix} V^\top \tag{2}$$

where Q is an $[(n-1) \times (n-1)]$ symmetric matrix and V is the matrix that you found above. *Hint: Define $\vec{v}_0 = \frac{\vec{u}_0}{\|\vec{u}_0\|}$, think of V as $V = \begin{bmatrix} \vec{v}_0 & R \end{bmatrix}$, where the dimension of R is $[n \times (n-1)]$. Also recall that S is a symmetric matrix.*

(g) According to our induction hypothesis, we can write Q as $U \Lambda U^\top$ where U is an orthonormal $[(n-1) \times (n-1)]$ square matrix and Λ is a diagonal matrix with real entries along the diagonal and 0s everywhere else. **Substitute this in the previous part to show that indeed there must exist an orthonormal $[n \times n]$ square matrix W such that**

$$S = W \begin{bmatrix} \lambda_0 & \vec{0}^\top \\ \vec{0} & \Lambda \end{bmatrix} W^\top \tag{3}$$

(Hint: If R and U are both orthonormal matrices, is RU also orthonormal? What is $(RU)^\top(RU)$?)

(h) **Finally, prove that the eigenvalues λ of real, symmetric matrix S are real.** *Hint: Suppose that S had a complex eigenvalue λ with eigenvector $\vec{v}$. Because S is a real matrix, what do you know about $S\overline{\vec{v}}$, where $\overline{\vec{v}}$ is the complex conjugate of $\vec{v}$? What happens when you take a potentially complex number and multiply it by its own complex conjugate? Consequently, what do you know happens if you multiply a complex vector $\vec{v}$ by the conjugate of its transpose: i.e. consider $\overline{\vec{v}}^\top \vec{v}$? Since S is symmetric, what do you know about $\vec{v}^\top S$? What is the conjugate of a real quantity?*

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

3. The Moore-Penrose pseudoinverse for "wide" matrices

Say we have a set of linear equations given by $A\vec{x} = \vec{y}$. If A is invertible, we know that the solution for $\vec{x}$ is $\vec{x} = A^{-1}\vec{y}$. However, what if A is not a square matrix? In 16A, you saw how this problem could be approached for tall "standing up" matrices A where it really wasn't possible to find a solution that exactly matches all the measurements, using linear least-squares. The linear least-squares solution gives us a reasonable answer that asks for the "best" match in terms of reducing the norm of the error vector.

This problem deals with the other case — when the matrix A is wide and "lying down" — with more columns than rows. In this case, there are generally going to be lots of possible solutions — so which should we choose? Why? We will walk you through the **Moore-Penrose pseudoinverse** that generalizes the idea of the matrix inverse and is derived from the singular value decomposition.

This approach to finding solutions complements the OMP approach that you learned in 16A. The OMP approach tries to minimize the number of nonzero entries in the solution. The approach here will try to minimize the length of the vector in a Euclidean sense.

(a) Say you have the matrix

$$A = \begin{bmatrix} 1 & -1 & 1 \\ 1 & 1 & -1 \end{bmatrix}.$$

To find the Moore-Penrose pseudoinverse we start by calculating the SVD of A. That is to say, calculate U, Σ, V such that

$$A = U\Sigma V^T$$

where U and V are orthonormal matrices.

Here we will give you that the decomposition of A is:

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

where:

$$U = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} 2 & 0 & 0 \\ 0 & \sqrt{2} & 0 \end{bmatrix}$$

$$V^T = \begin{bmatrix} 0 & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 1 & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}$$

It is a good idea to be able to calculate the SVD yourself as you may be asked to solve similar questions on your own in the exam.

Let us now think about what the SVD does. Let us look at matrix A acting on some vector $\vec{x}$ to give the result $\vec{y}$. We have

$$A\vec{x} = U\Sigma V^T\vec{x} = \vec{y}.$$

Observe that $V^T\vec{x}$ rotates the vector, Σ scales the result, and U rotates it again. We will try to "reverse" these operations one at a time and then put them together to construct the Moore-Penrose pseudoin-

verse.
If U "rotates" the vector $\left(\Sigma V^T\right)\vec{x}$, what operator can we derive that will undo the rotation?

