EECS 16A Designing Information Devices and Systems I
Spring 2023 Optional Bonus Problems

These are some optional problems that you might find interesting. They are not required. They involve concepts that are in scope for the final, but you do not need to turn anything in. There are no self-grades for this homework.

1. Image Analysis

Learning Goal: *This problem introduces a method of fitting a non-linear model through a set of measured data points using the least squares method.*

Applications in medical imaging often require an analysis of images based on the image's pixels. For instance, we might want to count the number of cells in a given biological sample. One way to do this is to take a picture of the cells and use the pixels to determine their locations and how many there are. Automatic detection of shape is useful in image classification as well (e.g. consider a robot trying to find out autonomously where a mug is in its field of vision).

Let us focus back on the medical imaging scenario. You are interested in finding the exact position and shape of a cell in an image. You will do this by finding the **equation of the circle or ellipse** that bounds the cell relative to a given coordinate system in the image. Your collaborator uses edge detection techniques to find **a bunch of points that are approximately along the edge of the cell**. We assume that the origin of the coordinate system is in the center of the image with standard axes (x,y) and your collaborator gives you the following points that approximately bound the cell:

$(0.3,-0.69)$, $(0.5,0.87)$, $(0.9,-0.86)$, $(1,0.88)$, $(1.2,-0.82)$, $(1.5,0.64)$, $(1.8,0)$.

Recall that an equation of the form

$$a_1x^2+b_1xy+c_1y^2+d_1x+e_1y=1$$

can be used to represent an ellipse (if $b_1^2-4a_1c_1<0$), and an equation of the form

$$a_1(x^2+y^2)+d_1x+e_1y=1$$

is a circle if $d_1^2+e_1^2+4a_1>0$. Notice that the circle has fewer parameters.

You don't need to consider these constraints in your least squares setup, but you are encouraged to check whether your least squares solutions satisfy these constraints.

(a) How can you find the equation of a *circle* that surrounds the cell by fitting the data points? First, provide a setup and formulate a minimization problem to do this, i.e. a least squares problem minimizing the squared error $\left\|\mathbf{A}\vec{v}-\vec{b}\right\|^2$, where you attempt to find the **unknown coefficients** a_1**,** d_1**, and** e_1 from your data points. Here your unknown vector $\vec{v}=\begin{bmatrix}a_1\\d_1\\e_1\end{bmatrix}$. *Hint: The quantities* (x^2+y^2)*, x, and y can be thought of as known values calculated from your data points.*

You do not need to simplify the numerical values for the matrix elements; just writing out the matrix with numerical expressions will suffice.

Solution:

The setup is:

$$\min_{a_1,d_1,e_1} \left\| \begin{bmatrix} x^2+y^2 & x & y \\ \vdots & \vdots & \vdots \end{bmatrix} \begin{bmatrix} a_1 \\ d_1 \\ e_1 \end{bmatrix} - \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} . \right\|$$

We plug in numbers to get:

$$\min_{a_1,d_1,e_1} \left\| \begin{bmatrix} 0.5661 & 0.3 & -0.69 \\ 1.0069 & 0.5 & 0.87 \\ 1.5496 & 0.9 & -0.86 \\ 1.7744 & 1 & 0.88 \\ 2.1124 & 1.2 & -0.82 \\ 2.6596 & 1.5 & 0.64 \\ 3.24 & 1.8 & 0 \end{bmatrix} \begin{bmatrix} a_1 \\ d_1 \\ e_1 \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} . \right\|$$

(b) How can you find the equation of an ellipse (instead of a circle) that surrounds the cell? Provide a setup and formulate a minimization problem similar to that in part (a). Now the unknown vector $\vec{v}$ will be different from the earlier parts.

You do not need to simplify the numerical values for the matrix elements; just writing out the numerical expressions will suffice.

