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 \( \vec{x}\vec{y}^\top \)?

**Solution:** Let \( \vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} \) and \( \vec{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \), then

\[
\vec{x}\vec{y}^\top = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} \begin{bmatrix} y_1 & y_2 & \cdots & y_n \end{bmatrix} = \begin{bmatrix} y_1x_1 & y_2x_2 & \cdots & y_nx_m \end{bmatrix}
\]

(1)

\[
\vec{x}\vec{y}^\top = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} \vec{y}^\top = \begin{bmatrix} -x_1 \vec{y}^\top \\ -x_2 \vec{y}^\top \\ \vdots \\ -x_m \vec{y}^\top \end{bmatrix}
\]

(2)

We will have \( n \) multiples of \( \vec{x} \), and since \( \vec{x} \) has \( m \) entries, the matrix has dimensions \( m \times n \). We can see that each column of \( \vec{x}\vec{y}^\top \) is a multiple of \( \vec{x} \) and each row is a multiple of \( \vec{y}^\top \), thus \( \vec{x}\vec{y}^\top \) has rank 1, unless either of the vectors \( \vec{x} \) or \( \vec{y} \) was the (appropriate shape) all zero vector \( \vec{0} \). If the result is the zero matrix, the rank would be 0.

(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 \( V \).

**Solution:** We have \( A = \sum_{i=1}^{r} \sigma_i \vec{u}_i\vec{v}_i^\top \) where \( \vec{u}_i \) and \( \vec{v}_i \) are the \( i \)-th columns of \( U \) and \( V \). Note that we only sum to \( r \) since \( A \) has rank \( r \) and hence it has \( r \) non-zero singular values \( \sigma_1, \ldots, \sigma_r \). This is the outer product form of the SVD.
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.**

**Solution:** We must show \( (U^\top SU)^\top = (U^\top SU) \) to show it is symmetric. The transpose of \( U^\top SU \), \( (U^\top SU)^\top \) is identical to \( U^\top SU \) as follows:

\[
(U^\top SU)^\top = (SU)^\top U = U^\top S^\top U = U^\top SU
\]

since \( S^\top = S \).

(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 \( \alpha \).**

**Solution:** Finding \( \vec{v}_0, \ldots, \vec{v}_{n-1} \) can be done with two steps: (1) based on \( \vec{u}_0 \), find \( \vec{u}_1, \ldots, \vec{u}_N \), such that \( \text{span}(\vec{u}_0, \vec{u}_1, \ldots, \vec{u}_N) = \mathbb{R}^n \) with \( N \geq n \) and (2) apply the Gram-Schmidt process to orthonormalize that set of vectors, dropping any zero vectors that we encounter along the way.

For (1), we can simply append the standard basis (i.e. the columns of the identity matrix) to a list starting with \( \vec{u}_0 = \frac{\vec{u}_0}{\|\vec{u}_0\|} \). This gives us \( N = n + 1 \geq n \) vectors that certainly span all of \( \mathbb{R}^n \) since the standard basis definitely spans all of \( \mathbb{R}^n \) even by itself.

For (2), we can simply apply the Gram-Schmidt process to find an orthonormal set of \( n \) vectors. The Gram-Schmidt process takes a finite, linearly independent list \( U = \{\vec{u}_0, \vec{u}_1, \ldots, \vec{u}_N\} \) and generates an orthonormal list \( W = \{\vec{w}_0, \vec{w}_1, \ldots, \vec{w}_{n-1}\} \) that spans the same space as the original list \( U \). If along the way, it finds a zero vector, it just drops that vector.

This gives us the basis \( V \) and we know that the first element is just a scaled version of the original \( \vec{u}_0 \) by construction.

(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?**

**Solution:** Yes, it has an eigenvector, because \( S \) would be a scalar. Let \( S = [s] \) and \( \vec{u} = [1] \). Then note that \( S\vec{u} = s\vec{u} \). This implies that \( s \) is an eigenvalue and \( \vec{u} = [1] \) is an eigenvector. This eigenvector \( \vec{u} \) makes up an eigenbasis in \( \mathbb{R}^1 \) and has real entries. This eigenbasis is trivially orthonormal, because there is no other eigenvector of \( S \) which is not orthogonal to \( \vec{u} \). We can think of \( S \) as a \([1 \times 1]\) matrix,
such that the only entry is real. Also, $S$ is diagonal in that basis because there is only one element. Writing it out in full, $S = [1][s][1]^T$.

(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 (c). |  |
| **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 $U\Lambda_QU^\top$, where $\Lambda_Q$ is a _______ matrix with _______ eigenvalues of $Q$ along its _______, and $U$ is an $[(n - 1) \times (n - 1)]$ matrix, where the columns are _______ of $Q$. |  |

**Solution:**

| **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 (c). |  |
| **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 $U\Lambda_QU^\top$, where $\Lambda_Q$ is a diagonal matrix with the real eigenvalues of $Q$ along its diagonal, and $U$ is an $[(n - 1) \times (n - 1)]$ matrix, where the columns are orthonormal real eigenvectors of $Q$. |  |

(e) Now, with the base case done and the inductive hypothesis written down, we can think about a symmetric matrix $S$ with size $[n \times n]$. We want to build towards proving that indeed $S$ can be diagonalized by a matrix of real orthonormal eigenvectors.

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 matrix $V$ to change basis to show that $S = VXV^\top$, where $X$ is of the form**

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

*(4)*

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?)*

**Solution:** According to (b), using the Gram-Schmidt process to construct a basis starting with $\vec{u}_0$, we could derive an orthonormal set of $n$ vectors, $\vec{v}_0, \ldots, \vec{v}_{n-1}$. Choose $\vec{v}_0 = \frac{\vec{u}_0}{||\vec{u}_0||}$. Let $\vec{v}_0, \ldots, \vec{v}_{n-1}$ be the columns of matrix $V$. Suppose $X = V^\top SV$, such that $S = VXV^\top$ (because $V$ is an orthonormal basis, $V^\top V =VV^\top = I$).
Hence we have

\[
X = \begin{bmatrix}
\vec{v}_0^\top \\
\vec{v}_1^\top \\
\vdots \\
\vec{v}_{n-1}^\top
\end{bmatrix}
S \begin{bmatrix}
\vec{v}_0 \\
\vec{v}_1 \\
\vdots \\
\vec{v}_{n-1}
\end{bmatrix},
\]  
(5)

where \(\vec{v}_0\) is the normalized eigenvector corresponding to the eigenvalue \(\lambda_0\) of \(S\), which means \(S\vec{v}_0 = \lambda_0\vec{v}_0\). The first column of \(X\) will be

\[
\begin{bmatrix}
\vec{v}_0^\top \\
\vec{v}_1^\top \\
\vdots \\
\vec{v}_{n-1}^\top
\end{bmatrix} S\vec{v}_0 = \begin{bmatrix}
\vec{v}_0^\top \\
\vec{v}_1^\top \\
\vdots \\
\vec{v}_{n-1}^\top
\end{bmatrix} \lambda_0 \vec{v}_0.
\]  
(6)

Since \(V\) is an orthonormal matrix, \(\vec{v}_i^\top \vec{v}_j = 1\) when \(i = j\); otherwise it is zero. Therefore, the first column of \(X\) is

\[
\begin{bmatrix}
\lambda_0 \\
0 \\
\vdots \\
0
\end{bmatrix}
\]  
(7)

which is what we wanted to show.

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

\[
S = V \begin{bmatrix}
\lambda_0 \\
0 \\
\vdots \\
0
\end{bmatrix} \begin{bmatrix}
\vec{0}^\top \\
Q
\end{bmatrix} V^\top
\]  
(8)

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.

Solution: Recall the lemma we proved in (a), \(X\) must be a symmetric matrix, because \(X = V^\top SV\) where \(S\) is a symmetric matrix and \(V\) is orthonormal. Since we have proved the first column of this matrix is

\[
\begin{bmatrix}
\lambda_0 \\
0 \\
\vdots \\
0
\end{bmatrix},
\]  
(9)

the transpose of this is the first row. Hence \(X\) can be written as

\[
\begin{bmatrix}
\lambda_0 & \vec{0}^\top \\
\vec{0} & Q
\end{bmatrix},
\]  
(10)
where \( Q \) is an \( n - 1 \) by \( n - 1 \) symmetric matrix. This is a short direct argument that gets us what we want.

