This homework is due Monday December 9, 2019, at 23:59.
Self-grades are due Wednesday December 11, 2019, at 23:59.

Submission Format
Your homework submission should consist of one file.

• hw14.pdf: A single PDF file that contains all of your answers (any handwritten answers should be scanned) as well as your IPython notebook saved as a PDF.
If you do not attach a PDF of your IPython notebook, you will not receive credit for problems that involve coding. Make sure that your results and your plots are visible. Assign the IPython printout to the correct problem(s) on Gradescope.

Submit the file to the appropriate assignment on Gradescope.

1. How Much Is Too Much?
When discussing circuits in this course, we only talked about resistor $I$-$V$ curves. There are many other devices that can be found in nature that do not have linear $I$-$V$ relations. Instead, $I$ is some general function of $V$, that is $I = f(V)$. Often times, the function describing the $I$-$V$ relationship is not known beforehand. The function $f$ is assumed to be a polynomial, and the parameters of $f$ (the coefficients for every power of $V$) are computed using least squares.
Throughout this problem, we are provided with $\vec{x}$, a set of voltage measurements, and $\vec{y}$, a set of current measurements.

(a) Let’s first try to model a resistor $I$-$V$ curve. Run the code in the attached IPython notebook. What is the degree of the polynomial that fits an ideal resistor $I$-$V$ curve? Play around with the degree in the IPython notebook and observe the best fit polynomial’s shape. Is the noise influencing the higher degree polynomials?

Solution:
According to Ohm’s law $I = \frac{1}{R}V$, the degree of the polynomial is one. As we increase the degree of the polynomial, the best fit polynomial starts to fluctuate, which means that it starts to fit the noise. The higher the degree of the polynomial, the more we are fitting the noise in the measurements.

(b) The attached IPython notebook provides functions `data_matrix`, `leastSquares` that allow you to fit polynomials of different degrees to the data provided. We also provide a function `cost` that computes the squared error of the fit. In the attached IPython notebook, plot the cost of various degree polynomials fitting to the measured $I$-$V$ data points for a resistor using the given `cost` function. The `cost` function returns $\|\vec{y} - A\hat{\vec{f}}\|^2$, i.e. the squared magnitude of the error vector.

$$
A = \begin{bmatrix}
1 & x_1 & x_1^2 & \cdots \\
1 & x_2 & x_2^2 & \cdots \\
1 & x_3 & x_3^2 & \cdots \\
\vdots & \ddots & \ddots & \ddots
\end{bmatrix}
$$
As seen above \( A \) is the appropriately sized matrix containing powers of the elements of \( \vec{x} \) and the vector \( \vec{f} \) contains entries \( f_n \) that are the coefficients for the \( n \)th power of the elements of \( \vec{x} \). Comment on the shape of the “Cost vs. Degree” graph. Do we want to choose a best fit polynomial of degree greater than one if the cost is lower than the polynomial of degree one? Should we choose the degree of the polynomial based on this graph? This question is meant to make you think, do not worry too much about getting a precise right answer here.

**Solution:**

We observe that the cost decreases as the degree of the polynomial increases. This is expected because as we increase the degree of the polynomial, we start to fit all of the data points in the data set, including the noise. This means that the best fit polynomial is “closer” to the data points, so the cost decreases. However, we know that according to Ohm’s law that the degree of the polynomial is one, so we should not pick the degree associated with the lowest cost. We should pick the polynomial of degree one instead.

In lecture we learned about dividing a data set into two sets. One set is the training set that is used to train the model, and the other data set is the test set where we test our model. Noise present in the training data set will then be different than that in the testing data set. If the best fit polynomial begins to fit noise in the training data set, its cost will therefore increase when applied the testing data set. For this case with noisy resistor measurements, greater-than-one degree best fit polynomials would give us lower cost on the training data set but higher cost on the testing data set when compared to the best fit polynomial of degree one.

2. **OMP Exercise**

Suppose we have a vector \( \vec{x} = [x_1 \ x_2 \ x_3 \ x_4]^T \in \mathbb{R}^4 \). The vector \( \vec{x} \) is related to the vector \( \vec{b} \) in the following way:

\[
\begin{bmatrix}
1 & 1 & 0 & -1 \\
1 & 0 & -1 & 0 \\
0 & 1 & 1 & 1
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4
\end{bmatrix}
\approx
\begin{bmatrix}
4 \\
6 \\
3
\end{bmatrix}
\]

For this undetermined and possibly noisy problem of finding \( \vec{x} \), assume that \( \vec{x} \) is sparse: it has 2 non-zero entries and 2 zero entries. Use Orthogonal Matching Pursuit to estimate \( x_1 \) to \( x_4 \).

(Note: Unlike previous examples you may have seen, we will not cross correlate \( \vec{b} \) with the columns of \( M \). Instead, we will just compute the inner product of \( \vec{b} \) with every column of \( M \).)