Solution:

The setup is:

$$\min_{a_1,b_1,c_1,d_1,e_1} \left\| \begin{bmatrix} x^2 & xy & y^2 & x & y \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \begin{bmatrix} a_1 \\ b_1 \\ c_1 \\ d_1 \\ e_1 \end{bmatrix} - \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} . \right\|$$

We then plug in values to get:

$$\min_{a_1,b_1,c_1,d_1,e_1} \left\| \begin{bmatrix} 0.09 & -0.207 & 0.4761 & 0.3 & -0.69 \\ 0.25 & 0.435 & 0.7569 & 0.5 & 0.87 \\ 0.81 & -0.774 & 0.7396 & 0.9 & -0.86 \\ 1 & 0.88 & 0.7744 & 1 & 0.88 \\ 1.44 & -0.984 & 0.6724 & 1.2 & -0.82 \\ 2.25 & 0.96 & 0.4096 & 1.5 & 0.64 \\ 3.24 & 0 & 0 & 1.8 & 0 \end{bmatrix} \begin{bmatrix} a_1 \\ b_1 \\ c_1 \\ d_1 \\ e_1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} . \right\|$$

(c) In the IPython notebook, run the widget and try to trace the outline of the example cell image with "measurements" by clicking the edge of the cell. Then try both fitting with circle and an ellipse. Which shape fits better, and why?

Solution: The ellipse is a better fit because it has more parameters, so the least squares technique can tune the parameters to be closer to the observations.

(d) Now let's try this ourselves with the data given in the problem. In the IPython notebook, write a short program that uses least-squares to fit a circle to the given points. A helper function `plot_circle` is provided. What is $\frac{\|\vec{e}\|}{N}$, where $\vec{e} = \mathbf{A}\vec{v} - \vec{b}$ and N is the number of data points? Plot your points and the best fit circle in IPython.

Solution:

See the IPython notebook.

The solution vector is:

$$\vec{v} = \begin{bmatrix} 4.87 \\ -7.89 \\ -0.23 \end{bmatrix}.$$

Thus, we would predict the equation of the circle to be: $4.87(x^2 + y^2) - 7.89x - 0.23y = 1$.
This gives the normalized error: $\frac{0.96}{7} = 0.137$.

(e) In the IPython notebook, write a short program that uses least-squares to fit an ellipse to the given points. A helper function `plot_ellipse` is provided. What is $\frac{\|\vec{e}\|}{N}$, where $\vec{e} = \mathbf{A}\vec{v} - \vec{b}$ and N is the number of data points? Now the unknown vector $\vec{v}$ will be different from the earlier parts. Plot your points and the best fit ellipse in IPython. How does this error compare to the one in the previous subpart?

Solution:

See the IPython notebook.

The solution vector is:

$$\vec{v} = \begin{bmatrix} 4.10 \\ 0.49 \\ 4.94 \\ -6.85 \\ -0.62 \end{bmatrix}.$$

We predict the general equation to be: $4.10x^2 + 0.49xy + 4.94y^2 - 6.85x - 0.62y = 1$.
This gives the normalized error: $\frac{0.090}{7} = 0.0128$.
The error is less than the circle fit, as predicted.

2. (OPTIONAL, PRACTICE) Labeling Patients Using Gene Expression Data

Learning Goal: *This problem aims to design a predictive model using the least squares method on a set of training data and test the efficacy of the model using a set of test data.*

Least squares techniques are useful for many different kinds of prediction problems. Numerous researchers have extensively further developed the core ideas that we have learned in class. These ideas are commonly used in machine learning for finance, healthcare, advertising, image processing, and many other fields. Here, we'll explore how least squares can be used for classification of data in a medical context.

Gene expression data of patients, along with other factors such as height, weight, age, and family history, are often used to predict the likelihood that a patient might develop a certain disease. This data can be combined into a vector that describes each patient. This vector is often referred to as a feature vector.

Many scientific studies examine mice to understand how gene expression relates to diabetes in humans. Studies have shown that the expression of the ***tomosinB*** and ***tsA*** genes are correlated to the onset of diabetes in mice. How can we predict whether or not a mouse will develop diabetes based on data about this expression as well as other factors of the mouse? We will use some (fake) data to explore this.

We are given feature vectors for each mouse as:

$$\begin{bmatrix} \text{age} \\ \text{weight} \\ \text{tomosinB} \\ \text{tsA} \\ \text{chnA} \end{bmatrix}$$

Age and weight in the vector above are represented by real numbers, while the presence or absence of the expression of the genes *tomosinB*, *tsA*, and *chnA* is captured by $+1$ and -1 respectively. For example, the vector $\begin{bmatrix} 2 & 20 & 1 & -1 & -1 \end{bmatrix}^T$ means a 2 month old mouse, that weighs 20 grams, expresses the genes *tomosinB*, but not *tsA* or *chnA*.