An alternative approach was shown in discussion where we computed \( X \) directly. Let \( R \) be a matrix with columns \( \vec{v}_1, \ldots, \vec{v}_{n-1} \), which are orthogonal unit vectors. Recall that \( \lambda_0 \) is an eigenvalue of \( S \), with the corresponding eigenvector \( \vec{v}_0 \). From here, we can compute \( X \) step by step:

\[
X = V^T S V = \begin{bmatrix} \vec{v}_0^T & R^T \end{bmatrix} S \begin{bmatrix} \vec{v}_0 & R \end{bmatrix}
\]

\[
= \begin{bmatrix} \vec{v}_0^T \\ R^T \end{bmatrix} S \begin{bmatrix} \vec{v}_0 & R \end{bmatrix}
\]

\[
= \begin{bmatrix} \vec{v}_0^T \\ R^T \end{bmatrix} \begin{bmatrix} S \vec{v}_0 \\ S R \end{bmatrix}
\]

\[
= \begin{bmatrix} \vec{v}_0^T \\ R^T \end{bmatrix} \begin{bmatrix} \lambda_0 \vec{v}_0 \\ SR \end{bmatrix}
\]

Now we could write \( X \) as

\[
\begin{bmatrix} \lambda_0 \vec{v}_0^T \vec{v}_0 & \vec{v}_0^T S R \\ \lambda_0 R^T \vec{v}_0 & \vec{v}_0^T \vec{v}_0 \end{bmatrix}
\]

Because \( \vec{v}_0 \) is a unit vector, \( \vec{v}_0^T \vec{v}_0 = 1 \). Since \( V \) is an orthonormal matrix, the inner product of \( \vec{v}_0 \) with column vectors in \( R \) must be 0, so \( R^T \vec{v}_0 = \vec{0} \). Also, \( \vec{v}_0^T S R = (S \vec{v}_0)^T R = \lambda_0 \vec{v}_0^T R = \vec{0} \). Then we get the desired form of \( X \), where \( Q = R^T S R \). Recall the lemma we proved in (a), because \( S \) is a symmetric matrix, while the columns of \( R \) are orthonormal, \( Q \) must be symmetric.

Either one of these approaches is fine.

(g) According to our induction hypothesis, we can write \( Q \) as \( U \Lambda U^T \) where \( U \) is an orthonormal \( (n-1) \times (n-1) \) square matrix and \( \Lambda \) 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**

\[
S = W \begin{bmatrix} \lambda_0 & 0 \\ 0 & \Lambda \end{bmatrix} W^T
\]

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

**Solution:** From the previous part, we know that, \( S \) can be written as follows.

\[
S = \begin{bmatrix} \vec{v}_0 \\ R \end{bmatrix} \begin{bmatrix} \lambda_0 & 0 \\ 0 & Q \end{bmatrix} \begin{bmatrix} \vec{v}_0^T \\ R^T \end{bmatrix}
\]

Thanks to our induction hypothesis, we can compute \( S \) as follows:

\[
S = \begin{bmatrix} \vec{v}_0 \\ R \end{bmatrix} \begin{bmatrix} \lambda_0 & 0 \\ 0 & U \Lambda U^T \end{bmatrix} \begin{bmatrix} \vec{v}_0^T \\ R^T \end{bmatrix}
\]

At this point, what we would like to do is to pull the \( U \)'s out of the inner matrix and stick them with
the Rs in the outer matrices. Then we would be done. This could be done immediately because of the properties of block matrix multiplication, or we could further justify it. If you wanted to further justify it (this was not required for full credit), we can take a few approaches. We detail three below.

**Approach 1:** Given that $U$ is orthonormal, we could try to guess a structure for some matrix $P$ which includes $U$ and satisfies the following equation.

$$P \begin{bmatrix} \lambda_0 & \vec{0}^\top \\ \vec{0} & \Lambda \end{bmatrix} P^\top = \begin{bmatrix} \lambda_0 & \vec{0}^\top \\ \vec{0} & U\Lambda U^\top \end{bmatrix}$$ (19)

In finding this $P$, we could pre-multiply it by the matrix $\begin{bmatrix} \vec{v}_0 & R \end{bmatrix}$ to find $W$.

We cannot factor out $U$ alone, as it is of size $(n-1) \times (n-1)$ and is incompatible with multiplying the $n \times n$ matrix $\begin{bmatrix} \lambda_0 & \vec{0}^\top \\ \vec{0} & \Lambda \end{bmatrix}$. We can augment $U$ with an extra row and column with an element that would leave the $\lambda_0$ the same. Such a matrix candidate is $P = \begin{bmatrix} 1 & \vec{0}^\top \\ \vec{0} & U \end{bmatrix}$, as the 1 may leave $\lambda_0$ untouched. As with any guess, we should verify that it satisfies the relevant equality. Substituting in our guess for $P$, we get the following.

$$P \begin{bmatrix} \lambda_0 & \vec{0}^\top \\ \vec{0} & \Lambda \end{bmatrix} P^\top = \begin{bmatrix} \lambda_0 & \vec{0}^\top \\ \vec{0} & U\Lambda U^\top \end{bmatrix} \begin{bmatrix} \lambda_0 & \vec{0}^\top \\ \vec{0} & U \end{bmatrix}$$ (20)

$$= \begin{bmatrix} \lambda_0 \vec{0} + \vec{0}^\top U \vec{0} + \lambda_0 \vec{0}^\top U + \vec{0}^\top U \Lambda \end{bmatrix}$$ (21)

$$= \begin{bmatrix} \lambda_0 & \vec{0}^\top U \Lambda \end{bmatrix} \begin{bmatrix} \lambda_0 & \vec{0}^\top \\ \vec{0} & U \end{bmatrix}$$ (22)

$$= \begin{bmatrix} \lambda_0 & \vec{0}^\top U \Lambda \end{bmatrix} \begin{bmatrix} \lambda_0 & \vec{0}^\top U \Lambda \end{bmatrix}$$ (23)

$$= \begin{bmatrix} \lambda_0 & \vec{0}^\top U \Lambda \end{bmatrix}$$ (24)

Which shows that the factorization of the matrix with $\lambda_0$ and $U\Lambda U^\top$ with this choice of $P$ is indeed valid and that we can compute $W$.

$$W = \begin{bmatrix} \vec{v}_0 & R \end{bmatrix} \begin{bmatrix} 1 & \vec{0}^\top \\ \vec{0} & U \end{bmatrix} = \begin{bmatrix} \vec{v}_0 + R \vec{0} & \vec{v}_0 \vec{0}^\top + RU \end{bmatrix} = \begin{bmatrix} \vec{v}_0 & RU \end{bmatrix}$$ (25)

**Approach 2:** We can express the $n \times n$ identity matrices in terms of the orthonormal $(n-1) \times (n-1)$ matrices $U$ and $U^\top$.

$$I = \begin{bmatrix} 1 & \vec{0}^\top \\ \vec{0} & U^\top \end{bmatrix} \begin{bmatrix} 1 & \vec{0}^\top \\ \vec{0} & U \end{bmatrix} = \begin{bmatrix} 1 & \vec{0}^\top U^\top \\ \vec{0} & \vec{0}^\top \end{bmatrix}$$ (26)

We can inject these expressions into $S$ after $\begin{bmatrix} \vec{v}_0 & R \end{bmatrix}$ and before $\begin{bmatrix} \vec{v}_0 & R \end{bmatrix}^\top$ and by something that
is akin to the reverse of the primary calculation of the first approach, we get $WP$ and $P^\top W^\top$ and a diagonal matrix due to cancellation of the $U$ and $U^\top$ in $U \Lambda U^\top$.

