EECS 16A Designing Information Devices and Systems I
Fall 2018 Homework 13

This homework is due November 30, 2018, at 23:59.

This is a long homework so you should start early.

Self-grades are due December 4, 2018, at 23:59.

Submission Format

Your homework submission should consist of **two** files.

- `hw13.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.

- `hw13.ipynb`: A single IPython notebook with all of your code in it.

  In order to receive credit for your IPython notebook, you must submit both a "printout" and the code itself.

Submit each file to its respective assignment on Gradescope.

1. Recipe Reconnaissance

Engineering Edibles has been growing in size and popularity and has introduced two new cookies: Decadent Dwight and Heavenly Hearst. As a result of their popularity there is an increased interest in understanding their secret recipes.

The bakery produces 40 Decadent Dwight and 50 Heavenly Hearst cookies each day. Each cookie costs \$1. The bakery business is way overpriced, and everyone knows about 80 cents of the cost comes from profits (and labor); ingredients are only worth *about* 20 cents.

The team from Berkeley wants to figure out the recipes for the two new cookies, and they know the recipes are *different*. For the purpose of this problem, each cookie only contains three ingredients: eggs, sugar and butter (of course, you would use flour, water, etc. in real life!).

(a) How many unknowns are there in the problem? How many linearly independent equations would you need to be able to solve for these unknowns (assuming a consistent system of equations)?

Solution:

There are 6 unknowns: the amount of eggs, sugar, and butter used in Decadent Dwight cookies and the amount of eggs, sugar, and butter used in Heavenly Hearst cookies. We will denote these $D_e \frac{\text{eggs}}{\text{cookie}}$, $D_s \frac{\text{grams sugar}}{\text{cookie}}$, $D_b \frac{\text{grams butter}}{\text{cookie}}$, $H_e \frac{\text{eggs}}{\text{cookie}}$, $H_s \frac{\text{grams sugar}}{\text{cookie}}$, $H_b \frac{\text{grams butter}}{\text{cookie}}$. So you would need 6 linearly independent equations, if the system is consistent.

(b) Unfortunately, the team is not able to find precise information about the ingredients used in the cookie making process.

They stake-out Engineering Edibles, and they see Bob the Baker buy a dozen eggs for \$2 **daily** and a 5 kg bag of sugar **every week**. They hire a master-taster, who tells them there is **exactly** 10 grams of butter in each of the cookies (for both Decadent Dwight and Heavenly Hearst). The team precisely knows how much butter is in the cookies. They know from the supermarket that 1 kg of sugar costs 5 dollars, and 100 grams of butter costs 1 dollar. (*The prices of sugar and butter are precise and is precisely known, as is the amount of butter from the master taster.*) However, the amount of eggs and sugar are imprecisely known.
All this is not enough for the team to unlock the secret recipe! (Why?)

One day, however, the team gets lucky and hears through the grapevine that Heavenly Hearst contains **about** 10 grams of sugar. (*This value might not be precise, since it's just gossip!*)

Let's also remember that the ingredients for each cookie of either variety would cost \$0.2.
Using the information above, set up the problem as a least-squares problem and find best estimate of the secret recipe. Identify what variables you are estimating. You are welcome to use a computer to solve this.
Solution:
Rubric Note: Give yourself full credit as long as you set up the least squares problem correctly, you don't have to worry about scaling changing the numerical solutions.
We know from the master-taster that $D_b = 10\frac{\text{grams}}{\text{cookie}}$ and $H_b = 10\frac{\text{grams}}{\text{cookie}}$. This leaves us with only 4 unknowns.

We set up the following system of equations:

$$40 \times D_e + 50 \times H_e = 12 \tag{1}$$
$$7 \times (40 \times D_s + 50 \times H_s) = 5000 \tag{2}$$
$$\frac{2}{12} \times D_e + \frac{5}{1000} \times D_s = 0.10 \tag{3}$$
$$\frac{2}{12} \times H_e + \frac{5}{1000} \times H_s = 0.10 \tag{4}$$
$$H_s = 10 \tag{5}$$

i. **Eggs**: Bob buys a dozen eggs every day, and he makes 40 Decadent Dwight cookies and 50 Heavenly Hearst cookies daily. Therefore the 12 eggs must be somehow divided into the cookies! 40 cookies $\times D_e \frac{\text{eggs}}{\text{cookie}} + 50$ cookies $\times H_e \frac{\text{eggs}}{\text{cookie}} = 12$ eggs
ii. **Sugar**: Similar logic as Equation 1, except he buys 5000 grams of sugar each week. 7 days $\times$ $(40 \frac{\text{cookies}}{\text{day}} \times D_s \frac{\text{grams}}{\text{cookie}} + 50 \frac{\text{cookies}}{\text{day}} \times H_s \frac{\text{grams}}{\text{cookie}}) = 5000$ grams
iii. **Cost**: We're told that each cookie uses about 20 cents worth of ingredients. We first subtract the cost of butter, which is 10 grams $\times \frac{\$1}{100\text{ grams}} = \$0.1$. Now we know the eggs and sugar in the cookie cost 20 - 10 = 10 cents. The rest follows from dimensional analysis: $\frac{\$2}{12\text{ eggs}} \times D_e \frac{\text{eggs}}{\text{cookie}} + \frac{\$5}{1000\text{ grams}} \times D_s \frac{\text{grams}}{\text{cookie}} = 0.1 \frac{\$}{\text{cookie}}$
iv. **Cost**: Same as Equation 3, except for Heavenly Hearst. This symmetry results from the ingredients costing the same regardless of cookie, and the cookies being sold for the same price!
v. **Sugar Estimate, H_s**: We are given this information!