**Solution:**

Let \( \vec{c}_1 \) to \( \vec{c}_4 \) be the column vectors of \( M \). We first find the column vector in \( M \) that correlates most with \( \vec{b} \):

\[
\begin{align*}
\langle \vec{c}_1, \vec{b} \rangle &= 10 \\
\langle \vec{c}_2, \vec{b} \rangle &= 7 \\
\langle \vec{c}_3, \vec{b} \rangle &= -3 \\
\langle \vec{c}_4, \vec{b} \rangle &= -1
\end{align*}
\]
Thus, $\vec{c}_1$ is the best matching vector. Now we compute the projection of $\vec{b}$ onto $\vec{c}_1$, i.e. $\frac{\langle \vec{b}, \vec{c}_1 \rangle}{\langle \vec{c}_1, \vec{c}_1 \rangle} \vec{c}_1$ and subtract it from $\vec{b}$:

$$
\vec{b}' = \vec{b} - \frac{\langle \vec{b}, \vec{c}_1 \rangle}{\langle \vec{c}_1, \vec{c}_1 \rangle} \vec{c}_1 = \begin{bmatrix} -1 \\ 1 \\ 3 \\ \end{bmatrix}.
$$

Then we find the column that has the largest inner product with $\vec{b}'$:

$$
\langle \vec{c}_1, \vec{b}' \rangle = 0,
\langle \vec{c}_2, \vec{b}' \rangle = 2,
\langle \vec{c}_3, \vec{b}' \rangle = 2,
\langle \vec{c}_4, \vec{b}' \rangle = 4.
$$

We compute the projection of $\vec{b}'$ onto $\vec{c}_4$, i.e. $\frac{\langle \vec{b}', \vec{c}_4 \rangle}{\langle \vec{c}_4, \vec{c}_4 \rangle} \vec{c}_4$ and subtract it from $\vec{b}'$:

$$
\vec{b}'' = \vec{b}' - \frac{\langle \vec{b}', \vec{c}_4 \rangle}{\langle \vec{c}_4, \vec{c}_4 \rangle} \vec{c}_4 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ \end{bmatrix}.
$$

We know that $\vec{c}_1$ and $\vec{c}_4$ contribute most to $\vec{b}$, but we have just seen that no linear combination of $\vec{c}_1$ and $\vec{c}_4$ can form $\vec{b}$ because of noise in the measurements. Thus, we need to find the least squares solution. Let

$$
A = \begin{bmatrix}
| & | \\
\vec{c}_1 & \vec{c}_4 \\
| & |
\end{bmatrix} = \begin{bmatrix}
1 & -1 \\
1 & 0 \\
0 & 1 \\
\end{bmatrix},
$$

and the least squares formula gives:

$$
\begin{bmatrix} x_1 \\ x_4 \end{bmatrix} = (A^T A)^{-1} A^T \vec{b} = \begin{bmatrix} 6 \frac{1}{3} \\ 1 \frac{2}{3} \end{bmatrix}.
$$

Thus, $\vec{x} \approx \begin{bmatrix} 6 \frac{1}{3} \\ 1 \frac{2}{3} \end{bmatrix}$.

3. **Greedy Algorithm for Calculating Matrix Eigenvalues**

In the real world, it is not computationally practical to directly solve for the eigenvectors for large matrices as you might do for small matrices on paper. You would like to build an algorithm that sequentially computes the eigenvectors for a square symmetric matrix $Q = A^T A$ (Note: A symmetric matrix $P$ is a matrix such that its transpose is equal to itself, i.e. $P^T = P$. Component-wise this means $p_{ij} = p_{ji}$.)

To accomplish this we are given access to a function,

$$
(\vec{v}_1, \lambda_1) = f(Q),
$$

that returns the largest eigenvalue of the matrix $Q$ and the corresponding eigenvector. You do not need to know about the origins of the function, but if you are curious to learn more you can look up “Power
iteration”. We will use this function to build a greedy algorithm similar in the spirit of orthogonal matching pursuit that computes the eigenvectors and eigenvalues in descending order. By greedy algorithm we mean that we will choose eigenvectors one by one, at any time picking the eigenvector corresponding to the largest eigenvalue.

IMPORTANT: Throughout the problem, assume that the magnitude of each eigenvector is 1 (i.e. \( \|\vec{v}_i\|=1\)), that the eigenvalues are real, unique, and distinct, and that the eigenvalues are indexed in a descending order \( \lambda_1 > \lambda_2 > \cdots > \lambda_N > 0 \).

(a) Let
\[
A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}.
\] (2)

Calculate \( Q \) and \( Q^T \), where \( Q = A^T A \). Show that \( Q = Q^T \) (i.e. \( Q \) is symmetric).

Now, consider a general matrix \( A \in \mathbb{R}^{2 \times 2} \). Let \( Q = A^T A \). Show that \( Q \in \mathbb{R}^{2 \times 2} \) is symmetric.

Solution:
\[
Q = A^T A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 10 & 14 \\ 14 & 20 \end{bmatrix} \] (3)

\[
Q^T = (A^T A)^T = \left( \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \right)^T = \begin{bmatrix} 10 & 14 \\ 14 & 20 \end{bmatrix}^T = \begin{bmatrix} 10 & 14 \\ 14 & 20 \end{bmatrix} \] (4)

Both \( Q \) and \( Q^T \) are equal and therefore \( Q \) is symmetric.