We would like the following expression to be **positive if the mouse has diabetes and negative if the mouse does not have diabetes**:

$$f(\text{age}, \text{weight}, \text{tomosinB}, \text{tsA}, \text{chnA}) = \alpha_1(\text{age}) + \alpha_2(\text{weight}) + \alpha_3(\text{tomosinB}) + \alpha_4(\text{tsA}) + \alpha_5(\text{chnA}). \tag{1}$$

(a) We wish to set up a linear model for the problem in the format $\mathbf{A}\vec{x} = \vec{b}$. Here, $\vec{b}$ will be a vector with $+1, -1$ entries where a 1 represents that the mouse is diabetic and -1 represents that the mouse is not diabetic. The feature vectors of each mouse will be included in each row of the matrix $\mathbf{A}$. For example, if the first row of $\mathbf{A}$ contains the data of mouse #1 and the first entry of $\vec{b}$ is $+1$, that means the mouse #1 has diabetes.

Set up the problem by writing $\mathbf{A}$, $\vec{x}$, and $\vec{b}$ in terms of the variables in the feature vectors, α_i, and any other variables you define. What are your unknowns?

Solution: Assume we have n mice. Define b_j as the indicator for whether the j-th mouse has diabetes. Each of the data variables will be indexed by $j = 1, \ldots, n$ for each of the n mice. The unknowns that we want to find are α_i, where $\{i = 1, \ldots, 5\}$. The vector $\vec{x}$ is the vector of unknowns $\vec{x} = \begin{bmatrix} \alpha_1 & \alpha_2 & \alpha_3 & \alpha_4 & \alpha_5 \end{bmatrix}^T$. The complete problem setup is:

$$\underbrace{\begin{bmatrix} \text{age}_1 & \text{weight}_1 & \text{tomosinB}_1 & \text{tsA}_1 & \text{chnA}_1 \\ \text{age}_2 & \text{weight}_2 & \text{tomosinB}_2 & \text{tsA}_2 & \text{chnA}_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \text{age}_n & \text{weight}_n & \text{tomosinB}_n & \text{tsA}_n & \text{chnA}_n \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \end{bmatrix}}_{\vec{x}} = \underbrace{\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}}_{\vec{b}}$$

(b) Training data is data that is used to develop your model. The matrix $\mathbf{A}$ and vector $\vec{b}$, provided in `gene_data_train.npy` and `diabetes_train.npy` respectively, represent the (fake) *training* data. Use the data to find the optimal model parameters $\alpha_1, \ldots, \alpha_5$ for the given data set. Find the optimal parameter values using least squares method and the provided IPython notebook.

Solution:

To solve for $\vec{x}$ in $\mathbf{A}\vec{x} = \vec{b}$ using the least squares technique, we find $(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\vec{b}$. The result is

$$\vec{x} = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \end{bmatrix} = \begin{bmatrix} 0.12131475 \\ -0.15253102 \\ -0.36111722 \\ -0.06427341 \\ 0.95936096 \end{bmatrix}$$

(c) Now it is time to use the model you have developed to make some predictions! It is interesting to note here that we are not looking for a real number to model whether each mouse has diabetes or not; we are looking for **a binary label**. Therefore, we will use the **sign of the expression from Equation (1)** to assign a ± 1 value to each mouse. Each mouse characteristics are represented by each row of `gene_data_test.npy`.

Predict whether each mouse with the characteristics in the *test* data set `gene_data_test.npy` will get diabetes. There are four mice/ rows in the test data set. Calculate the ± 1 prediction vector $\vec{b}_{calc}$ that shows the prediction whether a mouse will be diabetic or not. Observe the ± 1 vector $\vec{b}_{test}$ from `diabetes_test.npy` that indicates whether or not the mice *actually* have diabetes.

What is the prediction accuracy (number of correct predictions divided by total number of predictions) of your model?

Solution:

Using the values of α_i calculated in the previous part on the test data, we see that the prediction is $\vec{b} = \begin{bmatrix}1 & 1 & 1 & 1\end{bmatrix}^T$, which differs in the third entry of the given file `diabetes_test.npy`. Therefore the prediction accuracy is 75%.