Approach 3: Guess at $W = \begin{bmatrix} \vec{v}_0 & RU \end{bmatrix}$ directly, and verify that it gives the same expression for $S$ if we check the following equality by working out the multiplication.

$$S = \begin{bmatrix} \vec{v}_0 & RU \end{bmatrix} \begin{bmatrix} \lambda_0 & 0 \\ 0 & \Lambda \end{bmatrix} \begin{bmatrix} \vec{v}_0^\top \\ U^\top \end{bmatrix} = \begin{bmatrix} \vec{v}_0 & RU \end{bmatrix} \begin{bmatrix} \lambda_0 & 0 \\ 0 & \Lambda \end{bmatrix} \begin{bmatrix} \vec{v}_0^\top \\ U^\top \end{bmatrix}$$

(27)

First, evaluate the left hand side of $\equiv$.

$$\begin{bmatrix} \vec{v}_0 & RU \end{bmatrix} \begin{bmatrix} \lambda_0 & 0 \\ 0 & \Lambda \end{bmatrix} \begin{bmatrix} \vec{v}_0^\top \\ U^\top \end{bmatrix} = \begin{bmatrix} \lambda_0 \vec{v}_0 & RU \Lambda U^\top \end{bmatrix} \begin{bmatrix} \vec{v}_0^\top \\ U^\top \end{bmatrix} = \lambda_0 \vec{v}_0 \vec{v}_0^\top + RU \Lambda U^\top R^\top$$

(28)

Then, evaluate the right hand side.

$$\begin{bmatrix} \vec{v}_0 & RU \end{bmatrix} \begin{bmatrix} \lambda_0 & 0 \\ 0 & \Lambda \end{bmatrix} \begin{bmatrix} \vec{v}_0^\top \\ U^\top \end{bmatrix} = \begin{bmatrix} \lambda_0 \vec{v}_0 & RU \Lambda \end{bmatrix} \begin{bmatrix} \vec{v}_0^\top \\ U^\top \end{bmatrix} = \lambda_0 \vec{v}_0 \vec{v}_0^\top + RU \Lambda U^\top R^\top$$

(29)

We see that they are equal.

Regardless of whether we felt required to justify pulling the $U$’s out, we know that the multiplication of two orthonomal matrices (in this case, $R$ and $U$) is an orthonormal matrix as well. The following calculation shows why.

$$(RU)^\top (RU) = U^\top R^\top RU = U^\top IU = U^\top U = I$$

(30)

We can further verify that $\vec{v}_0$ is orthogonal to all columns of $RU$ by seeing what happens to $\vec{v}_0^\top RU$:

$$\vec{v}_0^\top RU = (\vec{v}_0^\top R)U = \vec{0}^\top U = \vec{0}^\top$$

(31)

The second equality above is true because $\vec{0}$ is orthogonal to column vectors of $R$. Therefore, we can now define the orthonormal basis $W$ as:

$$W = \begin{bmatrix} \vec{v}_0 & RU \end{bmatrix}$$

(32)

This shows what we wanted to prove:

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

(33)

and we are done.

(h) **Finally, prove that the eigenvalues $\lambda$ of real, symmetric matrix $S$ are real.** Hint: Suppose that $S$ had a complex eigenvalue $\lambda$ with eigenvector $\vec{v}$. Because $S$ is a real matrix, what do you know about $S\vec{v}$, where $\vec{v}$ 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 $\vec{v}^\top \vec{v}$? Since $S$ is symmetric, what do you know about $\vec{v}^\top S$? What is the conjugate of a real quantity?

**Solution:** Let $\lambda$ be a possibly complex eigenvalue of a real symmetric matrix $S$. Thus, there is a
nonzero vector $\vec{v}$ such that $S\vec{v} = \lambda \vec{v}$.

By taking the complex conjugates of both sides, and noting that $\overline{S} = S$ since $S$ has real entries, we get

$$\overline{S\vec{v}} = \overline{\lambda \vec{v}} \implies S\overline{\vec{v}} = \overline{\lambda} \cdot \overline{\vec{v}}$$

Now, using $S^\top = S$

$$\overline{\vec{v}}^\top S\vec{v} = \overline{\overline{\vec{v}}^\top} (S\vec{v}) = \overline{\vec{v}}^\top (\lambda \vec{v}) = \lambda \overline{\vec{v}}^\top \overline{\vec{v}} = \lambda \|\vec{v}\|^2 \quad (34)$$

$$\vec{v}^\top S\vec{v} = (S\overline{\vec{v}})^\top \overline{\vec{v}} = \overline{(\lambda \overline{\vec{v}})^\top \overline{\vec{v}}} = \overline{\lambda} \overline{(\overline{\vec{v}}^\top \overline{\vec{v}})} \|\vec{v}\|^2 \quad (35)$$

Since $\vec{v} \neq 0$, we have that $\overline{\vec{v}}^\top \vec{v} = \|\vec{v}\|^2 > 0$. Thus, $\lambda = \overline{\lambda}$ and consequently $\lambda \in \mathbb{R}$.

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.

**Solution:** The proof here also has this nice recursive character to it that goes along well with what you have learned in our sister courses 61ab.

You will get more practice doing inductive proofs in our successor course 70.
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 
\[
A = \begin{bmatrix}
1 & -1 & 1 \\
1 & 1 & -1 \\
1 & 1 & 1 \\
\end{bmatrix}
\]

To find the Moore-Penrose pseudoinverse we start by calculating the SVD of \( A \). That is to say, calculate \( U, \Sigma, V \) such that 
\[
A = U \Sigma V^\top
\]
where \( U \) and \( V \) are orthonormal matrices.
Here we will give you that the decomposition of \( A \) is:
\[
A = \begin{bmatrix}
\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\
-\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\
\end{bmatrix}
\begin{bmatrix}
2 & 0 & 0 \\
0 & \sqrt{2} & 0 \\
0 & \sqrt{2} & 0 \\
\end{bmatrix}
\begin{bmatrix}
0 & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\
\frac{1}{\sqrt{2}} & 0 & 0 \\
0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\
\end{bmatrix}
\]

where:
\[
U = \begin{bmatrix}
\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\
-\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\
\end{bmatrix}
\]
\[
\Sigma = \begin{bmatrix}
2 & 0 & 0 \\
0 & \sqrt{2} & 0 \\
0 & \sqrt{2} & 0 \\
\end{bmatrix}
\]
\[
V^\top = \begin{bmatrix}
0 & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\
\frac{1}{\sqrt{2}} & 0 & 0 \\
0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\
\end{bmatrix}
\]

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.

**Solution:** Though you did not have to do any work for deriving the SVD the following solutions will walk you through how to solve for the SVD:
\[
A = U \Sigma V^\top
\]
\[ AA^\top = \begin{bmatrix} 3 & -1 \\ -1 & 3 \end{bmatrix}. \]

Which has characteristic polynomial \( \lambda^2 - 6\lambda + 8 = 0 \), producing eigenvalues 4 and 2. Solving \( Av = \lambda_i v \) produces eigenvectors \( \left[ \frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}} \right]^\top \) and \( \left[ \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}} \right]^\top \) associated with eigenvalues 4 and 2 respectively. The singular values are the square roots of the eigenvalues of \( AA^\top \), so

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

and

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

We can then solve for the \( \vec{u} \) vectors using \( A^\top \vec{u}_i = \sigma_i \vec{v}_i \), producing \( \vec{v}_1 = [0, -\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}]^\top \) and \( \vec{v}_2 = [1, 0, 0]^\top \). The last \( \vec{v} \) must be orthonormal to the other two, so we can pick \( [0, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}]^\top \).

The SVD is:

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

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

\[ A\vec{x} = U\Sigma V^\top \vec{x} = \vec{y}. \]

Observe that \( V^\top \vec{x} \) rotates the vector, \( \Sigma \) 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 pseudoinverse.

**If \( U \) “rotates” the vector \( \left( \Sigma V^\top \right) \vec{x} \), what matrix can we derive that will undo the rotation?**

**Solution:** By orthonormality, we know that \( U^\top U = UU^\top = I \). Therefore, \( U^\top \) undoes the rotation.

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

**Solution:** If you observe the equation:

\[ \Sigma \vec{x} = U^\top \vec{y} = \vec{\tilde{y}}, \]

you can see that \( \sigma_i x_i = \tilde{y}_i \) for \( i = 1, ..., m \), which means that to obtain \( x_i \) from \( y_i \), we need to multiply \( y_i \) by \( \frac{1}{\sigma_i} \). For any \( i > m \), the information in \( x_i \) is lost by multiplying with 0. If the corresponding \( \tilde{y}_i \neq 0 \), there is no way of solving this equation. No solution exists, and we have to accept an approximate solution. If the corresponding \( \tilde{y}_i = 0 \), then any \( x_i \) would still work. Either way, it is reasonable to just say \( x_i \) is 0 in the case that \( \sigma_i = 0 \). That’s why we can legitimately pad 0s in the bottom of \( \Sigma \) given below:
If $\Sigma = \begin{bmatrix} \sigma_1 & 0 & 0 & 0 & 0 & \ldots & 0 \\ 0 & \sigma_2 & 0 & 0 & 0 & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \sigma_m & 0 & \ldots & 0 \end{bmatrix}$ then $\tilde{\Sigma} = \begin{bmatrix} \frac{1}{\sigma_1} & 0 & \ldots & 0 \\ 0 & \frac{1}{\sigma_2} & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \frac{1}{\sigma_m} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 0 \end{bmatrix}$.