In the general case:
\[
Q = A^T A = \begin{bmatrix} a_{11} & a_{12} \\ a_{12} & a_{22} \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = \begin{bmatrix} a_{11}^2 + a_{21}^2 & a_{11}a_{12} + a_{21}a_{22} \\ a_{12}a_{11} + a_{22}a_{21} & a_{12}^2 + a_{22}^2 \end{bmatrix} \] (5)

\[
Q^T = (A^T A)^T = \left( \begin{bmatrix} a_{11} & a_{12} \\ a_{12} & a_{22} \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \right)^T = \begin{bmatrix} a_{11}^2 + a_{21}^2 & a_{11}a_{12} + a_{21}a_{22} \\ a_{12}a_{11} + a_{22}a_{21} & a_{12}^2 + a_{22}^2 \end{bmatrix}^T \] (6)

\[
= \begin{bmatrix} a_{11}^2 + a_{21}^2 & a_{11}a_{12} + a_{21}a_{22} \\ a_{12}a_{11} + a_{22}a_{21} & a_{12}^2 + a_{22}^2 \end{bmatrix} \] (7)

Both \( Q \) and \( Q^T \) are equal and therefore \( Q \) is symmetric.

(b) Let us consider the matrix \( V \in \mathbb{R}^{N \times N} \):
\[
V = \begin{bmatrix} | & | & | \\ \vec{v}_1 & \vec{v}_2 & \ldots \vec{v}_N \end{bmatrix}.
\] (8)

Show that if \( \langle \vec{v}_i, \vec{v}_j \rangle = 0 \) for \( i, j = \{1, \ldots, N\} \) when \( i \neq j \) (all the columns of \( V \) are orthogonal to each other), then \( V^T V = I \).
Solution: 

\[ \mathbf{V}^T \mathbf{V} = \begin{bmatrix} \mathbf{v}_1^T \mathbf{v}_1 & \mathbf{v}_1^T \mathbf{v}_2 & \ldots & \mathbf{v}_1^T \mathbf{v}_n \\ \mathbf{v}_2^T \mathbf{v}_1 & \mathbf{v}_2^T \mathbf{v}_2 & \ldots & \mathbf{v}_2^T \mathbf{v}_n \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{v}_n^T \mathbf{v}_1 & \mathbf{v}_n^T \mathbf{v}_2 & \ldots & \mathbf{v}_n^T \mathbf{v}_n \end{bmatrix} = \mathbf{I} \] (9)

All \( \mathbf{v}_i^T \mathbf{v}_i = ||\mathbf{v}_i||^2 = 1 \) and when \( i \neq j \) all \( \mathbf{v}_i^T \mathbf{v}_j = 0 \) because the eigenvectors are orthogonal.

(c) Show that if the columns of matrix \( \mathbf{V} \) are orthogonal to each other, then the columns form a basis for \( \mathbb{R}^N \).

Solution: We want to show that the columns of \( \mathbf{V} \) form a basis for \( \mathbb{R}^N \). To show that the columns form a basis for \( \mathbb{R}^N \) we need to show two things:

• The columns must form a set of \( N \) linearly independent vectors.
• Any vector \( \mathbf{x} \in \mathbb{R}^N \) can be represented as a linear combination of the vectors in the set.

We already know we have \( N \) vectors, so first we will show they are linearly independent. We shall do this by showing that \( \mathbf{V} \mathbf{\beta} = \mathbf{0} \) implies that \( \mathbf{\beta} \) can be only \( \mathbf{0} \).

\[ \mathbf{V} \mathbf{\beta} = \mathbf{0} \] (10)

\[ \mathbf{\beta}_1 \mathbf{v}_1 + \ldots + \mathbf{\beta}_N \mathbf{v}_N = \mathbf{0} \] (11)

Then to exploit the properties of orthogonal vectors, we consider taking the inner product of each side of the above equation with \( \mathbf{v}_i \).

\[ \langle \mathbf{v}_i, \mathbf{\beta}_1 \mathbf{v}_1 + \ldots + \mathbf{\beta}_N \mathbf{v}_N \rangle = \langle \mathbf{v}_i, \mathbf{0} \rangle = 0 \] (12)

Now we apply the distributive property of the inner product and the definition of orthonormal vectors,

\[ \langle \mathbf{v}_i, \mathbf{\beta}_1 \mathbf{v}_1 \rangle + \ldots + \langle \mathbf{v}_i, \mathbf{\beta}_N \mathbf{v}_N \rangle = 0 \] (13)

\[ 0 + \ldots + \mathbf{\beta}_i \langle \mathbf{v}_i, \mathbf{v}_i \rangle + \ldots + 0 = 0 \] (14)

\[ 0 + \ldots + \mathbf{\beta}_i \mathbf{v}_i^T \mathbf{v}_i + \ldots + 0 = 0 \] (15)

Because \( \mathbf{v}_i^T \mathbf{v}_i = 1, \mathbf{\beta}_i = 0 \) for the equation to hold. Then, since this is true for all \( i \) from 1 to \( N \), all the elements of the vector \( \mathbf{\beta} \) must be zero (\( \mathbf{\beta} = \mathbf{0} \)). Because \( \mathbf{x} = \mathbf{0} \) implies \( \mathbf{\beta} = \mathbf{0} \), the columns of \( \mathbf{V} \) are linearly independent.