Now we can simplify and set this up in a matrix:

$$\begin{bmatrix} 40 & 50 & 0 & 0 \\ 0 & 0 & 40 & 50 \\ \frac{2}{12} & 0 & \frac{5}{1000} & 0 \\ 0 & \frac{2}{12} & 0 & \frac{5}{1000} \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} D_e \\ H_e \\ D_s \\ H_s \end{bmatrix} = \begin{bmatrix} 12 \\ \frac{5000}{7} \\ 0.1 \\ 0.1 \\ 10 \end{bmatrix}$$

Now we can use iPython to solve using least squares (see sol13.ipynb). This will return the following solution vector:

$$\begin{bmatrix} D_e \\ H_e \\ D_s \\ H_s \end{bmatrix} = \begin{bmatrix} 0.2386 \\ 0.0491 \\ 5.3571 \\ 10 \end{bmatrix}$$

(c) The Berkeley team decides to do scientific experiments to better their estimates of the recipe. They obtain a scale, and weigh the new cookies. Their (cheap and second-hand) scale is only accurate to the gram and easily affected by air currents, so they get noisy observations: Decadent Dwight is *about* 25 grams, and Heavenly Hearst is about 24 grams. Assume that 1 egg = 50 grams.

Update the least squares problem with this new information, and see how the values change. Are the new values more accurate? (*Hint: Find the sum of squared errors, $||e||^2$ then divide $||e||$ by the number of measurements to find the **average error for data points** for both (b) and (c).*)

Solution:

Rubric Note: Give yourself full credit as long as you set up the least squares problem correctly, you don't have to worry about scaling changing the numerical solutions.

To use the new information, we simply update the previous matrix:

$$\begin{bmatrix} 40 & 50 & 0 & 0 \\ 0 & 0 & 40 & 50 \\ \frac{2}{12} & 0 & \frac{5}{1000} & 0 \\ 0 & \frac{2}{12} & 0 & \frac{5}{1000} \\ 0 & 0 & 0 & 1 \\ 50 & 0 & 1 & 0 \\ 0 & 50 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} D_e \\ H_e \\ D_s \\ H_s \end{bmatrix} = \begin{bmatrix} 12 \\ \frac{5000}{7} \\ 0.1 \\ 0.1 \\ 10 \\ 15 \\ 14 \end{bmatrix}$$

Again, we first subtract the mass of butter (in this case, 10 grams) from the measurements. Since we know 1 egg = 50 grams, and since sugar is already given in units of grams, the equation follows:

$50\,\frac{\text{grams}}{\text{egg}} \times D_e\,\frac{\text{eggs}}{\text{cookie}} + D_s\,\frac{\text{grams}}{\text{cookie}} = 15\,\frac{\text{grams}}{\text{cookie}}$.

Similarly,

$50\,\frac{\text{grams}}{\text{egg}} \times H_e\,\frac{\text{eggs}}{\text{cookie}} + H_s\,\frac{\text{grams}}{\text{cookie}} = 14\,\frac{\text{grams}}{\text{cookie}}$.

We use the same iPython code as before, but with the updated information. This returns the following vector:

$$\begin{bmatrix} D_e \\ H_e \\ D_s \\ H_s \end{bmatrix} = \begin{bmatrix} 0.1946 \\ 0.0822 \\ 5.3571 \\ 10.000 \end{bmatrix}$$

With 5 equations the sum of squared errors is $||e||^2 = 0.0029$ and with 7 equations $||e||^2 = 0.0339$. We notice that with 7 equations the error increased. This is not surprising since we added more data points that our model is unable to predict perfectly. Thus we have more terms in the sum and more error. What is more interesting is that for this problem even when we calculate the average error for a given data point

$$\text{5 equations:} \qquad ||e||^2 = 0.0029 \to ||e|| = 0.0535 \to \frac{||e||}{5} = 0.0107 \tag{6}$$

$$\text{7 equations:} \qquad ||e||^2 = 0.0339 \to ||e|| = 0.1841 \to \frac{||e||}{N} = 0.0263. \tag{7}$$

it is larger for the 7 equation model. What this means is that our new observations are either noisier than the previous observations or they disagree with the model suggested by the first 5 data points. This is not necessarily a bad thing. If we know that the new data points are noisier than the old data points there are ways to adjust the least squares formula to compensate for this (outside the scope of this course). If we know that the new data points are actually quite accurate, this discrepancy could suggest that our linear model is naive and we have to consider more complicated models. Perhaps our previous model deceived us into thinking we had a good fit when we really were just not looking at the right data. Since we are trying to build a model that estimates the real world, we don't ever want to throw out good data just to make our model fit better.