3. 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 the with 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 $\|\vec{y} - \mathbf{A}\vec{f}\|^2$, i.e. the squared magnitude of the error vector.

$$\mathbf{A} = \begin{bmatrix} 1 & x_1 & x_1^2 & \dots \\ 1 & x_2 & x_2^2 & \dots \\ 1 & x_3 & x_3^2 & \dots \\ & \vdots & & \ddots \end{bmatrix}$$

As seen above $\mathbf{A}$ 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 nth 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.

(c) Now let’s put our hypothesis to the test by extrapolating some data! Using the provided code, graph the least squares polynomial fit IV characteristic for $V = [0,20]$ for both degree 1 and degree 15 polynomials. Which degree fits the extrapolated data better?

Solution: The degree 1 polynomial should roughly fit the linear function, with an approximate output of $I = 40A$ for $R = 0.5\Omega$ at $V = 20V$. On the other hand, the degree 15 polynomial goes completely wild outside of the voltage range it was trained on ($[-10,10]$), and its behavior depends on how the noise is distributed in the training data. This shows how important it is to avoiding model "overfitting", or fitting to noise features of a specific dataset – a common pitfall when training models with machine learning.

4. Constrained Least Squares Optimization

In this problem, we will guide you through solving the following optimization problem:

Consider a matrix $\mathbf{A} \in \mathbb{R}^{M\times N}$ where $M > N$ and all N columns are linearly independent. Determine a unit vector $\hat{\vec{x}}$ that minimizes $\|\mathbf{A}\vec{x}\|$, where $\|\cdot\|$ denotes the norm—that is,

$$\|\mathbf{A}\vec{x}\|^2 \triangleq \langle \mathbf{A}\vec{x}, \mathbf{A}\vec{x}\rangle = (\mathbf{A}\vec{x})^T\mathbf{A}\vec{x} = \vec{x}^T\mathbf{A}^T\mathbf{A}\vec{x}.$$

This is equivalent to solving the following optimization problem:

$$\min_{\vec{x}} \|\mathbf{A}\vec{x}\|^2 \quad \text{subject to the constraint} \quad \|\vec{x}\|^2 = 1.$$

This task may *seem* like solving a standard least squares problem $\mathbf{A}\vec{x} = \vec{b}$ where $\vec{b} = \vec{0}$, but it is different. As an example, notice $\vec{x} = \vec{0}$ is *not* a valid solution to our problem because the norm of the zero vector does not equal one. Our optimization problem is a least squares problem with a constraint—hence the term *constrained least squares optimization*. The constraint can be visualized as limiting the vector $\vec{x}$ to lie on a unit circle (radius of the circle is one) if $N = 2$ and on a unit sphere if $N = 3$.

Let $(\lambda_1, \vec{v}_1), \ldots, (\lambda_N, \vec{v}_N)$ denote the eigenpairs (i.e., eigenvalue/eigenvector pairs) of $\mathbf{A}^T\mathbf{A}$. Assume that the eigenvalues are all real, distinct and indexed in an descending fashion—that is,

$$\lambda_1 > \cdots > \lambda_N.$$

Assume, too, that each eigenvector has been normalized to have unit length—that is, $\|\vec{v}_k\| = 1$ for all $k \in \{1, \ldots, N\}$.

(a) Show that $\lambda_N > 0$, i.e. all the eigenvalues are strictly positive.

Hint: Consider $\|\mathbf{A}\vec{v}_N\|^2$. Write $\|\mathbf{A}\vec{v}_N\|^2$ in the matrix multiplication form, and use the eigenpair equation $\mathbf{A}^T\mathbf{A}\vec{v}_N = \lambda_N\vec{v}_N$.

Solution:

Consider $\|\mathbf{A}\vec{v}_N\|^2$ for eigenvector $\vec{v}_N$, with eigenvalue λ_N.

$$\begin{aligned} \|\mathbf{A}\vec{v}_N\|^2 &= \vec{v}_N{}^T(\mathbf{A}^T\mathbf{A}\vec{v}_N) \\ &= \vec{v}_N{}^T\lambda_N\vec{v}_N \\ \lambda_N &= \frac{\|\mathbf{A}\vec{v}_N\|^2}{\|\vec{v}_N\|^2} \end{aligned}$$

Therefore, $\lambda_N > 0$ since norms are positive if $\vec{v}_N \neq \vec{0}$. Since the columns of $\mathbf{A}$ are all linearly independent, the numerator is never zero unless $\vec{v}_N = \vec{0}$. Recall that even though the zero vector always satisfies the definition of an eigenvector, we never consider it to be an eigenvector because it is a trivial solution.