Now, we will show that any vector \( \mathbf{x} \in \mathbb{R}^N \) can be represented as a linear combination of the columns of \( \mathbf{V} \).

\[ \mathbf{x} = \mathbf{V} \mathbf{\beta} = \mathbf{\beta}_1 \mathbf{v}_1 + \ldots + \mathbf{\beta}_N \mathbf{v}_N \] (16)

Because we know that the \( N \) columns of \( \mathbf{V} \) are linearly independent, then there exists \( \mathbf{V}^{-1} \). Applying the inverse to the equation above,

\[ \mathbf{V}^{-1} \mathbf{V} \mathbf{\beta} = \mathbf{V}^{-1} \mathbf{x} \] (17)

\[ \mathbf{\beta} = \mathbf{V}^{-1} \mathbf{x}, \] (18)

we find that there exists a unique \( \mathbf{\beta} \) that allow us to represent any \( \mathbf{x} \) as a linear combination of the columns of \( \mathbf{V} \).
(d) Assume that the columns of matrix $V$ are orthogonal to each other for the remainder of the problem. Since the columns of $V$ form a basis, let $\vec{b} = \alpha_1 \vec{v}_1 + \cdots + \alpha_N \vec{v}_N$ for some scalars $\alpha_i$. Find $\langle \vec{v}_i, \vec{b} \rangle$.

**Solution:**

$$
\langle \vec{v}_i, \vec{b} \rangle = \vec{v}_i^T \vec{b} = \vec{v}_i^T \sum_{j=1}^{N} \alpha_j \vec{v}_j = 0 + \cdots + \alpha_i \vec{v}_i^T \vec{v}_i + \cdots + 0 = \alpha_i
$$

(e) Find the $\hat{x}$ that minimizes the following least squares problem:

$$
\min_{\vec{x}} ||V\vec{x} - \vec{b}||
$$

Let $\vec{e} = V\vec{x} - \vec{b}$. Also find $||\vec{e}||$ for the optimal $\hat{x}$.

*Hint:* Use what you proved in the earlier parts of this problem.

**Solution:**

We can apply the least squares formula to find:

$$
\hat{x} = (V^T V)^{-1} V^T \vec{b}
$$

$$
= V^T \vec{b}
$$

$$
= 0
$$

We know that $\vec{b}$ is in the column space of $V$ and that the norm of any vector is greater than or equal to zero. Therefore, the best possible solution to the least squares problem is zero. We are able to achieve zero when we pick $\hat{x}$ such that $V\hat{x} = \vec{b}$, and this is possible because $\vec{b}$ is in the column space of $V$.

(f) Let us define

$$
V_2 = \begin{bmatrix}
\vec{v}_2 & \cdots & \vec{v}_N
\end{bmatrix}
$$

(20)

where we have removed the first column of $V$. Note that $V_2 \in \mathbb{R}^{N\times(N-1)}$, and $V_2$ is not invertible. Assume $\vec{\alpha} = [\alpha_1 \ldots \alpha_N]^T$ are the same weights as in part (d). Find the $\hat{x}$ that minimizes the following least squares problem:

$$
\min_{\vec{x}} ||V_2\vec{x} - \vec{b}||.
$$

Let $\vec{e} = V_2\hat{x} - \vec{b}$. Also find $||\vec{e}||$. 

Solution: We apply the least squares formula to find:

\[
\hat{x} = (V_2^T V_2)^{-1} V_2^T \tilde{b}
\]

\[
= IV_2^T \tilde{b}
\]

\[
= V_2^T \hat{\alpha}
\]

\[
= \begin{bmatrix}
0 & 1 & 0 & \cdots & 0 \\
0 & 0 & 1 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
\alpha_1 \\
\alpha_2 \\
\vdots \\
\alpha_N
\end{bmatrix}
\]

\[
= \begin{bmatrix}
\alpha_2 \\
\alpha_3 \\
\vdots \\
\alpha_N
\end{bmatrix}
\]

\[
V_2 \hat{x} = \begin{bmatrix}
0 & 1 & 0 & \cdots & 0 \\
0 & 0 & 1 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
\alpha_1 \\
\alpha_2 \\
\alpha_3 \\
\vdots \\
\alpha_N
\end{bmatrix}
= (\alpha_2 \tilde{v}_2 + \alpha_3 \tilde{v}_3 + \cdots + \alpha_N \tilde{v}_N)
\]

\[
\| \tilde{e} \| = \| V_2 \hat{x} - \tilde{b} \|
\]

\[
= \| (\alpha_2 \tilde{v}_2 + \alpha_3 \tilde{v}_3 + \cdots + \alpha_N \tilde{v}_N) - (\alpha_1 \tilde{v}_1 + \alpha_2 \tilde{v}_2 + \cdots + \alpha_N \tilde{v}_N) \|
\]

\[
= \| (\alpha_2 \tilde{v}_2 + \alpha_3 \tilde{v}_3 + \cdots + \alpha_N \tilde{v}_N) - (\alpha_1 \tilde{v}_1) \|
= |\alpha_1|
\]

(g) Consider a matrix \( Q \) given as:

\[
Q = V \Lambda V^T = \sum_{i=1}^{N} \lambda_i \tilde{v}_i \tilde{v}_i^T
\]

(21)

where \( \Lambda = \begin{bmatrix}
\lambda_1 & 0 & 0 & \cdots & 0 \\
0 & \lambda_2 & 0 & \cdots & 0 \\
0 & 0 & \lambda_3 & \cdots & 0 \\
\vdots & \vdots & \ddots & \ddots & \vdots \\
0 & 0 & 0 & \cdots & \lambda_N
\end{bmatrix}
\]

(22)

It turns out that the eigenvectors of \( Q \) are given by the columns of \( V \). Try to prove this yourself.
Consider:

\[ Q\vec{v}_1 = \mathbf{V}\Lambda\mathbf{V}^T\vec{v}_1 \]

\[ = \mathbf{V}\Lambda \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \]

\[ = \mathbf{V} \begin{bmatrix} \lambda_1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \]

\[ = \lambda_1 \vec{v}_1 \]

Can you justify why each of the steps in the proof above is correct?