Rubric Note: Give yourself full credit if you calculated the magnitude of the error vector for Parts B and C. Don't take away points from yourself for your explanation to "Are the new values more accurate?" That question was meant to get you thinking about why the Least Squares technique would output certain results.

2. Constrained Least Squares Optimization

In this problem, you'll go through a process of guided discovery to solve the following optimization problem: Consider a matrix $\mathbf{A} \in \mathbb{R}^{M \times N}$, of full column rank, where $M > N$. Determine a unit vector $\vec{x}$ that minimizes $||\mathbf{A}\vec{x}||$, where $||\cdot||$ denotes the 2-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:

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

This task may *seem* like solving a standard least squares problem $\mathbf{A}\vec{x} = \vec{b}$, where $\vec{b} = \vec{0}$, but it isn't. As an example, notice $\vec{x} = \vec{0}$ is *not* a valid solution to our prolem, because the zero vector does not have unit length. However, it could be the solution to a different least squares problem. Our optimization problem is a least squares problem with a constraint—hence the term *Constrained Least Squares Optimization*. The constraint is that the vector $\vec{x}$ must lie on the unit sphere in $\mathbb{R}^N$. You'll tackle this problem in a methodical, step-by-step fashion.

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 ascending 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 $0 < \lambda_1$, i.e. all the eigenvalues are strictly positive.
Hint: Consider $\|\mathbf{A}\vec{v}\|^2$
Solution:
Consider $\|\mathbf{A}\vec{v}\|^2$ for eigenvector $\vec{v}$, with eigenvalue λ.

$$\begin{aligned}\|\mathbf{A}\vec{v}\|^2 &= \vec{v}^T\mathbf{A}^T\mathbf{A}\vec{v}\\ &= \vec{v}^T\lambda\vec{v}\\ \lambda &= \frac{\|\mathbf{A}\vec{v}\|^2}{\|\vec{v}\|^2}\end{aligned}$$

Therefore, $\lambda > 0$ since norms are positive if $\vec{v} \neq \vec{0}$. Since $\mathbf{A}$ is full rank, the numerator is never 0 unless $\vec{v} = \vec{0}$.

(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

$$\mathbf{A}^T\mathbf{A}\vec{v}_k = \lambda_k\vec{v}_k \tag{8}$$

and

$$\vec{v}_\ell^T\mathbf{A}^T\mathbf{A} = \lambda_\ell\vec{v}_\ell^T. \tag{9}$$

Premultiply Equation 8 with $\vec{v}_\ell^T$, postmultiply Equation 9 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. $\langle\vec{v}_k, \vec{v}_\ell\rangle = 0$.
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 the two expressions on the left are equal, so 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) implies that the N eigenvectors of $\mathbf{A}^T\mathbf{A}$ are mutually orthogonal—and each has unit length. A basis formed by vectors that are both (1) mutually orthogonal and (2) has unit length is called an orthonormal basis. Since the eigenvalues of $\mathbf{A}^T\mathbf{A}$ are distinct, the eigenvectors form a basis, and the extra properties imply they 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 n^{th} coefficient α_n in terms of $\vec{x}$ and one or more of the eigenvectors $\vec{v}_1, \ldots, \vec{v}_N$.
Solution: Method 1: Let the eigenvectors of $\mathbf{A}^T\mathbf{A}$ be the columns of $\mathbf{V}$. We can formulate the finding the coordinates of an arbitrary vector in this basis as a linear system of equations,

$$\mathbf{V}\vec{\alpha} = \vec{x} \tag{10}$$

.
Because we know that the columns of $\mathbf{V}$ form as basis of $\mathbb{R}^{N\times N}$, the columns of $\mathbf{V}$ will be linearly independent. This means their will be a unique solution for $\vec{\alpha}$, where the n^{th} coefficient will be at the n^{th} index of the vector, $\vec{\alpha}$.
Method 2: Since $\vec{v}_i$ are orthogonal, the coefficient α_i is the projection of $\vec{x}$ on to $\vec{v}_i$.
Since $\vec{v}_i$ are all unit vectors, the projection is simply the inner product.

$$\alpha_i = \langle \vec{x}, \vec{v}_i \rangle = \vec{x}^T \vec{v}_i$$

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 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 you're well-positioned to tackle the grand challenge of this problem—determine the unit vector $\vec{x}$ that minimizes $\|\mathbf{A}\vec{x}\|$.
Note that the task is the same as finding a unit vector $\vec{x}$ that minimizes $\|\mathbf{A}\vec{x}\|^2$.
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 $\vec{\hat{x}}$ such that $\|\mathbf{A}\vec{x}\|^2$ is minimized. You may *not* use any tool from calculus to solve this problem—so avoid differentiation of any flavor.
For the optimal vector $\vec{\hat{x}}$ that you determine—that is, the vector

$$\vec{\hat{x}} = \text{argmin}_{\vec{x}} \|\mathbf{A}\vec{x}\|^2 \quad \text{subject to the constraint} \quad \|\vec{x}\|^2 = 1,$$

determine a simple, closed-form expression for the minimum value

$$\min_{\|\vec{x}\|=1} \|\mathbf{A}\vec{x}\| = \left\|\mathbf{A}\vec{\hat{x}}\right\|.$$

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.