(b) Consider two eigenpairs $(\lambda_k, \vec{v}_k)$ and $(\lambda_\ell, \vec{v}_\ell)$ corresponding to distinct eigenvalues of $\mathbf{A}^T\mathbf{A}$—that is, $\lambda_k \neq \lambda_\ell$. Prove that the corresponding eigenvectors $\vec{v}_k$ and $\vec{v}_\ell$ are orthogonal: $\vec{v}_k \perp \vec{v}_\ell$.

To help you get started, consider the two equations

$$\mathbf{A}^T\mathbf{A}\,\vec{v}_k = \lambda_k\,\vec{v}_k \tag{2}$$

and

$$\vec{v}_\ell^T\mathbf{A}^T\mathbf{A} = \lambda_\ell\,\vec{v}_\ell^T. \tag{3}$$

The second equation can be derived by taking the transpose of both sides in the eigenvalue equation $\mathbf{A}^T\mathbf{A}\,\vec{v}_\ell = \lambda_\ell\,\vec{v}_\ell$. Premultiply Equation 2 with $\vec{v}_\ell^T$, postmultiply Equation 3 with $\vec{v}_k$, compare the two, and explain how one may then infer that $\vec{v}_k$ and $\vec{v}_\ell$ are orthogonal, i.e. $\langle\vec{v}_k, \vec{v}_\ell\rangle = 0$.

Premultiplication by a vector $\vec{x}$ means multiplying an expression by $\vec{x}$ on the left. For example, premultiplying the matrix $\mathbf{A}$ by $\vec{x}$ gives $\vec{x}\mathbf{A}$. Postmultiplication means multiplying on the right. Remember that in general matrix-vector multiplication is not commutative, so those operations are not identical.

Solution:

Following the hint:

$$\begin{aligned} \vec{v}_\ell^T\mathbf{A}^T\mathbf{A}\,\vec{v}_k &= \vec{v}_\ell^T\lambda_k\,\vec{v}_k \\ \vec{v}_\ell^T\mathbf{A}^T\mathbf{A}\vec{v}_k &= \lambda_\ell\,\vec{v}_\ell^T\,\vec{v}_k \end{aligned}$$

We see that the two expressions on the left are equal, so we set the two expressions on the right equal to each other:

$$\lambda_k\vec{v}_\ell^T\,\vec{v}_k = \lambda_\ell\,\vec{v}_\ell^T\,\vec{v}_k$$

If $\lambda_k \neq \lambda_l$, then the only possible solution is that $\vec{v}_\ell^T\,\vec{v}_k = 0$, which means $\vec{v}_\ell$ and $\vec{v}_k$ are orthogonal.

(c) The results of part (b) imply that the N eigenvectors of $\mathbf{A}^T\mathbf{A}$ are mutually orthogonal. A basis formed by vectors that are both (1) mutually orthogonal and (2) have unit length is called an orthonormal basis. Since the eigenvalues of $\mathbf{A}^T\mathbf{A}$ are distinct and have a norm of one, the eigenvectors form an orthonormal basis in $\mathbb{R}^N$. This means that we can express an arbitrary vector $\vec{x} \in \mathbb{R}^N$ as a linear combination of the eigenvectors $\vec{v}_1, \ldots, \vec{v}_N$, as follows:

$$\vec{x} = \sum_{n=1}^{N} \alpha_n \vec{v}_n.$$

i. Determine the nth coefficient α_n in terms of $\vec{x}$ and one or more of the eigenvectors $\vec{v}_1, \ldots, \vec{v}_N$.
Solution:
Since $\vec{v}_n$ are orthogonal, the coefficient α_n is the projection of $\vec{x}$ on to $\vec{v}_n$.
Since $\vec{v}_n$ are all unit vectors, the projection is simply the inner product.