(b) **Derive a matrix that will "unscale", or undo the effect of Σ where it is possible to undo.** Recall that Σ has the same dimensions as A. Ignore any division by zeros (that is to say, let it stay zero).

(c) **Derive an operator that would "unrotate" by V^T.**

(d) **Try to use this idea of "unrotating" and "unscaling" to derive an "inverse", denoted as $A^\dagger$.** That is to say,

$$\vec{x} = A^\dagger \vec{y}$$

The reason why the word inverse is in quotes (or why this is called a pseudo-inverse) is because we're ignoring the "divisions" by zero.

(e) **Use $A^\dagger$ to solve for a vector $\vec{x}$ in the following system of equations.**

$$\begin{bmatrix} 1 & -1 & 1 \\ 1 & 1 & -1 \end{bmatrix} \vec{x} = \begin{bmatrix} 2 \\ 4 \end{bmatrix}$$

(f) Now we will see why this matrix is a useful proxy for the matrix inverse in such circumstances. **Show that the solution given by the Moore-Penrose pseudoinverse satisfies the minimality property that if $\vec{\hat{x}}$ is the pseudo-inverse solution to $A\vec{x} = \vec{y}$, then $\|\vec{\hat{x}}\| \leq \|\vec{z}\|$ for all other vectors $\vec{z}$ satisfying $A\vec{z} = \vec{y}$.** *(Hint: look at the vectors involved in the V basis. Think about the relevant nullspace and how it is connected to all this.)*

This minimality property is useful in many applications. You saw a control application in lecture. You'll see a communications application in another problem. This is also used all the time in machine learning, where it is connected to the concept behind what is called ridge regression or weight shrinkage.

(g) Consider a generic wide matrix A. We know that A can be written using $A = U\Sigma V^T$ where U and V each are the appropriate size and have orthonormal columns, while Σ is the appropriate size and is a diagonal matrix — all off-diagonal entries are zero. Further assume that the rows of A are linearly independent. **Prove that $A^\dagger = A^T(AA^T)^{-1}$.**

(HINT: Just substitute in $U\Sigma V^T$ for A in the expression above and simplify using the properties you know about U, Σ, V. Remember the transpose of a product of matrices is the product of their transposes in reverse order: $(CD)^T = D^T C^T$.)

This shows that you don't actually need to compute the SVD to get the pseudo-inverse.

4. Weighted minimum norm

You saw in lecture in the context of open-loop control, how we consider problems in which we have a wide matrix A and solve $A\vec{x} = \vec{y}$ such that $\vec{x}$ is a minimum norm solution:

$$\|\vec{x}\| \leq \|\vec{z}\| \tag{4}$$

for all $\vec{z}$ such that $A\vec{z} = \vec{y}$. You then saw this idea again earlier in this HW where you saw how to compute the appropriate "pseudo-inverse" for such wide matrices.

But what if you weren't interested in just the norm of $\vec{x}$? What if you instead cared about minimizing the norm of a linear transformation $C\vec{x}$? For example, suppose that controls were more or less costly at different times.

The problem can be written out mathematically as:

Given a wide matrix A and a matrix C find $\vec{x}$ such that $A\vec{x} = \vec{y}$ and $\|C\vec{x}\| \leq \|C\vec{z}\|$ for all $\vec{z}$ such that $A\vec{z} = \vec{y}$.

(a) Let's start with the case of C being invertible. **Solve this problem (i.e. find the optimal $\vec{x}$ with the minimum $\|C\vec{x}\|$) for the specific matrices and $\vec{y}$ given below. Show your work.**

It is fine to leave your answer as an explicit product of matrices and vectors.

(HINT: You might want to change variables to solve this problem. Don't forget to change back!)