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

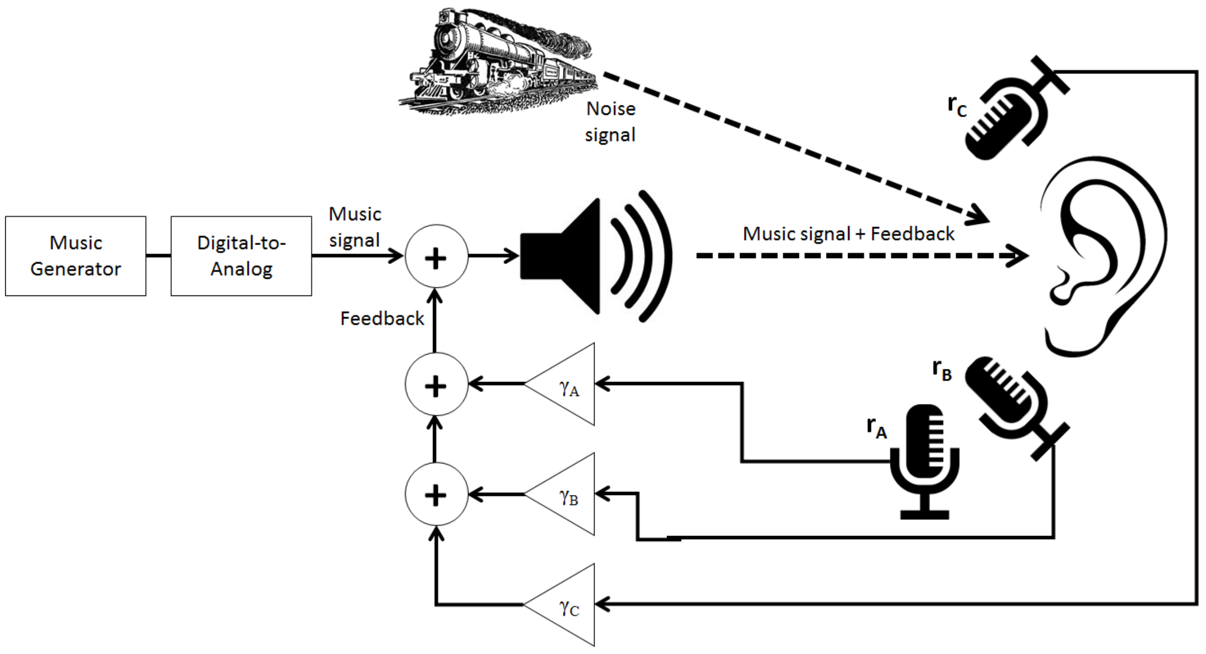
Now:

$$\begin{aligned}
\vec{x}^T\mathbf{A}^T\mathbf{A}\vec{x} &= \vec{x}^T\sum_{n=1}^{N}\alpha_n\lambda_n\vec{v}_n \\
&= \sum_{n=1}^{N}\alpha_n^2\lambda_n
\end{aligned}$$

To minimize, we pick the entire weight of α of the smallest λ, i.e. we pick $\vec{x}$ to be the eigenvector $\vec{v}_1$ with the smallest eigenvalue, and the minimum value is λ_1.

3. Noise Cancelling Headphones

In this problem, we will explore a common design for noise cancellation, using noise-cancelling 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 (such as other people's voices, a train going by, or just any kind of noise), 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 γ (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, so that they don't perfectly record the noise and that they are distinct recordings:

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

We can arrange the recordings into a matrix **R** and the microphone gains, γ, into a vector $\vec{\gamma}$

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

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

- the music signal $\vec{m} = \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ m_5 \end{bmatrix}$
- the noise signal $\vec{n}$
- the matrix of recorded noise signals **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:

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

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 represnted by the vector, $\vec{m}$, and the signal at the listener's ear is represented by the vector, $\vec{s}$ and can be written as the matrix sum:

$$\vec{s} = \vec{m} + \mathbf{R}\vec{\gamma} + \vec{n}$$

$$\begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \end{bmatrix} = \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ m_5 \end{bmatrix} + \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \\ a_4 & b_4 & c_4 \\ a_5 & b_5 & c_5 \end{bmatrix} \begin{bmatrix} \gamma_A \\ \gamma_B \\ \gamma_C \end{bmatrix} + \begin{bmatrix} n_1 \\ n_2 \\ n_3 \\ n_4 \\ n_5 \end{bmatrix}$$

(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 $\mathbf{R}\vec{\gamma} + \vec{n} = 0$. This equality cannot always be true, so we aim to minimize the norm of this quantity, so the minimization statement should be written as:

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

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

$$\vec{x} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\vec{b} \tag{11}$$

.