$$\alpha_n = \langle \vec{x}, \vec{v}_n \rangle = \vec{x}^T \vec{v}_n$$

ii. Suppose $\vec{x}$ is a unit-length vector (i.e., a unit vector) in $\mathbb{R}^N$. Show that

$$\sum_{n=1}^{N} \alpha_n^2 = 1$$

where the α_n's are the coefficients of $\vec{x}$ in the basis defined earlier.
Solution:
Consider $||\vec{x}||^2 = 1$.

$$\begin{aligned} ||\vec{x}||^2 &= \vec{x}^T \vec{x} \\ &= \left(\sum_{i=1}^{N} \alpha_i \vec{v}_i\right)^T \left(\sum_{j=1}^{N} \alpha_j \vec{v}_j\right) = \left(\sum_{i=1}^{N} \alpha_i \vec{v}_i^T\right) \left(\sum_{j=1}^{N} \alpha_j \vec{v}_j\right) \\ &= \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j \vec{v}_i^T \vec{v}_j \end{aligned}$$

Now, since the $\vec{v}_i$ are orthogonal, we know: $\vec{v}_i^T \vec{v}_j = \begin{cases} 0, & i \neq j \\ 1, & i = j \end{cases}$

Therefore, $||\vec{x}||^2 = \sum_{n=1}^{N} \alpha_n^2 = 1$.

(d) Now express $||\mathbf{A}\vec{x}||^2$ in terms of $\{\alpha_1, \alpha_2 \ldots \alpha_N\}$, $\{\lambda_1, \lambda_2 \ldots \lambda_N\}$, and $\{\vec{v}_1, \vec{v}_2 \ldots \vec{v}_N\}$, and find an expression for $\vec{x}$ such that $||\mathbf{A}\vec{x}||^2$ is minimized. Do *not* use any tool from calculus to solve this problem, so avoid differentiation.

Hint: After expressing $||\mathbf{A}\vec{x}||^2$ in terms of $\{\alpha_1, \alpha_2 \ldots \alpha_N\}$, $\{\lambda_1, \lambda_2 \ldots \lambda_N\}$, and $\{\vec{v}_1, \vec{v}_2 \ldots \vec{v}_N\}$, which variable are you minimizing over?

Solution:
Note that $||\mathbf{A}\vec{x}||^2 = \vec{x}^T \mathbf{A}^T \mathbf{A} \vec{x}$. We express $\vec{x}$ in terms of $\vec{v}_i$ (the eigenvalues of $\mathbf{A}^T\mathbf{A}$) and expand.

$$\begin{aligned}\mathbf{A}^T\mathbf{A}\vec{x} &= \mathbf{A}^T\mathbf{A}\sum_{n=1}^{N}\alpha_n\vec{v}_n \\ &= \sum_{n=1}^{N}\alpha_n\mathbf{A}^T\mathbf{A}\vec{v}_n \\ &= \sum_{n=1}^{N}\alpha_n\lambda_n\vec{v}_n\end{aligned}$$

Now:

$$\begin{aligned}\vec{x}^T\mathbf{A}^T\mathbf{A}\vec{x} &= \vec{x}^T\sum_{n=1}^{N}\alpha_n\lambda_n\vec{v}_n \\ &= \left(\sum_{n=1}^{N}\alpha_n\vec{v}_n^T\right)\left(\sum_{n=1}^{N}\alpha_n\lambda_n\vec{v}_n\right) \\ &= \sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j\lambda_j\vec{v}_i^T\vec{v}_j \\ &= \sum_{n=1}^{N}\alpha_n^2\lambda_n \qquad (4)\end{aligned}$$

We know from part (c) that α_n are constrained by the following equation:

$$\sum_{n=1}^{N}\alpha_n^2 = 1 \qquad (5)$$

Equation 4 is a weighted sum of all the eigenvalues. All of the eigenvalues are larger than zero. To minimize that sum under the constraint in equation 5, we want to put as much weight on the smallest eigenvalue λ_N as possible. Therefore, we pick $\alpha_N = 1$ and $\alpha_k = 0$, where $k \in \{1,3,\ldots,N-1\}$. This is equivalent to picking $\vec{x}$ to be the eigenvector $\vec{v}_N$ with the smallest eigenvalue. The minimum value achieved is λ_N.