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix}, \quad C = \begin{bmatrix} 0 & 0 & 2 \\ 0 & 1 & 0 \\ 2 & 0 & 0 \end{bmatrix}, \quad \vec{y} = \begin{bmatrix} 2 \\ 1 \end{bmatrix} \tag{5}$$

For convenience, $C^{-1} = \begin{bmatrix} 0 & 0 & \frac{1}{2} \\ 0 & 1 & 0 \\ \frac{1}{2} & 0 & 0 \end{bmatrix}$ and you are also given some SVDs on the following page.

$$A = \left(U_A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\right)\left(\Sigma_A = \begin{bmatrix} \sqrt{2} & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}\right)\left(V_A^\top = \begin{bmatrix} 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 1 & 0 & 0 \\ 0 & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}\right) \tag{6}$$

$$C = \left(U_C = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix}\right)\left(\Sigma_C = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix}\right)\left(V_C^\top = \begin{bmatrix} 0 & 0 & -1 \\ -1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}\right) \tag{7}$$

$$AC = \left(U_{AC} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\right)\left(\Sigma_{AC} = \begin{bmatrix} \sqrt{5} & 0 & 0 \\ 0 & 2 & 0 \end{bmatrix}\right)\left(V_{AC}^\top = \begin{bmatrix} \frac{2}{\sqrt{5}} & \frac{1}{\sqrt{5}} & 0 \\ 0 & 0 & 1 \\ -\frac{1}{\sqrt{5}} & \frac{2}{\sqrt{5}} & 0 \end{bmatrix}\right) \tag{8}$$

$$AC^{-1} = \left(U_{AC^{-1}} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\right)\left(\Sigma_{AC^{-1}} = \begin{bmatrix} \frac{\sqrt{5}}{2} & 0 & 0 \\ 0 & \frac{1}{2} & 0 \end{bmatrix}\right)\left(V_{AC^{-1}}^\top = \begin{bmatrix} \frac{1}{\sqrt{5}} & \frac{2}{\sqrt{5}} & 0 \\ 0 & 0 & 1 \\ -\frac{2}{\sqrt{5}} & \frac{1}{\sqrt{5}} & 0 \end{bmatrix}\right) \tag{9}$$

(b) What if C were a tall matrix with linearly independent columns? **Explicitly describe how you would solve this problem in that case, step by step.**

For convenience, we have copied the problem statement again here: Given a wide matrix A and a matrix C, **find** $\vec{x}$ **such that** $A\vec{x} = \vec{y}$ **and** $\|C\vec{x}\| \leq \|C\vec{z}\|$ **for all** $\vec{z}$ **such that** $A\vec{z} = \vec{y}$.

Here, you can assume that the wide matrix A has linearly-independent rows but is otherwise generic. Similarly, $\vec{y}$ is a generic vector.

(HINT: Does C have a nullspace? Does $C^\top C$ have a nullspace? Does the SVD of C suggest any (invertible) change of coordinates from $\vec{x}$ to $\vec{\widetilde{x}}$ such that $\left\|\vec{\widetilde{x}}\right\| = \|C\vec{x}\|$?)

5. SVD

(a) Consider the matrix

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

Observe that the columns of matrix A are mutually orthogonal with norms $\sqrt{14}$, $\sqrt{3}$, $\sqrt{42}$.

Verify numerically that columns $\begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}$ **and** $\begin{bmatrix} 5 \\ -1 \\ 4 \end{bmatrix}$ **are orthogonal to each other.**

(b) **Write $A = BD$, where B is an orthonormal matrix and D is a diagonal matrix.** What is B? What is D?

(c) **Write out a singular value decomposition of $A = U\Sigma V^T$ using the previous part.** Note the ordering of the singular values in Σ should be from the largest to smallest.

(HINT: There is no need to compute the eigenvalues of anything.)

(d) Given the matrix

$$A = \frac{1}{\sqrt{50}} \begin{bmatrix} 3 \\ 4 \end{bmatrix} \begin{bmatrix} 1 & -1 \end{bmatrix} + \frac{3}{\sqrt{50}} \begin{bmatrix} -4 \\ 3 \end{bmatrix} \begin{bmatrix} 1 & 1 \end{bmatrix},$$

write out a singular value decomposition of matrix A in the form $U\Sigma V^T$. Note the ordering of the singular values in Σ should be from the largest to smallest.