Solution:

\[ Q\vec{v}_1 = \mathbf{V}\Lambda\mathbf{V}^T\vec{v}_1 \]

\[ = \mathbf{V}\Lambda \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \] (The vector \(\vec{v}_1\) is orthogonal to all other vectors \(\vec{v}_i\) for \(i \neq 1\), and \(\vec{v}_1^T\vec{v}_1 = 1\).)

\[ = \mathbf{V} \begin{bmatrix} \lambda_1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \] (The first column of \(\Lambda\) is extracted as a result from the matrix multiplication.)

\[ = \lambda_1 \vec{v}_1 \] (The first column of \(\mathbf{V}\) multiplied by \(\lambda_1\) remains as a result from the matrix multiplication.)

(h) Now consider \(Q_2 = Q - \lambda_1 \vec{v}_1\vec{v}_1^T\). Thus, \(Q_2\) represents \(Q\) after the component associated with direction \(\vec{v}_1\) is removed. Show that \(\vec{v}_1\) is in the null space of \(Q_2\), and \(\vec{v}_2\) to \(\vec{v}_N\) are eigenvectors of \(Q_2\).

Hint: Can you write \(Q_2\) using Eq. (27)?

Solution: To show that \(\vec{v}_1\) is in the nullspace of \(Q_2\), we want to show that \(Q_2\vec{v}_1 = \vec{0}\).

\[
Q_2 = Q - \lambda_1 \vec{v}_1\vec{v}_1^T \\
= \sum_{i=1}^{N} \lambda_i \vec{v}_i\vec{v}_i^T - \lambda_1 \vec{v}_1\vec{v}_1^T \\
= \sum_{i=2}^{N} \lambda_i \vec{v}_i\vec{v}_i^T \\
Q_2\vec{v}_1 = \sum_{i=2}^{N} \lambda_i \vec{v}_i\vec{v}_i^T \vec{v}_1 \\
\]
Since \( \vec{v}_1 \) is orthogonal to all of the other eigenvectors, \( \vec{v}_i^T \vec{v}_1 = 0 \) for all \( i \neq 1 \).

\[
Q_2 \vec{v}_1 = \sum_{i=2}^{N} \lambda_i \vec{v}_i = 0
\]

We’ve shown that \( \vec{v}_1 \) is in the null space of \( Q_2 \).

Now we show that \( \vec{v}_2 \) to \( \vec{v}_N \) are eigenvectors of \( Q_2 \).

\[
Q_2 \vec{v}_j = \sum_{i=2}^{N} \lambda_i \vec{v}_i^T \vec{v}_j \quad \text{for } j = 2, \ldots, N
\]

Since \( \vec{v}_j \) is orthogonal to all of the other eigenvectors, \( \vec{v}_i^T \vec{v}_j = 0 \) for all \( i \neq j \). Also, \( \vec{v}_j^T \vec{v}_j = 1 \) for \( i = j \).

The above expression becomes:

\[
Q_2 \vec{v}_j = \lambda_j \vec{v}_j \quad \text{for } j = 2, \ldots, N
\]

We have shown that \( \vec{v}_2 \) to \( \vec{v}_N \) are eigenvectors of \( Q_2 \).

(i) Recall the function that returns the \textbf{largest} eigenvalue and corresponding eigenvector of a matrix,

\[
(\vec{v}_1, \lambda_1) = f(Q).
\]  

(23)

Design an algorithm that returns a list of eigenvalues of matrix \( Q \) in descending order of values. You may assume that all the eigenvalues of \( Q \) are positive (\( > 0 \)). You are allowed to use the function defined in Eq. (23) that returns the largest eigenvalue and corresponding eigenvector and what you know from part (g).

Solution:

1. function OMP(Q):
   2. list_of_eigenvalues = []
   3. For i in range(0,N):
      4. \((\vec{v}, \lambda) = f(Q)\)
      5. list_of_eigenvalues.append(\(\lambda\))
      6. Q = Q - \(\lambda \vec{v} \vec{v}^T\)
   7. return list_of_eigenvalues

What is this algorithm above doing? It is using the function that returns the largest eigenvalue (and corresponding eigenvector) and “peeling off” the eigenvectors one by one. After we get an eigenvector, we remove it from the matrix by \( Q - \lambda \vec{v} \vec{v}^T \). The remainder of the eigenvectors are preserved and in the next iteration of the loop, the eigenvector corresponding to the next largest eigenvector is picked and returned.

4. Sparse Imaging

Recall the imaging lab where we projected masks onto an object to scan it into 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
‘mask matrix’ $A$ to be invertible. In this problem, we have a 2D image $I$ of size $91 \times 120$ pixels for a total of 10920 pixels. We will explore how to reconstruct the image using only 6500 measurements.

The image is made up of mostly black pixels (represented by a zero) except for 476 white ones (represented by a one). In cases where there are a small number of non-black pixels, we can reduce the overall number of samples necessary using the orthogonal matching pursuit algorithm. This reduces the time required for scanning an image, a real-world concern for lengthy processes like MRI where people have to stay still while being imaged.