This $\tilde{\Sigma}$ matrix undoes the effect of $\Sigma$ where that effect can be undone. If we happen to have non-zero singular values up to $\sigma_k$, and zeroes after, we would have the following $\tilde{\Sigma}$.

$$\tilde{\Sigma} = \begin{bmatrix} \frac{1}{\sigma_1} & 0 & \ldots & 0 & 0 & \ldots & 0 \\ 0 & \frac{1}{\sigma_2} & \ldots & 0 & 0 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 1/\sigma_k & 0 & \ldots & 0 \\ 0 & 0 & \ldots & 0 & 0 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 0 & 0 & \ldots & 0 \end{bmatrix}_{n \times m} \quad (37)$$

For the specific matrix in this problem,

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

we have

$$\tilde{\Sigma} = \begin{bmatrix} \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 \end{bmatrix}.$$

(c) **Derive a matrix that would "unrotate" by $V^\top$.**

**Solution:** By orthonormality, we know that $V^\top V = V V^\top = I$. Therefore, $V$ undoes the rotation.

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

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

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.

**Solution:** We can use the unrotation and unscaling matrices we derived above to "undo" the effect of $A$ and get the required solution. Of course, nothing can possibly be done for the information that was destroyed by the nullspace of $A$ — there is no way to recover any component of the true $\tilde{x}$ that was in the nullspace of $A$. However, we can get back everything else.
\[
\vec{y} = A\vec{x} = U\Sigma V^\top \vec{x}
\]

Therefore, we have \(A^\dagger = V\tilde{\Sigma}U^\top\), where \(\tilde{\Sigma}\) is given in (c).

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

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

**Solution:** From the above, we have the solution given by:

\[
\vec{x} = A^\dagger\vec{y} = V\tilde{\Sigma}U^\top \vec{y}
\]

\[
= \begin{bmatrix}
0 & \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\
-\frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} & 0 \\
\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 & 0
\end{bmatrix}
\begin{bmatrix}
2 \\
4
\end{bmatrix}
\]

\[
= \begin{bmatrix}
\frac{3}{2} \\
\frac{1}{2} \\
\frac{1}{2}
\end{bmatrix}
\]

Therefore, a reasonable solution to the system of equations is:

\[
\vec{x} = \begin{bmatrix}
\frac{3}{2} \\
\frac{1}{2} \\
-\frac{1}{2}
\end{bmatrix}
\]

(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 \(\vec{z}\) is the pseudo-inverse solution to \(A\vec{x} = \vec{y}\), then \(\|\vec{z}\| \leq \|\vec{z}\|\) 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.

**Solution:**

Since \(\vec{x}\) is the pseudoinverse solution, we know that,

\[
\vec{x} = V\tilde{\Sigma}U^\top \vec{y}
\]

Let us write down what \(\vec{x}\) is with respect to the orthonormal basis formed by the columns of \(V\). We call this quantity \(\vec{x}_V\). Let there be \(k\) non-zero singular values. The following expression comes from
expanding the matrix multiplication.

\[ \vec{x}_V = V^T \vec{x} = V^T A^\dagger \vec{y} = V^T V \Sigma U^T \vec{y} = \Sigma U^T \vec{y} \]

\[ = \begin{bmatrix} \frac{\vec{u}_1^T \vec{y}}{\sigma_1} & \frac{\vec{u}_2^T \vec{y}}{\sigma_2} & \ldots & \frac{\vec{u}_k^T \vec{y}}{\sigma_k} & 0 & \ldots & 0 \end{bmatrix}^T \]

The \( n - k \) zeros at the end come from the fact that there are only \( k \) non-zero singular values. Therefore, by construction, \( \vec{x} \) is a linear combination of the first \( k \) columns of \( V \).

For any other \( \vec{z} \) that is also a solution to the original problem, we have:

\[ A \vec{z} = U \Sigma V^T \vec{z} = U \Sigma \vec{z}_V = \vec{y} \]

\[ \implies \Sigma \vec{z}_V = U^T \vec{y} \]

\[ \begin{bmatrix} \sigma_1 \vec{z}_{V,1} \\ \vdots \\ \sigma_k \vec{z}_{V,k} \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} \vec{u}_1^T \vec{y} \\ \vdots \\ \vec{u}_k^T \vec{y} \\ \vec{u}_{k+1}^T \vec{y} \\ \vdots \\ \vec{u}_m^T \vec{y} \end{bmatrix} \]

\( \vec{z}_V \) is the vector of coordinates of \( \vec{z} \) in the \( V \) basis, which comes from \( \vec{z} = V \vec{z}_V \). If we apply the “unrotation” \( U^T \) to both \( U \Sigma \vec{z}_V \) and \( \vec{y} \), and see that the first \( k \) elements of \( \Sigma \vec{z}_V \) are the entries of \( \vec{z}_V \) scaled by the non-zero \( \sigma_i \), with the remaining \( m - k \) zeros. This also points out that for solvability of \( A \vec{x} = \vec{y} \), we must have \( \vec{u}_\ell^T \vec{y} = 0 \) for \( \ell = k + 1, \ldots, m \).

Using the idea of “unscaled” for the first \( k \) elements (where the unscaling is invertible) we see that the first \( k \) elements of \( \vec{z}_V \) must be identical to those first \( k \) elements of \( \vec{z}_V \).

However, since the information for the last \( n - k \) elements of \( \vec{z}_V \) is lost by multiplying by the last \( n - k \) zero vector columns \((\vec{0} \in \mathbb{R}^m)\) in \( \Sigma \), any of the \( k + 1 \)-th values and beyond of \( \vec{z}_V \), namely \( \vec{z}_{V,\ell} = \alpha_\ell \) for \( \ell = k + 1, \ldots, m \), are unconstrained as weights on the last part of the \( V \) basis. These are the weights on the basis for the nullspace of \( A \). Therefore,

\[ \vec{z}_V = \begin{bmatrix} \frac{\vec{u}_1^T \vec{y}}{\sigma_1} & \frac{\vec{u}_2^T \vec{y}}{\sigma_2} & \ldots & \frac{\vec{u}_k^T \vec{y}}{\sigma_k} & \alpha_{k+1} & \alpha_{k+2} & \ldots & \alpha_n \end{bmatrix}^T \]

Now, since the columns of \( V \) are orthonormal, observe that,

\[ \| \vec{x} \|^2 = \| V \vec{x}_V \|^2 = \| \vec{x}_V \|^2 = \sum_{i=1}^{k} \left| \frac{\vec{u}_i^T \vec{y}}{\sigma_i} \right|^2 \]

and that,

\[ \| \vec{z} \|^2 = \| V \vec{z}_V \|^2 = \| \vec{z}_V \|^2 = \sum_{i=1}^{k} \left| \frac{\vec{u}_i^T \vec{y}}{\sigma_i} \right|^2 + \sum_{i=k+1}^{n} |\alpha_i|^2 \]
Therefore,
\[ \| \vec{z} \|^2 = \| \vec{x} \|^2 + \sum_{i=k+1}^{n} |\alpha_i|^2 \]

This tells us that,
\[ \| \vec{z} \| \geq \| \vec{x} \| \]

and consequently, \( \vec{x} \) is optimal.

(g) Consider a generic wide matrix \( A \). We know that \( A \) can be written using \( A = U\Sigma V^\top \) where \( U \) and \( V \) each are the appropriate size and have orthonormal columns, while \( \Sigma \) 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** \( A^\dagger = A^\top (AA^\top)^{-1} \).