HINT: You don't have to compute any eigenvalues for this. Some useful observations are that

$$\begin{bmatrix} 3, 4 \end{bmatrix} \begin{bmatrix} -4 \\ 3 \end{bmatrix} = 0, \quad \begin{bmatrix} 1, -1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 0, \quad \| \begin{bmatrix} 3 \\ 4 \end{bmatrix} \| = \| \begin{bmatrix} -4 \\ 3 \end{bmatrix} \| = 5, \quad \| \begin{bmatrix} 1 \\ -1 \end{bmatrix} \| = \| \begin{bmatrix} 1 \\ 1 \end{bmatrix} \| = \sqrt{2}.$$

(e) Define the matrix

$$A = \begin{bmatrix} -1 & 4 \\ 1 & 4 \end{bmatrix}.$$

Find the SVD of A. Then, find the eigenvectors and eigenvalues of A. Is there a relationship between the eigenvalues or eigenvectors of A with the SVD of A?

6. **(Updated) Mid-semester Survey**

   Please fill out this survey Google Form. There is a section pertaining to study groups if you are interested in adjustments.

7. (Updated) Orthonormalization for Speeding Up Model Order Selection [Optional – because this problem might have some bugs in the code. Code now released.]

We want to see how orthonormalization and the Gram-Schmidt algorithm can help us speed up nested least squares problems for the purposes of system identification, and in particular, when one of the things that we need to do is figure out the complexity of a model. In various aspects of the course, you have seen models of the following type.

$$x[i+1] = ax[i] + bu[i] + w[i] \tag{10}$$

$$x[i+1] = ax[i] + b_1u_1[i] + b_2u_2[i] + w[i] \tag{11}$$

$$x[i+1] = a_1x[i] + a_2x[i-1] + bu[i] + w[i] \tag{12}$$

However, one of the key challenges in the real world is that we do not necessarily know whether a system is necessarily described by any one of the models above, or perhaps something more complex. Consider a particular generalization of the following form.

$$x[i+1] = a_1x[i] + a_2x[i-1] + \cdots + a_Nx[i-(N-1)] + b_1u[i] + b_2u[i-1] + b_Mu[i-(M-1)] + w[i] \tag{13}$$

Models that take the form in eq. (13) are a particular type of linear filter defined by what is called a higher-order difference-equation or recurrence relation. Such models are sometimes called autoregressive moving average models or ARMA models and are quite flexible in terms of what they can fit.

When starting out with data, we do not know what the coefficients a_i and b_j are, nor do we know M and N. If M and N are known, we say that we know the order of the system model, and we can proceed by least squares to identify our coefficients. It turns out we can use repeated least squares to get an idea of what M and N are. We essentially try out all the different plausible choices for M and N, fit to our training data, and pick the one that works best on some data that we have held out (not used for doing the least-squares fit) for the purpose of evaluating the choice of model order.

To play with this, let us consider the simpler case where $a_i = 0$ for all $i \in \{1, 2, \ldots, N\}$ (or alternatively, $N = 0$), and that the b_i and M are the only unknowns. Such models are called moving average models because the $x[i]$ is a kind of weighted average on a sliding window of input data $u[i]$. Let us further suppose that we knew that M is at most 500, i.e., $M \in \{1, 2, \ldots, 500\}$. We have written such a case below in eq. (14).

$$x[i+1] = b_1u[i] + b_2u[i-1] + b_Nu[i-(M-1)] + w[i] \tag{14}$$

In the equation above of order M, we only use a linear combination of the inputs $u[i]$ up to a lag of M to determine the output. For reasons outside of the scope of this course eq. (14) describes a system called a Finite Impulse Response filter (FIR filter) which you can learn about in EE120 and EE123. Such filters are also foundational to the understanding of what are called convolutional neural networks and two-dimensional generalizations of them are used widely in graphics and image processing.

In this problem, we are given two traces of inputs $u[1], u[2], \ldots, u[L]$ and resulting outputs $x[1], x[2], \ldots, x[L]$. Our goal is to find a reasonable order M and good coefficients $b_1, \ldots, b_N$ using least squares.