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 γ's that minimize the effect of noise.

$$\vec{n} = \begin{bmatrix} 0.10 \\ 0.37 \\ -0.45 \\ 0.068 \\ 0.036 \end{bmatrix}; \vec{r}_A = \begin{bmatrix} 0 \\ 0.11 \\ -0.31 \\ -0.012 \\ -0.018 \end{bmatrix}; \vec{r}_B = \begin{bmatrix} 0 \\ 0.22 \\ -0.20 \\ 0.080 \\ 0.056 \end{bmatrix}; \vec{r}_C = \begin{bmatrix} 0 \\ 0.37 \\ -0.44 \\ 0.065 \\ 0.038 \end{bmatrix}$$

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

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

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:

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

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 train our algorithm for a given number of microphones and a training signal. 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) **For Fun**:(ie. not graded) Use the IPython notebook to record the first noise signal using the `recordAmbientNoise` function and calculate a vector $\vec{\gamma}$. Create the noise cancellation signal by performing the multiplication $\mathbf{R}\vec{\gamma}$.

Solution: See the IPython notebook.

(g) **For Fun**:(ie. not graded) 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) **For Fun**:(ie. not graded) Try adding the other noise signal to the music signal without re-calculating new values for $\vec{\gamma}$ (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-cancelled signal seems to have the static noise removed. Some of the laughter and train whistle are removed, but there's still some distortion that seems to have the shape of the laughter. This will look like an envelope on the amplitude of the temporal signal.

(Any comment that notices distortion where the laughter or whistle are is a valid answer.)

4. Image Analysis

Applications in medical imaging often require an analysis of images based on the pixels of the image. For instance, we might want to count the number of cells in a given 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 is in the center of the image with standard axes and collect the following points: $(0.3,-0.69)$, $(0.5,0.87)$, $(0.9,-0.86)$, $(1,0.88)$, $(1.2,-0.82)$, $(1.5,0.64)$, $(1.8,0)$.

Recall that a quadratic equation of the form

$$ax^2+bxy+cy^2+dx+ey=1$$

can be used to represent an ellipse (if $b^2-4ac<0$), and a quadratic equation of the form

$$a(x^2+y^2)+dx+ey=1$$

is a circle if $d^2+e^2-4a>0$. The circle has fewer parameters.

(a) How can you find the equation of a circle that surrounds the cell? First, provide a setup and formulate a set of matrix equations to do this, i.e. an equation of the form $\mathbf{A}\vec{x}=\vec{b}+\vec{e}$ where you attempt to find the unknown coefficients a, d, and e from your points and $\vec{e}$ is the error vector. *Hint: x^2+y^2, x, and y can be thought of as variables calculated from your data points*

Solution:

The setup is:

$$\begin{bmatrix} x^2+y^2 & x & y \\ \vdots & \vdots & \vdots \end{bmatrix}\begin{bmatrix} a \\ d \\ e \end{bmatrix}=\begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$$

We plug in numbers to get:

$$\begin{bmatrix} 0.5661 & 0.3 & -0.69 \\ 1.0069 & 0.5 & 0.87 \\ 1.5496 & 0.9 & -0.86 \\ 1.7744 & 1 & 0.88 \\ 2.1124 & 1.2 & -0.82 \\ 2.6596 & 1.5 & 0.64 \\ 3.24 & 1.8 & 0 \end{bmatrix}\begin{bmatrix} a \\ d \\ e \end{bmatrix}=\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

(b) How can you find the equation of an ellipse that surrounds the cell? Provide a setup and formulate a set of matrix equations similar to in part a.

Solution:

The setup is:

$$\begin{bmatrix} x^2 & xy & y^2 & x & y \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}\begin{bmatrix} a \\ b \\ c \\ d \\ e \end{bmatrix}=\begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$$

We then plug in values to get:

$$\begin{bmatrix} 0.09 & -0.207 & 0.4761 & 0.3 & -0.69 \\ 0.25 & 0.435 & 0.7569 & 0.5 & 0.87 \\ 0.81 & -0.774 & 0.7396 & 0.9 & -0.86 \\ 1 & 0.88 & 0.7744 & 1 & 0.88 \\ 1.44 & -0.984 & 0.6724 & 1.2 & -0.82 \\ 2.25 & 0.96 & 0.4096 & 1.5 & 0.64 \\ 3.24 & 0 & 0 & 1.8 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

(c) In the IPython notebook, write a short program to fit a circle using these points. If you model your system of equations as $\mathbf{A}\vec{x} = \vec{b} + \vec{e}$, where $\vec{e}$ is the error vector and the number of data points is N, what is $\frac{\|\vec{e}\|}{N}$? Plot your points and the best fit circle in IPython.

Solution:

See `sol12.ipynb`.

The solution vector is:

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

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: $\frac{0.96}{7} = 0.137$.

(d) Write a short program in IPython to fit an ellipse using these points. If you model your system of equations as $\mathbf{A}\vec{x} = \vec{b} + \vec{e}$, where $\vec{e}$ is the error vector and the number of data points is N, what is $\frac{\|\vec{e}\|}{N}$? Plot your points and the best fit ellipse in IPython. How does this error compare to the one in the previous subpart? Which technique is better?

Solution:

See `sol12.ipynb`.

The solution vector is:

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

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 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. This is similar to how a higher degree polynomial has greater degrees of freedom.