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

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

**Solution:** We just substitute in to see what happens:
\[
A^\top (AA^\top)^{-1} = (U\Sigma V^\top)^\top (U\Sigma V^\top (U\Sigma V^\top)^\top)^{-1} \\
= V\Sigma^\top U^\top (U\Sigma V^\top V\Sigma^\top U^\top)^{-1} \\
= V\Sigma^\top U^\top (U(\Sigma\Sigma^\top)U^\top)^{-1} \\
= V\Sigma^\top U^\top U(\Sigma\Sigma^\top)^{-1}U^\top \\
= V\Sigma^\top (\Sigma\Sigma^\top)^{-1}U^\top.
\]

Note that by assuming the invertibility of \( AA^\top \), we know that \( A^\top \) must have linearly independent columns, or equivalently that \( A \) has linearly independent rows. This implies that \( m \times n \) matrix \( A \) has \( m \) non-zero singular values. At this point, we are almost done in reaching \( A^\dagger = V\Sigma U^\top \). We have the leading \( V \) and the ending \( U^\top \). All that we need to do is multiply out the diagonal matrices in the middle.

\[
\Sigma^\top (\Sigma\Sigma^\top)^{-1} = \begin{bmatrix} \sigma_1 & 0 & 0 & 0 & \ldots & 0 \\ 0 & \sigma_2 & 0 & 0 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \sigma_m & \ldots & 0 \\ \end{bmatrix}^\top \begin{bmatrix} \sigma_1 & 0 & 0 & 0 & \ldots & 0 \\ 0 & \sigma_2 & 0 & 0 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \sigma_m & \ldots & 0 \\ \end{bmatrix}^{-1} \\
= \begin{bmatrix} \sigma_1 & 0 & 0 & 0 & \ldots & 0 \\ 0 & \sigma_2 & 0 & 0 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \sigma_m & \ldots & 0 \\ \end{bmatrix}^\top \begin{bmatrix} \sigma_1 & 0 & \ldots & 0 \\ 0 & \sigma_2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \sigma_m \\ \end{bmatrix}^{-1} \\
= \begin{bmatrix} \sigma_1 & 0 & 0 & 0 & \ldots & 0 \\ 0 & \sigma_2 & 0 & 0 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \sigma_m & \ldots & 0 \\ \end{bmatrix}^\top \begin{bmatrix} \sigma_1 & 0 & \ldots & 0 \\ 0 & \sigma_2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \sigma_m \\ \end{bmatrix}^{-1}
\]

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

In case you were wondering, the alternative form $A^\top (AA^\top)^{-1}$ can be found directly (without using the SVD) by first noticing that the columns of $A^\top$ are all orthogonal to the nullspace of $A$ by definition, and so we are justified in using them as the basis for the subspace in which we want to find the solution — a nonzero linear combination of these will output something non-zero if it goes through the matrix $A$. The columns of $AA^\top$ are the columns of where each of these basis vectors (columns of $A^\top$) end up through $A$. Inverting this tells us how to get to where we want. Once we know the coordinates in the basis of the columns of $A^\top$, we then multiply again by $A^\top$ to get our vector in the standard basis for $\mathbb{R}^n$.

Anyway, it is interesting to step back and see that at this point, between 16A and 16B, you now know two different ways to solve problems in which there are fewer linear equations than you have unknowns. You learned OMP in 16A which proceeded in a greedy fashion and basically tried to minimize the number of variables that it set to anything other than zero. And now you have learned the Moore-Penrose Pseudoinverse that finds the solution that minimizes the Euclidean norm.

Both of these, as well as least-squares, are ways to manifest the philosophical principle of Occam’s Razor algorithmically for learning. Occam’s Razor says “Namquam ponenda est pluralitas sine necessitate” (translation: don’t posit more than you need.) But there are two different ways to measure “more” — counting and weighing. Least squares (when we have more equations\(^1\) than variables) is on the path of weighing — the norm of the error is minimized. OMP iterates that to also follow the path of counting, where the number of nonzero variables corresponds to the things that are counted. The Moore-Penrose Pseudoinverse is fully in the path of weighing.

Anyway, both of these paths grow into major themes in machine learning (and signal processing generally), and both play a very important role in modern machine learning in particular. This is because in many contemporary approaches to machine learning, we try to learn models that have more parameters than we have data points.

---

\(^1\)In our successor course 70, you will learn an approach to the setting where you have more equations than variables that is on the path of counting — but that only works for matrices that have a very special structure. It is no coincidence that like least-squares, the problem reduces to solving a system of linear equations.
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:

$$\|\vec{x}\| \leq \|\vec{z}\|$$  \hspace{1cm} (50)

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.**

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

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix}, \quad C = \begin{bmatrix} 0 & 0 & 2 \\ 0 & 1 & 0 \\ 2 & 0 & 0 \end{bmatrix}, \quad \vec{y} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$  \hspace{1cm} (51)

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.

$$A = \begin{pmatrix} U_A \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \end{pmatrix} \begin{bmatrix} \sqrt{2} & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} V_A^T \begin{bmatrix} 0 & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{2} \end{bmatrix} \end{pmatrix}$$  \hspace{1cm} (52)

$$C = \begin{pmatrix} U_C \begin{bmatrix} -1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix} \end{pmatrix} \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} V_C^T \begin{bmatrix} 0 & 0 & -1 \\ -1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \end{pmatrix}$$  \hspace{1cm} (53)

$$AC = \begin{pmatrix} U_{AC} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \end{pmatrix} \begin{bmatrix} \sqrt{5} & 0 & 0 \\ 0 & 2 & 0 \end{bmatrix} \begin{pmatrix} V_{AC}^T \begin{bmatrix} 0 & 0 & 0 \\ \frac{1}{\sqrt{5}} & \frac{1}{\sqrt{5}} & 0 \\ 0 & 0 & \frac{1}{2} \end{bmatrix} \end{pmatrix}$$  \hspace{1cm} (54)

$$AC^{-1} = \begin{pmatrix} U_{AC^{-1}} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \end{pmatrix} \begin{bmatrix} \frac{1}{\sqrt{5}} & 0 & 0 \\ \frac{2}{\sqrt{5}} & \frac{1}{\sqrt{5}} & 0 \\ -\frac{2}{\sqrt{5}} & -\frac{1}{\sqrt{5}} & 0 \end{bmatrix} \begin{pmatrix} V_{AC^{-1}}^T \begin{bmatrix} 0 \\ \frac{1}{\sqrt{5}} & \frac{2}{\sqrt{5}} & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{pmatrix}$$  \hspace{1cm} (55)
Solution: In lecture and earlier problems on this HW, you solved a similar problem $A\vec{x} = \vec{y}$ such that $\vec{x}$ is a minimum norm solution: $\|\vec{x}\| \leq \|\vec{z}\|$ for any $\vec{z}$ that satisfies $A\vec{z} = \vec{y}$.

When you solved this problem, you computed the appropriate pseudoinverse to solve for $\vec{x}$. This was the Moore Penrose pseudo inverse — sometimes depicted as $A^\dagger$.

Seeing that we already know how to solve such problems, we can first try to reformulate the current problem: $A\vec{x} = \vec{y}$ such that $\|C\vec{x}\| \leq \|C\vec{z}\|$ for any vector $\vec{z}$ that satisfies $A\vec{z} = \vec{y}$. What is this something?

Originally, we had $A\vec{x} = \vec{y}$ and so in the changed variables, we have $\vec{x} = C^{-1}\vec{z}$ and so the constraint that needs to be satisfied is $AC^{-1}\vec{p} = \vec{y}$.

To solve this we proceed exactly like we did earlier and find the Moore-Penrose psuedo inverse of $AC^{-1}$:

$$\vec{x} = V_{\text{compact},AC^{-1}}\Sigma^{-1}_{\text{compact},AC^{-1}}U_{\text{compact},AC^{-1}}\vec{y}$$

where here, we need to be using the compact form of the SVD vis-a-vis $AC^{-1}$. Why compact? We need the $\Sigma$ matrix to be square so we can invert it. This just means that we drop the parts of $V^\top$ that are just a basis for the nullspace of $AC^{-1}$ — the last row. To be explicit, the compact SVD is:

$$AC^{-1} = \begin{pmatrix} U_{AC^{-1}} & \Sigma_{\text{compact},AC^{-1}} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \sqrt{5} & 0 \\ 0 & \frac{1}{2} \end{pmatrix} \begin{pmatrix} V_{\text{compact},AC^{-1}}^\top \\ 0 \end{pmatrix} = \begin{pmatrix} 1 & \sqrt{5} & 0 & 0 \\ \frac{\sqrt{5}}{2} & 0 & 0 & 0 \end{pmatrix}$$

Calculating this out, we get that $\vec{x} = \begin{pmatrix} \frac{2}{\sqrt{5}} \\ \frac{2}{5} \end{pmatrix}$.