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. Say that we have an imaging mask $M_0$ of size $91 \times 120$. The measurements using this imaging mask can be represented as follows:

First, let us put every element in the matrix $I$ into a column vector $\vec{i}$ of length 10920. This operation is referred to as vectorization. Likewise, let us vectorize the mask $M_0$ to $\vec{m}_0$ which is a column vector of length 10920. Then the measurement using the mask $M_0$ 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

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

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

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

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. We will generate $\vec{A}$ so that the columns of $\vec{A}$ are approximately orthogonal with each other. The following question refers to the part of IPython notebook file accompanying this homework related to this question.

(a) In the IPython notebook, we call a function `simulate` that generates masks and the measurements. You can see the masks and the measurements in the IPython notebook file. Complete the function `OMP` that does the OMP algorithm described in lecture.

**Solution:**

See `sol14.ipynb`.

**Remark:** 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 $\vec{A}$ also have zero average value. To satisfy these conditions, they 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 $\vec{A}$ and subtract it from all entries of $\vec{A}$ 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 $\vec{A}$.

(b) Run the code $\text{rec} = \text{OMP}((\text{height, width}), \text{sparsity}, \text{measurements}, \vec{A})$ 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?
Solution:
The reconstructed image is the following.

(c) PRACTICE: We have supplied code that reads a PNG file containing a sparse image, takes measurements, and performs OMP to reconstruct the original image. An example input image file is also supplied together with the code. Recover smiley.png using OMP with 6500 measurements.
You can answer the following parts of this question in very general terms. Try reducing the number of measurements. Does the algorithm start to fail in recovering your sparse image? Why do you think it fails?

Solution:
The answer to this question depends on the size of your input image and the total number of non-zero pixels in it. In order to successfully recover the image, the number of measurements (masks used for imaging) needs to increase as the number of non-zero pixels increases. The reason is that as we have more nonzero pixels, they start to contribute to our measurements. In terms of the example given in the lecture, it is like having more smart devices trying to transmit their messages at the same time. In order to better distinguish these contributions, we need to take more measurements with different masks.

5. Trolls Revisited
You decided to skip EECS 16A lecture last week to head off to Thanksgiving early. You left a microphone in the lecture hall to capture the audio of the lecture, being a good student and not wanting to fall behind.

(a) Open the IPython notebook and run the first cell to listen to the recording. Can you hear what Prof. Ranade is saying in lecture clearly? (Hint: This is a yes or no question, don’t overthink it!)

Solution: No, you cannot. There is too much noise.

(b) You decide to build a model to solve this problem. You say the recording \( \tilde{r} \) is given by

\[
\tilde{r} = \alpha \tilde{l} + \tilde{n}.
\]  

(24)

where \( \tilde{l} \) is the true lecture and \( \tilde{n} \) is the interference played by some other professors. \( \alpha \) is a scalar.

To understand the behavior of this system, you go to the lecture microphone and play a known other lecture \( \tilde{l}_1 \), and you use your microphone to record

\[
\tilde{r}_1 = \alpha \tilde{l}_1 + \tilde{n}.
\]  

(25)
The $\alpha$ and $\vec{n}$ in (25) are the same as in (24), since the microphone location has not changed and the same interference has been playing throughout.

How could you recover $\alpha$ and $\vec{n}$ using the model (25) and assuming you knew $\vec{l}_1$ and $\vec{r}_1$? Then fill in the blanks to implement your approach in the IPython notebook.

*(You can assume that $\vec{l}_1$ is almost orthogonal to $\vec{n}$, since this tends to be true for long signals.)*

**Solution:** Pre-multiplying by $\vec{l}_i^T$, we obtain

$$\vec{l}_i^T \vec{r}_i = \alpha \vec{l}_i^T \vec{l}_i + \vec{l}_i^T \vec{n}_i.$$ 

But since they are orthogonal, $\vec{l}_i^T \vec{n}_i = 0$. Thus, rearranging, we find that

$$\alpha = \frac{\vec{l}_i^T \vec{r}_i}{\vec{l}_i^T \vec{l}_i}.$$ 

Substituting back into our original equation and rearranging, we obtain

$$\beta \vec{n}_i = \vec{r}_i - \frac{\vec{l}_i^T \vec{r}_i}{\vec{l}_i^T \vec{l}_i} \vec{l}_i.$$ 

In a real world system you could use this technique to “calibrate” a system — first play a known signal in place of $\vec{l}$ and then compute $\alpha$ and $\vec{n}$.

(c) It turns out that not just one, but multiple professors are interfering with the 16A lecture. Say we use the above technique to recover multiple potential interference signals $\vec{n}_i$.

The recorded signal $\vec{r}$ as well as each interference signal $\vec{n}_i$ is 1 million samples long, i.e. the vectors are of length 1 million. There are at most 4 different interference signals.

We can write the model:

$$\vec{r} = \vec{l} + \sum_{i=1}^{4} \beta_i \vec{n}_i,$$

You want to recover the four unknowns $\beta_i$ to see which interferences are being transmitted. How many equations do you have? Can you project $\vec{r}$ onto the space spanned by $\vec{n}_1$ to $\vec{n}_4$ to estimate the $\beta_i$’s even though you don’t know $\vec{l}$?

Implement this in the IPython notebook. Do you successfully recover the lecture signal $\vec{l}$? Or is it still too noisy?