(a) In order to use least squares for estimating the parameters, b_i, we first need to form the data into an

M-column matrix equation like eq. (15)

$$\underbrace{\begin{bmatrix} u[M'] & u[M'-1] & \cdots & u[M'-M] \\ u[M'+1] & u[M'] & \cdots & u[M'-M+1] \\ \vdots & \vdots & \vdots & \vdots \\ u[L-1] & u[L-2] & \cdots & u[L-M] \end{bmatrix}}_{D_M} \underbrace{\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_M \end{bmatrix}}_{\vec{p}_M} = \underbrace{\begin{bmatrix} x[M'+1] \\ x[M'+2] \\ \vdots \\ x[L] \end{bmatrix}}_{\vec{s}_M} \tag{15}$$

This done for you in the code. Because we are going to be searching for the right model order, we will just choose M' to be big enough to allow the largest possible model order to also fit without forcing us to look at inputs u for time indexes that we do not have.

Run the code in the jupyter notebook that evaluates estimated parameters for models of orders $M = 1, 2, \ldots, 500$. **What is the optimal order M? How long did this take to run?**

(b) As discussed in lecture, we can speed up nested least-squares computations by using orthonormalization.

The QR decomposition of a matrix $A = [\vec{d}_1, \vec{d}_2, \ldots, \vec{d}_M]$ is a way of summarizing the Gram-Schmidt process:

$$A = QR = \begin{bmatrix} | & | & & | \\ \vec{q}_1 & \vec{q}_2 & \cdots & \vec{q}_M \\ | & | & & | \end{bmatrix} \begin{bmatrix} \vec{q}_1^\top \vec{d}_1 & \vec{q}_1^\top \vec{d}_2 & \cdots & \vec{q}_1^\top \vec{d}_M \\ 0 & \vec{q}_2^\top \vec{d}_2 & \cdots & \vec{q}_2^\top \vec{d}_M \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \vec{q}_N^\top \vec{d}_M \end{bmatrix} \tag{16}$$

where $\vec{d}_i$ are the column vectors of A and $\vec{q}_i$ are the orthonormal column vectors of Q output by the Gram-Schmidt procedure. The upper-triangular R matrix essentially keeps track of the calculations that happened during the Gram-Schmidt process and tells you how to go to the orthonormal basis given that you started in the original basis.

To see why this helps, consider the solution to a least-squares problem $A\vec{x} \approx \vec{b}$ in terms of the final projected vector:

$$\begin{aligned} A(A^\top A)^{-1}A^\top \vec{b} &= QR((QR)^\top QR)^{-1}(QR)^\top \vec{b} \\ &= QR(R^\top Q^\top QR)^{-1}R^\top Q^\top \vec{b} \\ &= QR(R^\top R)^{-1}R^\top Q^\top \vec{b} \\ &= QRR^{-1}(R^\top)^{-1}R^\top Q^\top \vec{b} \\ &= QQ^\top \vec{b} \end{aligned}$$

which is a much simpler computation. However, even getting the original parameters is simpler since those are given by

$$\begin{aligned} (A^\top A)^{-1}A^\top \vec{b} &= ((QR)^\top QR)^{-1}(QR)^\top \vec{b} \\ &= (R^\top Q^\top QR)^{-1}R^\top Q^\top \vec{b} \\ &= (R^\top R)^{-1}R^\top Q^\top \vec{b} \\ &= R^{-1}Q^\top \vec{b} \end{aligned}$$

Notice that although there is an inverse there, this is simply asking us to solve an upper-triangular system of equations

$$R\vec{x} = Q^\top \vec{b} \tag{17}$$

which can be done in time proportional to the number of nonzero entries of R since we can use back-substitution. This is a fast inverse to calculate.

A naive approach to attempting to get a speedup would simply be to use this procedure to solve the individual least-squares problems while leaving the earlier code structure intact. However, this will not work to give us any real speed-up.

To get an actual speed-up requires us to reduce redundant computations by reusing values that have been already computed, and not wasting time by recalculating them.

Conceptually, we can speed up our computations by reusing the details from the previous iteration of the Gram-Schmidt based QR decomposition.