However since the original question was to find $\vec{x}$ we have one more substitution to arrive at our final answer:

$$\vec{x} = C^{-1}\vec{x} = C^{-1}V_{\text{compact},AC^{-1}}\Sigma^{-1}_{\text{compact},AC^{-1}}U_{\text{compact},AC^{-1}}\vec{y}$$

$$= \begin{pmatrix} 2 \\ 0.8 \\ 0.2 \end{pmatrix}.$$
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 $\tilde{x}$ such that $\|\vec{x}\| = \|C\vec{x}\|$?)

Solution:

Now we have the condition where $C$ is a tall matrix with linearly independent columns. This means that $C$ itself is no longer invertible and we cannot just repeat the procedure done in the previous part of the problem. We don’t have access to a $C^{-1}$ and so need to stop and think. What we want is a square matrix $\tilde{C}$ that is invertible, and gives us the same norm to minimize. That is, we need $\|\tilde{C}\vec{x}\| = \|C\vec{x}\|$.

Writing this out, we see that since $\|C\vec{x}\|^2 = \vec{x}^\top C^\top C \vec{x}$, what we want is that $C^\top C = \tilde{C}^\top \tilde{C}$. Following the hint and using the compact-form SVD of $C = U_{\text{compact},C} \Sigma_{\text{compact},C} V_{\text{compact},C}^\top$ in which $\Sigma_{\text{compact},C}$ is square. So, $C^\top C = V_C \Sigma_{\text{compact},C}^2 V_C^\top$ since $U_{\text{compact},C}$ has orthonormal columns. This immediately suggests using $\tilde{C} = \Sigma_{\text{compact},C} V_C^\top$. And by construction $C^\top C = \tilde{C}^\top \tilde{C}$.

The only question now is whether $\tilde{C}$ is invertible. Because $C$ has linearly independent columns, it cannot have a nullspace. But we know from lecture that if $C^\top C \vec{v} = \vec{0}$, that indeed $C \vec{v} = \vec{0}$ and so $C^\top C$ also does not have a nullspace. So $C^\top C$ is invertible, and since $V_C \Sigma_{\text{compact},C}^2 V_C^\top$ is the diagonalization of $C^\top C$ by the basis $V_C$ of eigenvectors, $\Sigma_{\text{compact},C}$ is also invertible. The product of invertible matrices is invertible, and so indeed $\tilde{C}$ is invertible.

At this point, we have reduced this problem to what we did in the previous part. We just want to minimize $\|\tilde{C}\vec{x}\|$ over all $\vec{x}$ that satisfy $A\vec{x} = \vec{y}$. This is equivalent to minimizing $\|\tilde{x}\|$ over all $\tilde{x}$ that satisfy $A\tilde{C}^{-1}\tilde{x} = \tilde{y}$.

So in terms of an explicit procedure:

i. Compute the compact SVD of $C = U_{\text{compact},C} \Sigma_{\text{compact},C} V_{\text{compact},C}^\top$.

ii. Compute the matrix $\tilde{C} = \Sigma_{\text{compact},C} V_C^\top$.

iii. Compute the compact form SVD of the matrix $A\tilde{C}^{-1} = U \Sigma V^\top$.

iv. Compute the solution $\tilde{x} = \tilde{C}^{-1} V \Sigma^{-1} U^\top \tilde{y}$.

This comes from changing variables to $\tilde{x} = \tilde{C}^{-1}\tilde{x}$ and finding the minimum norm $\tilde{x}$ that works.

An alternative solution (that amounts to the same thing, effectively) exists where we use the pseudo-inverse of $C$ (i.e. the least-squares solution) and build our solution around that instead. Arguing why that works is a bit more involved.
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 \[
\begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}
\text{ and } 
\begin{bmatrix} 5 \\ -1 \\ 4 \end{bmatrix}
\] are orthogonal to each other.

Solution: Taking the inner product of the two vectors, we have

\[
\langle \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}, \begin{bmatrix} 5 \\ -1 \\ 4 \end{bmatrix} \rangle = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}^\top \begin{bmatrix} 5 \\ -1 \\ 4 \end{bmatrix} = 5 - 1 - 4 = 0.
\]

So the two columns 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 \( D \)?

Solution: We compute the norm for each column and divide each column by its norm to obtain matrix \( B \). Matrix \( D \) is formed by placing the norms on the diagonal.

\[
B = \begin{bmatrix} -\frac{1}{\sqrt{14}} & \frac{1}{\sqrt{3}} & \frac{5}{\sqrt{42}} \\ \frac{1}{\sqrt{14}} & \frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{42}} \\ \frac{1}{\sqrt{14}} & -\frac{1}{\sqrt{3}} & \frac{1}{\sqrt{42}} \end{bmatrix}
\]

\[
D = \begin{bmatrix} \sqrt{14} & 0 & 0 \\ 0 & \sqrt{3} & 0 \\ 0 & 0 & \sqrt{42} \end{bmatrix}
\]

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

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

Solution:

\[
A = BD = BD I = \begin{bmatrix} -\frac{1}{\sqrt{14}} & \frac{1}{\sqrt{3}} & \frac{5}{\sqrt{42}} \\ \frac{1}{\sqrt{14}} & \frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{42}} \\ \frac{1}{\sqrt{14}} & -\frac{1}{\sqrt{3}} & \frac{1}{\sqrt{42}} \end{bmatrix} \begin{bmatrix} \sqrt{14} & 0 & 0 \\ 0 & \sqrt{3} & 0 \\ 0 & 0 & \sqrt{42} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.
\]

Reordering the singular values and corresponding left and right singular vectors, we have the SVD:

\[
\begin{bmatrix} \frac{5}{\sqrt{42}} & -\frac{1}{\sqrt{14}} & \frac{1}{\sqrt{3}} \\ -\frac{1}{\sqrt{14}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{42}} \\ \frac{1}{\sqrt{42}} & -\frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{42}} \end{bmatrix} \begin{bmatrix} \sqrt{42} & 0 & 0 \\ 0 & \sqrt{14} & 0 \\ 0 & 0 & \sqrt{3} \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}.
\]
(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 \(U \Sigma V^\top\). Note the ordering of the singular values in \(\Sigma\) should be from the largest to smallest.

**HINT:** You don’t have to compute any eigenvalues for this. Some useful observations are that
\[
\begin{bmatrix} 3 & 4 \\ -4 & 3 \end{bmatrix} = 0, \quad \begin{bmatrix} 1 & -1 \\ 4 & 3 \end{bmatrix} = 0, \quad \| \begin{bmatrix} 3 \\ 4 \end{bmatrix} \| = \| \begin{bmatrix} -4 \\ 3 \end{bmatrix} \| = 5, \quad \| \begin{bmatrix} 1 \\ -1 \end{bmatrix} \| = \| \begin{bmatrix} 1 \\ 1 \end{bmatrix} \| = \sqrt{2}.
\]

**Solution:**
The singular value decomposition can be written in the form
\[
A = \sum_{i=1}^{2} \sigma_i \vec{u}_i \vec{v}_i^\top,
\]
with unit orthonormal vectors \(\{\vec{u}_i\}\) and \(\{\vec{v}_i\}\). From the given observations, we can see that the vectors we were provided are orthogonal, so we can just normalize them to get the desired answer. Taking it step by step:

\[
A = \frac{1}{\sqrt{50}} \begin{bmatrix} 3 \\ 4 \end{bmatrix} \begin{bmatrix} 1 & -1 \end{bmatrix} + \frac{3}{\sqrt{50}} \begin{bmatrix} -4 \\ 3 \end{bmatrix} \begin{bmatrix} 1 & 1 \end{bmatrix}, \quad (60)
\]
\[
= \frac{5}{\sqrt{50}} \begin{bmatrix} 3 \\ 4 \end{bmatrix} \begin{bmatrix} 1 & -1 \end{bmatrix} + \frac{3 \cdot 5}{\sqrt{50}} \begin{bmatrix} -4 \\ 3 \end{bmatrix} \begin{bmatrix} 1 & 1 \end{bmatrix}, \quad (61)
\]
\[
= \frac{5 \sqrt{2}}{\sqrt{50}} \begin{bmatrix} \frac{3}{5} \\ \frac{4}{5} \end{bmatrix} \begin{bmatrix} 1 & -1 \end{bmatrix} + \frac{3 \cdot 5 \sqrt{2}}{\sqrt{50}} \begin{bmatrix} -\frac{4}{3} \\ \frac{1}{3} \end{bmatrix} \begin{bmatrix} 1 \\ \sqrt{2} \end{bmatrix}, \quad (62)
\]
\[
= 1 \begin{bmatrix} \frac{3}{5} \\ \dfrac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 1 & -1 \end{bmatrix} + 3 \begin{bmatrix} -\frac{4}{3} \\ \frac{1}{3} \end{bmatrix} \begin{bmatrix} 1 \\ \dfrac{1}{\sqrt{2}} \end{bmatrix}. \quad (63)
\]

From this, we can derive
\[
\vec{u}_1 = \begin{bmatrix} -\frac{4}{3} \\ \frac{1}{\sqrt{2}} \end{bmatrix}, \quad \vec{u}_2 = \begin{bmatrix} \frac{3}{5} \\ \frac{1}{\sqrt{2}} \end{bmatrix}, \quad \vec{v}_1 = \begin{bmatrix} 1 \\ \sqrt{2} \end{bmatrix}, \quad \vec{v}_2 = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{bmatrix}
\]
and corresponding singular values \(\sigma_1 = 3, \sigma_2 = 1\) because we need to order them by size in decreasing order. This gives the singular value decomposition
\[
A = \begin{bmatrix} \frac{3}{5} \\ \frac{4}{5} \end{bmatrix} \begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ \sqrt{2} \\ \sqrt{2} \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{bmatrix}^\top.
\]
(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 \( A \)?