*(You can assume that all the $\vec{n}_i$ and $\vec{l}$ are all mutually orthogonal.)*

**Solution:** You have 1 million equations and only four unknowns, so you can do this projection using least squares.

We can set up a least-squares problem that looks like

$$\vec{r} \approx \begin{bmatrix} \vec{n}_1 & \cdots & \vec{n}_4 \end{bmatrix} \begin{bmatrix} \beta_1 \\ \vdots \\ \beta_4 \end{bmatrix}.$$ 

By solving for the $\beta_i$ using least-squares and considering the residual, we can recover $\vec{l}$. Unfortunately, when you try doing so in the IPython notebook, you get back a noisy signal. As we see in part (d), this is due to an error in our model, even if we solved it correctly.

This goes to show that having the correct model of our physical system is extremely important if we wish to obtain meaningful results from our data.
(d) It turns out that your assumption from the previous part was not quite correct, because you forgot to account for the travel time/delay of the interference signals! A better model would be to write

\[ \vec{r} = \vec{l} + \sum_{i=1}^{4} \beta_i \vec{n}_i^{(k_i)} , \]

where the \( k_i \) represent the delay for each interference signal to reach your microphone.

Since you have now have to figure out the delays, and there are a million possible delays, you suddenly have a much larger number of unknowns. As a result, the least squares approach from earlier doesn’t quite work.

Describe how you could use OMP to recover \( \vec{l} \) from \( \vec{r} \) and the known \( \vec{n}_i \). Then fill in the blanks in the IPython notebook to do so. Now, does it work? If so, what is being said in the lecture?

**Solution:** Rather than just having the \( \vec{n}_i \) as columns in your matrix, you need to have all the \( \vec{n}_i^{(k_i)} \) for \( 0 \leq k_i < 10^6 \) and \( i \in \{1, 2, 3, 4\} \), as we are considering all possible shifts. If we put all these columns in some matrix \( A \), then we have the equation

\[ \vec{r} = A\vec{\beta} + \vec{l} . \]

By using OMP to determine the coefficients of \( \vec{\beta} \), we can identify and remove the four components of the form \( \beta_i \vec{n}_i^{(k_i)} \) from our recording, leaving us with \( \vec{l} \), our original lecture signal.

See the IPython solution notebook for an implementation. Notice that we do not explicitly build the matrix \( A \) as it would be too large - instead, we compute the inner product of our residual with each of the columns of \( A \) using the cross_correlation function.

Ultimately, we can recover Prof. Ranade’s lecture with only a little bit of residual noise, and can hear her talk about \( A^T A \) in the context of least squares, and describe how it may not be invertible.

6. Noise Canceling Headphones (PRACTICE)

In this problem, we will explore a common design for noise cancellation using noise-canceling headphones as an example application. We will work with the model shown in the figure below.

A music signal is generated at a speaker and transmitted to the listener’s ear. If there is noise in the environment (e.g. other people’s voices, a train going by), this noise signal will be superimposed on the music signal and the listener will hear both. In order to cancel the noise, we will try to record the noise and subtract it directly from the transmitted signal with the hope that we can achieve perfect cancellation of everything but the music. Since our system is imperfect, we’ll have to solve a least squares problem.

The gain blocks marked by \( \gamma \) (Greek “gamma”) represent scalar multiplication, and we will assume that they can take on any real number, positive or negative.

(a) First, consider a noise signal noted by \( \vec{n} \),

\[ \vec{n} = \begin{bmatrix} n_1 \\ n_2 \\ n_3 \\ n_4 \\ n_5 \end{bmatrix} \]

We can use three microphones to record this signal, Mic A, Mic B, and Mic C. Each microphone records the noise, but they each have their own characteristics. This means that they do not perfectly record the noise and that they are distinct recordings. Let \( \vec{r}_A \), \( \vec{r}_B \), and \( \vec{r}_C \) represent the noise that each microphone picks up:
We can arrange the recordings into a matrix $\mathbf{R}$ and the microphone gains, $\gamma$, into a vector $\vec{\gamma}$

$$
\vec{r}_A = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix}, \vec{r}_B = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \end{bmatrix}, \vec{r}_C = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \end{bmatrix}
$$

$\mathbf{R} = \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \\ a_4 & b_4 & c_4 \\ a_5 & b_5 & c_5 \end{bmatrix}, \vec{\gamma} = \begin{bmatrix} \gamma_A \\ \gamma_B \\ \gamma_C \end{bmatrix}$

For the system that is drawn in the figure above, write down the signal at the listener’s ear using matrix notation. It should include:

- the music signal $\vec{m}$
- the noise signal $\vec{n}$
- the matrix of recorded noise signals $\mathbf{R}$
- the microphone gain vector $\vec{\gamma}$

You can assume that the microphones do not pick up the music signal.
**Solution:** Let \( R \) be the matrix comprised of each microphone’s recording:

\[
R = \begin{bmatrix}
\vec{r}_A & \vec{r}_B & \vec{r}_C
\end{bmatrix} =
\begin{bmatrix}
a_1 & b_1 & c_1 \\
a_2 & b_2 & c_2 \\
a_3 & b_3 & c_3 \\
a_4 & b_4 & c_4 \\
a_5 & b_5 & c_5
\end{bmatrix}
\]

The gain stages are represented by the vector, \( \vec{\gamma} \):

\[
\vec{\gamma} = \begin{bmatrix}
\gamma_A \\
\gamma_B \\
\gamma_C
\end{bmatrix}
\]

If the music signal is represented by the vector \( \vec{m} \), the signal at the listener’s ear, denoted by \( \vec{s} \), can be represented as the sum:

\[
\vec{s} = \vec{m} + R \vec{\gamma} + \vec{n}
\]

(b) Ideally, we would want to have a signal at the ear that matches the original music signal perfectly. In reality, this is not possible, so we will aim to minimize the effect of the noise. What quantity would we need to minimize to make sure this happens? Write your answer in terms of the matrix \( R \), the vector of mic gains \( \vec{\gamma} \), and the noise vector \( \vec{n} \).