Take a look at the QR decomposition of order M:

$$Q_M R_M = \begin{bmatrix} | & | & & | \\ \vec{q}_1 & \vec{q}_2 & \cdots & \vec{q}_M \\ | & | & & | \end{bmatrix} \begin{bmatrix} \vec{q}_1^\top \vec{d}_1 & \vec{q}_1^\top \vec{d}_2 & \cdots & \vec{q}_1^\top \vec{d}_M \\ 0 & \vec{q}_2^\top \vec{d}_2 & \cdots & \vec{q}_2^\top \vec{d}_M \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \vec{q}_M^\top \vec{d}_M \end{bmatrix} \tag{18}$$

The following is the QR decomposition of order $M+1$:

$$Q_{M+1} R_{M+1} = \begin{bmatrix} | & | & & | & | \\ \vec{q}_1 & \vec{q}_2 & \cdots & \vec{q}_M & \vec{q}_{M+1} \\ | & | & & | & | \end{bmatrix} \begin{bmatrix} \vec{q}_1^\top \vec{d}_1 & \vec{q}_1^\top \vec{d}_2 & \cdots & \vec{q}_1^\top \vec{d}_M & \vec{q}_1^\top \vec{d}_{M+1} \\ 0 & \vec{q}_2^\top \vec{d}_2 & \cdots & \vec{q}_2^\top \vec{d}_M & \vec{q}_2^\top \vec{d}_{M+1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & \vec{q}_M^\top \vec{d}_M & \vec{q}_M^\top \vec{d}_{M+1} \\ 0 & 0 & \cdots & 0 & \vec{q}_{M+1}^\top \vec{d}_{M+1} \end{bmatrix} \tag{19}$$

$$= \begin{bmatrix} Q_M & \vec{q}_{M+1} \end{bmatrix} \begin{bmatrix} R_M & Q_M^\top \vec{d}_{M+1} \\ \vec{0}^\top & \vec{q}_{M+1}^\top \vec{d}_{M+1} \end{bmatrix} \tag{20}$$

Observe that Q and R of order M are sub-blocks of Q and R of order $M+1$, respectively.

Finally, in our problem we know the order in which we are going to consider the model orders. This means that we already know when we start what the widest data matrix is going to be.

If you think about the matrix R, it is upper-triangular. Consequently, inverting it is like doing back-substitution. This means that computing R^{-1} once for the largest size will actually give us access to R_M^{-1} as well since those will just be blocks within R^{-1} as well.

This allows us to safely apply another principle of optimizing code: if you can use an optimized function that was written by a group of people a lot more experienced than you, try to use that function instead of trying to implement it yourself. In this case, numpy has a built in and well-optimized QR decomposition in its linalg library. We should use that instead of trying to reinvent the wheel.

Implement the faster nested least squares method that computes solutions for orders $M = 1, 2, \ldots, M_{\max}$, by filling in the `FIR_solve_fast` function in the jupyter notebook. Compare the run time of this method with the original least-squares based method that did not try to exploit problem structure in any way. *Hint: Find a relationship between the parameters of model order M and the parameters of the model order $M+1$ by utilizing the relationship between R_M^{-1} and R_{M+1}^{-1}.*

8. Speeding Up OMP [OPTIONAL — because OMP is not in clean scope]

Recall the imaging lab from EECS16A where we projected masks on an object to scan it to our computer using a single pixel measurement device, that is, a photoresistor. In that lab, we were scanning a 30×40 image having 1200 pixels. In order to recover the image, we took exactly 1200 measurements because we wanted our 'measurement matrix' to be invertible.

However, we saw in 16A lecture near the end of the semester that an iterative algorithm that does "matching and peeling" can enable reconstruction of a sparse vector (i.e. one that has mostly zeros in it) while reducing the number of samples that need to be taken from it. In the case of imaging, the idea of sparsity corresponds to most parts of the image being black with only a small number of light pixels. In these cases, we can reduce the overall number of samples necessary. This would reduce the time required for scanning the image. (This is a real-world concern for things like MRI where people have to stay still while being imaged. It is also a key principle that underlays a lot of modern machine learning.)

In this problem, we have a 2D image I of size 91×120 pixels for a total of 10920 pixels. The image is made up of mostly black pixels except for 476 of them that are white.