5. Pollster: Regularized Least Squares

Least squares techniques are useful for many different kinds of prediction problems (also called *regression*). Here, we'll explore how least squares can be used for polling prediction. [1]

[1] The core ideas we learned in class have been extensively further developed—if you're interested you **should take** EE127A (convex optimization), CS189 (machine learning), EE221 (linear systems)! In these classes you learn ideas that build off of the basic least squares problem for applications in finance, healthcare, advertising, image processing, control, and many other fields.

Your job to predict how each county will vote, either for candidate A or candidate B. To do this you will build a linear model that predicts how each county will vote based on the (un)importance of several topics (the economy, healthcare, education, pineapple pizza, etc). Each topic is graded on an importance scale where a negative value means that the topic is not important and a positive value means that the topic is important. The magnitude of the number corresponds to how important/unimportant the topics are to members of the county.

Each county is represented by a "feature vector" of length 10. The value of each element of the vector captures the importance of that feature to the county. The dataset you are given contains the "feature vectors" for 100 counties as well as how those 100 counties will vote, a value of $+1$ if the county votes for candidate A and a value of -1 if the county votes for candidate B. We want you to find a "good" linear model using the first 90 counties (this is known as training data) and test the "good"ness of your model on the remaining 10 counties (testing data). Your job in this problem is to find the "good" linear model, with weights α_i, using the training data's "feature vectors", $\vec{f}$, and voting decision, b, to minimize the total prediction error of the testing data.

$$b = \alpha_1 f_1 + \alpha_2 f_2 + \ldots + \alpha_{10} f_{10} = \vec{\alpha}^T \vec{f} \tag{12}$$

(a) **Using only the training data**, set up the least squares problem ($\mathbf{A}\vec{x} = \vec{b}$). How are $\mathbf{A}$, $\vec{x}$, and $\vec{b}$ constructed? What are the dimensions of $\mathbf{A}$, $\vec{x}$ and $\vec{b}$?

Solution: In this problem the dimensions for training are $N_{examples} = 90$ and $N_{features} = 10$.

We will create $\mathbf{A}$ by using the equation above written for each training example. We combine them by placing $\vec{f}^T$ for each training example along the columns of $\mathbf{A}$. Because their are 90 training examples and each "feature vector" is length 10. The dimensions of $\mathbf{A}$ will be $N_{examples} \times N_{features} = 90 \times 10$.

We will generate $\vec{b}$ by placing the voting decision for each county at the index corresponding to its $\vec{f}$ in $\mathbf{A}$. Thus, it will have length $N_{examples} = 90$.

$\vec{x}$ will contain the linear model weights which we will solve for using linear least squares and is a column vector with length $N_{features} = 10$.

$$\underbrace{\begin{bmatrix} - & \vec{f}_1 & - \\ - & \vec{f}_2 & - \\ & \vdots & \\ - & \vec{f}_{90} & - \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_9 \\ \alpha_{10} \end{bmatrix}}_{\vec{x}} = \underbrace{\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{90} \end{bmatrix}}_{\vec{b}} \tag{13}$$

(b) Using IPython, build these matrices and solve for $\hat{\vec{x}}$ using linear least squares with the provided function `doLeastSquares(A,b)`. Evaluate what the total prediction error is on the **training** data using your linear model (i.e. the length of the error vector, $\|\vec{e}\| = \|\vec{b} - \mathbf{A}\vec{x}\|$). Also, evaluate what the total prediction error is on the **testing** data using your linear model.

Solution:

See IPython notebook for exact implementation, but at the high level: Setting up the matrices according to the solution from the previous part and using `doLeastSquares(A,b)` to solve for the feature weights returns roughly

Total prediction error: 0.8633164435610858
Total prediction error: 0.32034153218315026

(c) A real life problem when building models can be the data itself. In a country with two very polarizing candidates, knowing how a county feels about one topic allows us to predict how important they consider *every* topic. This issue makes the columns of $\mathbf{A}$ almost linearly dependent—something we observed in the image stitching homework problem and in the imaging lab! Let us analyze it further by looking at the eigenvalues of $\mathbf{A}^T\mathbf{A}$ denoted λ_i. Show that the total prediction error

$$\|\vec{e}\| = \|\vec{b} - \mathbf{A}\vec{\hat{x}}\| = \left\|\vec{b} - \mathbf{A}\left(\frac{\beta_1}{\lambda_1}\vec{v}_1 + \frac{\beta_2}{\lambda_2}\vec{v}_2 + \ldots + \frac{\beta_N}{\lambda_N}\vec{v}_N\right)\right\|. \tag{14}$$

The β_i are the coordinates of the vector $\mathbf{A}^T\vec{b}$ in the eigenbasis of $(\mathbf{A}^T\mathbf{A})^{-1}$,

$$\mathbf{A}^T\vec{b} = \beta_1\vec{v}_1 + \beta_2\vec{v}_2 + \ldots + \beta_N\vec{v}_N \tag{15}$$

(Hint: Consider how the eigenbasis for $(\mathbf{A}^T\mathbf{A})^{-1}$ is related to the eigenbasis of $\mathbf{A}^T\mathbf{A}$). The issue described above will make some of the $\lambda_i \approx 0$. What happens to the total prediction error when there exist eigenvalues $\lambda_i \approx 0$? For this problem assume all eigenvalues are distinct and unique.