**Solution:** Since we have a square matrix, we will arbitrarily use \( A^\top A \) for our SVD:

\[
A^\top A = \begin{bmatrix} -1 & 1 \\ 4 & 4 \end{bmatrix} \begin{bmatrix} -1 & 4 \\ 1 & 4 \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 32 \end{bmatrix}
\]

Next, we find the eigenvalues of the above matrix.

\[
\det(A^\top A - \lambda I) = (2 - \lambda)(32 - \lambda) = 0
\]

Hence, the eigenvalues are \( \lambda_1 = 32 \) and \( \lambda_2 = 2 \), and the singular values are \( \sigma_1 = \sqrt{32} = 4\sqrt{2} \) and \( \sigma_2 = \sqrt{2} \).

Next, we find the right singular vectors (i.e. the columns of \( V \)). Finding \( \text{null}(A^\top A - \lambda_1 I) \) and \( \text{null}(A^\top A - \lambda_2 I) \) will give us \( \vec{v}_1 \) and \( \vec{v}_2 \) respectively.

Hence, \( \vec{v}_1 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \) and \( \vec{v}_2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \) (the eigenvectors are already normalized here).

Lastly, we find the right singular vectors (the columns of \( U \))

\[
\vec{u}_1 = \frac{1}{\sigma_1} Av_1 = \frac{1}{4\sqrt{2}} \begin{bmatrix} -1 & 4 \\ 1 & 4 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}
\]

Similarly, we get \( \vec{u}_2 = \begin{bmatrix} -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} \).

So the full SVD representation of \( A \) is given below

\[
A = \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 4\sqrt{2} & 0 \\ 0 & \sqrt{2} \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{64}
\]

Now that we have found the SVD of \( A \), we will find the eigenvalues and eigenvectors of \( A \). Let us start with the eigenvalues:

\[
\det(A - \lambda I) = (-1 - \lambda)(4 - \lambda) - 4 = \lambda^2 - 3\lambda - 8 = 0
\]

Using the quadratic formula, the eigenvalues are \( \lambda_1 = \frac{3 + \sqrt{41}}{2} \approx 4.7 \) and \( \lambda_2 = \frac{3 - \sqrt{41}}{2} \approx -1.7 \).
Since we already used \( \vec{v}_1, \vec{v}_2 \) for the SVD, let us denote the eigenvectors of \( A \) as \( \vec{r}_1, \vec{r}_2 \).

Finding \( \text{null}(A - \lambda_1 I) \) and \( \text{null}(A - \lambda_2 I) \) will give us \( \vec{r}_1 \) and \( \vec{r}_2 \) respectively.

Hence, the normalized eigenvectors of \( A \) are \( \vec{r}_1 \approx \begin{bmatrix} -0.98 \\ 0.17 \end{bmatrix} \) and \( \vec{r}_2 \approx \begin{bmatrix} -0.57 \\ -0.82 \end{bmatrix} \).

We notice that there is no 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.

**Solution:** If you filled out this form and \( \geq 70\% \) of each year (i.e., 70\% of the freshman year class and 70\% of the sophomore year class, ...) fill it out, everyone will receive some extra credit (to be specified later). The form will remain open for another week.

For self grades, if you fill out the form, mark yourself as 10 regardless of the precise time you’ve submitted the form.
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.

\[ x[i + 1] = ax[i] + bu[i] + w[i] \]  
\[ x[i + 1] = ax[i] + b_1 u_1[i] + b_2 u_2[i] + w[i] \]  
\[ x[i + 1] = a_1 x[i] + a_2 x[i - 1] + bu[i] + w[i] \]  

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.

\[ x[i + 1] = a_1 x[i] + a_2 x[i - 1] + \cdots + a_N x[i - (N - 1)] + b_1 u[i] + b_2 u[i - 1] + b_M u[i - (M - 1)] + w[i] \]  

Models that take the form in eq. (68) 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. (69).

\[ x[i + 1] = b_1 u[i] + b_2 u[i - 1] + b_M u[i - (M - 1)] + w[i] \]  

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. (69) 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 \( b_1, \ldots, b_M \) 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. (70)

\[
\begin{bmatrix}
 u[M'] & u[M' - 1] & \cdots & u[M' - M + 1] \\
 u[M' + 1] & u[M'] & \cdots & u[M' - M + 2] \\
 \vdots & \vdots & \ddots & \vdots \\
\end{bmatrix}
\begin{bmatrix}
 b_1 \\
 b_2 \\
 \vdots \\
 b_M \\
\end{bmatrix}
= \begin{bmatrix}
 x[M' + 1] \\
 x[M' + 2] \\
 \vdots \\
 x[L] \\
\end{bmatrix}
\tag{70}
\]

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?

**Solution:** For a successful implementation of code in part (a), we will see that as $M$ increases training error will keep decreasing, while validation error will reach a minimum and then increase. From the error plot, the minimum occurs around $M = 250$, so we can conclude $M = 250$ is the optimal model order.

(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:

\[
A = QR = \begin{bmatrix}
| & | & | & | \\
\vec{q}_1 & \vec{q}_2 & \cdots & \vec{q}_M \\
| & | & | & | \\
\end{bmatrix}
\begin{bmatrix}
\vec{q}_1^\top \vec{d}_1 & \vec{q}_1^\top \vec{d}_2 & \cdots & \vec{q}_1^\top \vec{d}_M \\
0 & \vec{q}_2^\top \vec{d}_2 & \cdots & \vec{q}_2^\top \vec{d}_M \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & \vec{q}_M^\top \vec{d}_M \\
\end{bmatrix}
\tag{71}
\]

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:

\[
A(A^\top A)^{-1}A^\top \vec{b} = QR((QR)^\top QR)^{-1}(QR)^\top \vec{b}
= QR(R^\top Q^\top QR)^{-1}R^\top Q^\top \vec{b}
= QR(R^\top R)^{-1}R^\top Q^\top \vec{b}
= QRR^{-1}(R^\top)^{-1}R^\top Q^\top \vec{b}
= QQ^\top \vec{b}
\]

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

\[
(A^\top A)^{-1}A^\top \vec{b} = ((QR)^\top QR)^{-1}(QR)^\top \vec{b}
= (R^\top Q^\top QR)^{-1}R^\top Q^\top \vec{b}
\]
\[(R^\top R)^{-1}R^\top Q^\top \vec{b} = R^{-1}Q^\top \vec{b}\]

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

\[R\vec{x} = Q^\top \vec{b}\]  \hspace{1cm} (72)

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 \(M\):

\[
Q_M R_M = \begin{bmatrix}
q_1 & q_2 & \cdots & q_M
\end{bmatrix}
\begin{bmatrix}
\vec{d}_1 & \vec{d}_2 & \cdots & \vec{d}_M
\end{bmatrix}
\]

(73)

The following is the QR decomposition of order \(M + 1\):

\[
Q_{M+1} R_{M+1} = \begin{bmatrix}
q_1 & q_2 & \cdots & q_M & q_{M+1}
\end{bmatrix}
\begin{bmatrix}
\vec{d}_1 & \vec{d}_2 & \cdots & \vec{d}_M & \vec{d}_{M+1}
\end{bmatrix}
\]

(74)

\[
= \begin{bmatrix}
Q_M & \vec{q}_{M+1}
\end{bmatrix}
\begin{bmatrix}
R_M & Q_{M+1}^\top \vec{d}_{M+1}
\end{bmatrix}
\]

(75)

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 \(R^{-1}_M\) 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 \(M = 1, 2, \ldots, M_{\text{max}}\), 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**
To speed up our computation, we want to see the relationship between $\vec{p}_M$ and $\vec{p}_{M+1}$ that satisfy $D_M\vec{p}_M = \vec{y}$ with $D_{M+1}\vec{p}_{M+1} = \vec{y}$.