Although the imaging illumination masks we used in the lab consisted of zeros and ones, in this question, we are going to have masks with real values — i.e. the light intensity is going to vary in a more finely grained way. Say that we have an imaging mask M_0 of size 91×120. The measurements using the photoresistor using this imaging mask can be represented as follows.

First, let us vectorize our image to $\vec{i}$ which is a column vector of length 10920. Likewise, let us vectorize the mask M_0 to $\vec{m}_0$ which is a column vector of length 10920. Then the measurement using M_0 can be represented as

$$b_0 = \vec{m}_0^T \vec{i}.$$

Say we have a total of K measurements, each taken with a different illumination mask. Then, these measurements can collectively be represented as

$$\vec{b} = \mathbf{A}\vec{i},$$

where $\mathbf{A}$ is an $K \times 10920$ size matrix whose rows contain the vectorized forms of the illumination masks, that is

$$\mathbf{A} = \begin{bmatrix} \vec{m}_1^T \\ \vec{m}_2^T \\ \vdots \\ \vec{m}_K^T \end{bmatrix}.$$

To show that we can reduce the number of samples necessary to recover the sparse image I, we are going to only generate 6500 masks. The columns of $\mathbf{A}$ are going to be approximately uncorrelated with each other. The following question refers to the part of Jupyter notebook file accompanying this homework related to this question.

(a) In the jupyter notebook, we have completed a function `OMP` to run the naive OMP algorithm you learned in EECS16A. Read through the code and understand the function `OMP`.

We have also supplied code that reads a PNG file containing a sparse image, takes measurements, and performs OMP to recover it. An example input image file is also supplied together with the code. Using `smiley.png`, generate an image of size 91×120 pixels of sparsity less than 400 and recover it using OMP with 6500 measurements.

Run the code `rec = OMP((height, width), sparsity, measurements, A)` and see the image being correctly reconstructed from a number of samples smaller than the number of

pixels of your figure. What is the image? Report how many seconds this took to run.

Remark: Note that this remark is not important for solving this problem; it is about how such measurements could be implemented in our lab setting. When you look at the vector `measurements` you will see that it has zero average value. Likewise, the columns of the matrix containing the masks $\mathbf{A}$ also have zero average value. To satisfy these conditions, some entries need to have negative values. However, in an imaging system, we cannot project negative light. One way to get around this problem is to find the smallest value of the matrix $\mathbf{A}$ and subtract it from all entries of $\mathbf{A}$ to get the actual illumination masks. This will yield masks with positive values, hence we can project them using our real-world projector. After obtaining the readings using these masks, we can remove their average value from the readings to get measurements as if we had multiplied the image using the matrix $\mathbf{A}$. This is being done silently for you in the code.

(b) Now let us try using our naive implementation of OMP to recover a slightly less sparse image: An example input image file is supplied together with the code. Using `pika.png`, generate an image of size 100×100 pixels of sparsity less than 800 and recover it using OMP with 9000 measurements.

Run the corresponding code blocks in the accompanying jupyter notebook and report back the number of seconds it took to to reconstruct the image. (Take a well deserved break, this may take upwards of ten minutes to run!)

(c) As you saw, reconstructing the image with the staff's naive implementation took quite a while. Modify the code to run faster by using a Gram-Schmidt orthonormalization to speed it up. **Edit the code given to you in the jupyter notebook. Report back the number of seconds it took to run the reconstruct the image `pika.png`.** Note this should be less than previous part.

This is the only place in the problem where you should have to actually edit code.

(d) (Optional, not in scope) **Do any other modifications you want to further speed up the code.**

Hint: When possible, how would you safely extract multiple peaks corresponding to multiple pixels in one go and add them to the recovered list? Would this speed things up?

9. 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.

10. 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:

- Siddharth Iyer.
- Yu-Yun Dai.
- Sanjit Batra.
- Anant Sahai.

- Sidney Buchbinder.
- Gaoyue Zhou.
- Justin Yim.
- Druv Pai.
- Nikhil Shinde.
- Elena Jia.
- Divija Hasteer.
- Jichan Chung.
- Moses Won.
- Orhan Ocal.
- Vasuki Narasimha Swamy.