Solution:

$$\vec{e} = \|\vec{b} - \mathbf{A}\vec{\hat{x}}\| \tag{16}$$
$$= \|\vec{b} - \mathbf{A}(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\vec{b}\| \tag{17}$$

Now consider expressing $\vec{z} = \mathbf{A}^T\vec{b}$ in the eigenbasis of $(\mathbf{A}^T\mathbf{A})^{-1}$. This will allow us to understand how different components in $\vec{z}$ are affected by applying the matrix $(\mathbf{A}^T\mathbf{A})^{-1}$.

$$(\mathbf{A}^T\mathbf{A})^{-1}\vec{z} = (\mathbf{A}^T\mathbf{A})^{-1}(\beta_1\vec{v}_1 + \beta_2\vec{v}_2 + \ldots) \tag{18}$$
$$= \beta_1(\mathbf{A}^T\mathbf{A})^{-1}\vec{v}_1 + \beta_2(\mathbf{A}^T\mathbf{A})^{-1}\vec{v}_2 + \ldots \tag{19}$$
$$= \frac{\beta_1}{\lambda_1}\vec{v}_1 + \frac{\beta_2}{\lambda_2}\vec{v}_2 + \ldots + \frac{\beta_N}{\lambda_N}\vec{v}_N \tag{20}$$

Plugging back in,

$$\|\vec{e}\| = \|\vec{b} - \mathbf{A}\vec{\hat{x}}\| \tag{21}$$
$$= \left\|\vec{b} - \mathbf{A}\left(\frac{\beta_1}{\lambda_1}\vec{v}_1 + \frac{\beta_2}{\lambda_2}\vec{v}_2 + \ldots + \frac{\beta_N}{\lambda_N}\vec{v}_N\right)\right\| \tag{22}$$

What this implies is that our total prediction error will be large when a $\lambda_i \approx 0$. This is bad and will cause poor prediction performance for individual predictions.

(d) In IPython, plot the eigenvalues of $\mathbf{A}^T\mathbf{A}$. Do we encounter the problem described in part (c)?

Solution: See the IPython notebook. After plotting $\mathbf{A}^T\mathbf{A}$'s 10 eigenvalues, the last 6 should be ≈ 0. This will cause the issue described in part (c), i.e. extremely large prediction error.

(e) There are many solutions to this issue, but a common one involves including prior knowledge. We introduce now a value γ which we will use to try and rectify the error from part (c). We are given that the weights of our linear model tend to be small (close to zero). Written in equation form,

$$\sqrt{\gamma}\alpha_1 = 0 \ \ldots \ \sqrt{\gamma}\alpha_{10} = 0 \tag{23}$$

and as a matrix,

$$\sqrt{\gamma}\,\mathbf{I}\vec{\alpha} = \vec{0}. \tag{24}$$

Let us concatentate the new equations to the bottom of our matrix equation, $\mathbf{A}\vec{x} = \vec{b}$, from the previous parts to create the augmented matrix, $\tilde{\mathbf{A}}$, and the augmented vector, $\tilde{\vec{b}}$. Our modified matrix equations will be,

$$\tilde{\mathbf{A}}\vec{x} = \tilde{\vec{b}} \rightarrow \left[\frac{\mathbf{A}}{\sqrt{\gamma}\,\mathbf{I}}\right]\vec{x} = \left[\frac{\vec{b}}{\vec{0}}\right] \tag{25}$$

What are the dimensions of $\tilde{\mathbf{A}}$, $\tilde{\vec{b}}$? Using IPython, create these matrices using `np.concatenate`.
Solution: We know from part (a) that the dimensions of the original matrix $\mathbf{A}$ was $N_{examples} \times N_{features}$, or 90×10. The size of $\vec{x}$ will not change (because the size of our model, i.e. the amount of α_i values, will not change), so we know that $\sqrt{\gamma}\mathbf{I}$ will be $N_{features} \times N_{features}$. Thus, the dimensions of $\tilde{\mathbf{A}}$ are $(N_{examples} + N_{features}) \times N_{features}$ and the length of $\tilde{\vec{b}}$ is $N_{examples} + N_{features}$.

See IPython notebook for details on implementation with NumPy..

(f) Using the solution to the linear least squares problem derived in class, show that solution to the modified linear least squares problem is

$$\tilde{\vec{x}} = (\tilde{\mathbf{A}}^T\tilde{\mathbf{A}})^{-1}\tilde{\mathbf{A}}^T\tilde{\vec{b}} = (\mathbf{A}^T\mathbf{A} + \gamma\mathbf{I})^{-1}\mathbf{A}^T\vec{b}. \tag{26}$$

Hint: Matrix-matrix multiplication can be handled in blocks! However, there are restrictions on the dimensions of the matrices when written in this block format, certain dimensions must agree. In this setting, $\mathbf{W} \in \mathbb{R}^{n\times m}$, $\mathbf{Y} \in \mathbb{R}^{m\times l}$, $\mathbf{X} \in \mathbb{R}^{n\times p}$, and $\mathbf{Z} \in \mathbb{R}^{p\times l}$. User beware: Remember that matrix-matrix multiplication does not, in general, commute.