The following two equations apply in terms of the QR decompositions of $D_M$ and $D_{M+1}$.

$$Q_M R_M \vec{p}_M = \vec{y}$$  \hspace{1cm} (76)

$$\begin{bmatrix} Q_M & \vec{q}_{M+1} \end{bmatrix} \begin{bmatrix} R_M & Q_M \vec{d}_{M+1} \\ \vec{0}^\top & \vec{q}_{M+1}^\top \vec{d}_{M+1} \end{bmatrix} \begin{bmatrix} \vec{p}_{M+1} \\ \vec{y}_{M+1} \end{bmatrix} = \begin{bmatrix} \vec{y} \\ \vec{y}_{M+1} \end{bmatrix}$$  \hspace{1cm} (77)

Processing the first equation, we have that $\vec{p}_M = R_M^{-1} Q_M^\top \vec{y}$.

Now, we can try to compute $\vec{p}_{M+1}$.

$$\begin{bmatrix} Q_M & \vec{q}_{M+1} \end{bmatrix} \begin{bmatrix} R_M & Q_M \vec{d}_{M+1} \\ \vec{0}^\top & \vec{q}_{M+1}^\top \vec{d}_{M+1} \end{bmatrix} \begin{bmatrix} \vec{p}_{M+1} \\ \vec{y}_{M+1} \end{bmatrix} = \begin{bmatrix} \vec{y} \\ \vec{y}_{M+1} \end{bmatrix}$$  \hspace{1cm} (78)

$$\begin{bmatrix} R_M & Q_M \vec{d}_{M+1} \\ \vec{0}^\top & \vec{q}_{M+1}^\top \vec{d}_{M+1} \end{bmatrix} \vec{p}_{M+1} = \begin{bmatrix} Q_M^\top \vec{y} \\ \vec{q}_{M+1}^\top \vec{y} \end{bmatrix}$$  \hspace{1cm} (79)

$$\vec{p}_{M+1} = R_M^{-1} \begin{bmatrix} Q_M^\top \vec{y} \\ \vec{q}_{M+1}^\top \vec{y} \end{bmatrix}$$  \hspace{1cm} (80)

It remains to calculate $R_{M+1}^{-1}$. We can do this by constructing a block matrix containing $R_{M}^{-1}$ and examining the other remaining quantities. Let us guess the following form for $R_{M+1}^{-1}$.

$$R_{M+1}^{-1} R_M = \begin{bmatrix} R_M^{-1} & \vec{a} \\ \vec{b}^\top & c \end{bmatrix} \begin{bmatrix} R_M & Q_M \vec{d}_{M+1} \\ \vec{0}^\top & \vec{q}_{M+1}^\top \vec{d}_{M+1} \end{bmatrix} = \begin{bmatrix} R_M^{-1} R_M + \vec{a} \vec{b}^\top & R_M^{-1} Q_M \vec{d}_{M+1} + \vec{a} \vec{q}_{M+1}^\top \vec{d}_{M+1} \\ \vec{b}^\top R_M + c \vec{b}^\top & \vec{b}^\top Q_M \vec{d}_{M+1} + c \vec{q}_{M+1}^\top \vec{d}_{M+1} \end{bmatrix} = I$$  \hspace{1cm} (81)

Since the above matrix is $I$, this implies that $\vec{b} = \vec{0}$, which then by the bottom right corner implies that $c = \frac{1}{\vec{q}_{M+1}^\top \vec{d}_{M+1}}$. The top right block must also be $\vec{0}$, so $\vec{a} = -\frac{1}{\vec{q}_{M+1}^\top \vec{d}_{M+1}} R_M^{-1} Q_M \vec{d}_{M+1}$. With these in mind, we can return back to computing $\vec{p}_{M+1}$.

$$\vec{p}_{M+1} = R_M^{-1} \begin{bmatrix} Q_M^\top \vec{y} \\ \vec{q}_{M+1}^\top \vec{y} \end{bmatrix}$$  \hspace{1cm} (82)

$$= \begin{bmatrix} R_M^{-1} & -\frac{1}{\vec{q}_{M+1}^\top \vec{d}_{M+1}} R_M^{-1} Q_M \vec{d}_{M+1} \\ \vec{0}^\top & \vec{q}_{M+1}^\top \vec{d}_{M+1} \end{bmatrix} \begin{bmatrix} Q_M^\top \vec{y} \\ \vec{q}_{M+1}^\top \vec{y} \end{bmatrix}$$  \hspace{1cm} (83)

$$= \begin{bmatrix} R_M^{-1} Q_M^\top \vec{y} \\ \vec{0}^\top Q_M^\top \vec{y} \end{bmatrix} + \begin{bmatrix} -\frac{1}{\vec{q}_{M+1}^\top \vec{d}_{M+1}} R_M^{-1} Q_M \vec{d}_{M+1} \\ \frac{1}{\vec{q}_{M+1}^\top \vec{d}_{M+1}} \end{bmatrix} \vec{q}_{M+1}^\top \vec{y}$$  \hspace{1cm} (84)

$$= \begin{bmatrix} R_M^{-1} Q_M^\top \vec{y} \\ \vec{0}^\top Q_M^\top \vec{y} \end{bmatrix}$$  \hspace{1cm} (85)
\[ \vec{r}_{M+1} := \begin{bmatrix} \frac{1}{\vec{q}_{M+1}^\top \vec{d}_{M+1}} R^{-1}_M Q^\top M \vec{d}_{M+1} \\ \frac{1}{\vec{q}_{M+1}^\top \vec{d}_{M+1}} \end{bmatrix} \]  

(86)

\[ \vec{p}_{M+1} = \begin{bmatrix} \vec{p}_M \\ 0 \end{bmatrix} + (\vec{q}_{M+1}^\top \vec{y}) \vec{r}_{M+1} \]  

(87)

We see how \( \vec{p}_{M+1} \) depends on the last column of \( R^{-1}_M, \vec{r}_{M+1} \). In fact, this is what lets us compute \( R^{-1}_{M+1} \) once and use column by column to generate the next set of parameters by adding a scaled version of the next column to the zero padded previous parameter estimate. You can see what this looks like in the code in the ipython notebook. For a successful implementation, the runtime for this function will be significantly less than both the least squares based method.
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 $30 \times 40$ 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 $91 \times 120$ 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 $91 \times 120$. 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

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

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

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

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 $\text{OMP}$ to run the naive OMP algorithm you learned in EECS16A. Read through the code and understand the function $\text{OMP}$.

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 $\text{smiley.png}$, generate an image of size $91 \times 120$ pixels of sparsity less than 400 and recover it using OMP with 6500 measurements.

Run the code $\text{rec} = \text{OMP}((\text{height}, \text{width}), \text{sparsity}, \text{measurements}, \mathbf{A})$ 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.

**Solution:**
The reconstructed image is the following.

It took about 9.5 seconds to run on a staff machine. This number will vary.

**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 $A$ 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 $A$ and subtract it from all entries of $A$ 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 $A$. 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 $100 \times 100$ 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 reconstruct the image. (Take a well deserved break, this may take upwards of ten minutes to run!)

**Solution:**
This answer will vary. It took about 345 seconds on a staff machine.

(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.

**Solution:**
See omp_speedup_16b_sol.ipynb.

The solution in the IPython notebook does Gram-Schmidt as it goes along and finds vectors. (The code is far from efficient in its implementation since it copies vectors too often.)
(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?

**Solution:**

See `omp_speedup_16b_sol.ipynb`.

This approach grabs many pixels at once. This saves a lot of effort in computing correlations over and over again. The threshold to decide how many pixels are safe to grab is chosen based on knowledge that derives from topics covered in 126 and touched upon in 70. Play with the number 6. For example, try to change it to 1 or 2. See what happens. Notice that it starts grabbing some false pixels.
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.