**Solution:** Based on part (a), we would like to make \( R \vec{\gamma} + \vec{n} = 0 \). This equality cannot always be true, so we aim to minimize the norm of this quantity. The minimization problem is written as:

\[
\min_{\vec{\gamma}} \left\| \begin{bmatrix}
a_1 & b_1 & c_1 \\
a_2 & b_2 & c_2 \\
a_3 & b_3 & c_3 \\
a_4 & b_4 & c_4 \\
a_5 & b_5 & c_5
\end{bmatrix} \begin{bmatrix}
\gamma_A \\
\gamma_B \\
\gamma_C
\end{bmatrix} + \begin{bmatrix}
n_1 \\
n_2 \\
n_3 \\
n_4 \\
n_5
\end{bmatrix} \right\| = \min_{\vec{\gamma}} \left\| \vec{\gamma} \right\| + \left\| \vec{n} \right\|
\]

(c) We can solve minimization problems by the least squares method. In effect, if we have a problem, \( \min_{\vec{x}} \left\| A\vec{x} - \vec{b} \right\| \), then the \( \vec{x} \) that solves this problem is:

\[
\vec{x} = (A^T A)^{-1} A^T \vec{b}
\]

Implement this least squares method in the IPython Notebook helper function `doLeastSquares`.

**Solution:** See the IPython notebook.
(d) For the given $\vec{n}$ and the recordings, $\vec{r}_A$, $\vec{r}_B$, $\vec{r}_C$, below, report the $\gamma$'s that minimize the effect of noise.

$$\vec{n} = \begin{bmatrix} 0.10 \\ 0.37 \\ -0.45 \\ 0.068 \\ 0.036 \end{bmatrix}, \vec{r}_A = \begin{bmatrix} 0 \\ -0.31 \\ -0.012 \\ 0.056 \end{bmatrix}, \vec{r}_B = \begin{bmatrix} 0 \\ -0.20 \\ 0.080 \\ 0.065 \end{bmatrix}, \vec{r}_C = \begin{bmatrix} 0 \\ 0.37 \\ -0.44 \\ 0.038 \end{bmatrix}$$

**Solution:** For the matrix $A$ in the `doLeastSquares` input, we use $R$:

$$R = \begin{bmatrix} \vec{r}_A & \vec{r}_B & \vec{r}_C \end{bmatrix} = \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \\ a_4 & b_4 & c_4 \\ a_5 & b_5 & c_5 \end{bmatrix}$$

For the vector $\vec{b}$ in the `doLeastSquares` input, we use $\vec{n}$:

$$\vec{n} = \begin{bmatrix} n_1 \\ n_2 \\ n_3 \\ n_4 \\ n_5 \end{bmatrix}$$

The result from `doLeastSquares` is:

$$\tilde{\gamma} = \begin{bmatrix} \gamma_1 \\ \gamma_2 \\ \gamma_3 \end{bmatrix} \approx \begin{bmatrix} -0.088 \\ -0.093 \\ -0.92 \end{bmatrix}$$

Note that the sign is negative because we need to cancel the noise!

The next few questions can be answered in the IPython notebook by running the associated cells.

(e) We can use this least squares solution to find the best $\gamma$ values for our algorithm for a given number of microphones. Follow the instructions in the IPython notebook to load a music signal and some noise signals. Listen to the music signal and the two noise signals. Which ones are full of static and which ones are not?

**Solution:** The music signal has a repeated musical clip. The first noise signal sounds like static. The second noise signal has static in addition to some people laughing and a train whistle. (Any comment on the main differences between these signals is a valid answer.)

(f) Use the IPython notebook to record the first noise signal using the `recordAmbientNoise` function and calculate a vector $\vec{\tilde{\gamma}}$. Create the noise cancellation signal by performing the multiplication $R\tilde{\gamma}$.

**Solution:** See the IPython notebook.

(g) Add the noise cancellation signal (with the correct sign) to the music signal to get the signal from the speaker and, finally, add the noise signal to the speaker signal. Play the noisy signal and the noise-cancelled signal. Can you hear a difference?

**Solution:** The noisy signal sounds like the music but with static. The noise-cancelled signal preserves the music, but the static seems to be have been eliminated.
(h) Try adding the other noise signal to the music signal without re-calculating new values for $\hat{y}$ (don’t solve the least squares problem again). Add the noise-cancelling signal to your speaker signal and add the noise as well. Comment on the quality of the resulting noise-cancelled signal. Is it perfect or are there artifacts?

**Solution:** The new noise-canceled signal seems to have the static noise removed. Some of the laughter and train whistle are removed, but there’s still some distortion where the laughter was present. (Any comment that notices distortion where the laughter or whistle are is a valid answer.)

7. **Homework Process and Study Group**

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

**Solution:**

I worked on this homework with...

I first worked by myself for 2 hours, but got stuck on problem 5, so I went to office hours on...

Then I went to homework party for a few hours, where I finished the homework.