$$\left[\begin{array}{c|c} \mathbf{W} & \mathbf{X} \end{array}\right]\left[\frac{\mathbf{Y}}{\mathbf{Z}}\right] = \mathbf{W}\mathbf{Y} + \mathbf{X}\mathbf{Z}$$

Solution:

$$\tilde{\vec{x}} = (\tilde{\mathbf{A}}^T\tilde{\mathbf{A}})^{-1}\tilde{\mathbf{A}}^T\tilde{\vec{b}} \tag{27}$$

$$= \left(\left[\frac{\mathbf{A}}{\sqrt{\gamma}\,\mathbf{I}}\right]^T\left[\frac{\mathbf{A}}{\sqrt{\gamma}\,\mathbf{I}}\right]\right)^{-1}\left[\frac{\mathbf{A}}{\sqrt{\gamma}\,\mathbf{I}}\right]^T\left[\frac{\vec{b}}{\vec{0}}\right] \tag{28}$$

$$= \left(\left[\begin{array}{c|c} \mathbf{A}^T & \sqrt{\gamma}\,\mathbf{I} \end{array}\right]\left[\frac{\mathbf{A}}{\sqrt{\gamma}\,\mathbf{I}}\right]\right)^{-1}\left[\begin{array}{c|c} \mathbf{A}^T & \sqrt{\gamma}\,\mathbf{I} \end{array}\right]\left[\frac{\vec{b}}{\vec{0}}\right] \tag{29}$$

$$= (\mathbf{A}^T\mathbf{A} + \gamma\mathbf{I})^{-1}(\mathbf{A}^T\vec{b} + \sqrt{\gamma}\,\mathbf{I}\vec{0}) \tag{30}$$

$$= (\mathbf{A}^T\mathbf{A} + \gamma\mathbf{I})^{-1}\mathbf{A}^T\vec{b} \tag{31}$$

(g) Show that the new total prediction error

$$\|\vec{e}\| = \|\vec{b} - \mathbf{A}\vec{x}\| = \left\|\vec{b} - \mathbf{A}\left(\frac{\beta_1}{\lambda_1+\gamma}\vec{v}_1 + \frac{\beta_2}{\lambda_2+\gamma}\vec{v}_2 + \ldots + \frac{\beta_N}{\lambda_N+\gamma}\vec{v}_N\right)\right\|, \tag{32}$$

using our modified least squares solution, $\vec{x} = (\mathbf{A}^T\mathbf{A}+\gamma\mathbf{I})^{-1}\mathbf{A}^T\vec{b}$, the eigenvalues λ_i of $\mathbf{A}^T\mathbf{A}$, and the eigenvectors $\vec{v}_i$ of $(\mathbf{A}^T\mathbf{A}+\gamma\mathbf{I})^{-1}$. The β_i are the coordinates of the vector $\mathbf{A}^T\vec{b}$ in the eigenbasis of $(\mathbf{A}^T\mathbf{A}+\gamma\mathbf{I})^{-1}$,

$$\mathbf{A}^T\vec{b} = \beta_1\vec{v}_1 + \beta_2\vec{v}_2 + \ldots + \beta_N\vec{v}_N \tag{33}$$

(Hint: Consider how the eigenbasis for $\mathbf{A}^T\mathbf{A}+\gamma I$ is related to the eigenbasis of $\mathbf{A}^T\mathbf{A}$. Think about diagonalization of $\mathbf{A}^T\mathbf{A}$.) For this problem assume all eigenvalues are distinct and unique.

Solution:

$$\begin{aligned}\|\vec{e}\| &= \|\vec{b} - \mathbf{A}\vec{x}\| &(34)\\ &= \|\vec{b} - \mathbf{A}(\mathbf{A}^T\mathbf{A}+\gamma\mathbf{I})^{-1}\mathbf{A}^T\vec{b}\| &(35)\end{aligned}$$

Let us calculate the eigenvalues of $(\mathbf{A}^T\mathbf{A}+\gamma\mathbf{I})^{-1}$ in terms of the eigenvalues of $\mathbf{A}^T\mathbf{A}$. First, we will calculate the eigenvalues of $\mathbf{A}^T\mathbf{A}+\gamma\mathbf{I}$ in terms of the eigenvalues of $\mathbf{A}^T\mathbf{A}$ by considering its diagonalization, $\mathbf{U}\Lambda\mathbf{U}^T$,

$$\mathbf{A}^T\mathbf{A}+\gamma\mathbf{I} = \mathbf{U}\Lambda\mathbf{U}^T + \gamma\mathbf{U}\mathbf{U}^T = \mathbf{U}(\Lambda+\gamma\mathbf{I})\mathbf{U}^T \tag{36}$$

.

The eigenvalues of $\mathbf{A}^T\mathbf{A}+\gamma\mathbf{I}$ are $\lambda_i+\gamma$ and the eigenvalues of $(\mathbf{A}^T\mathbf{A}+\gamma\mathbf{I})^{-1}$ are simply $\frac{1}{\lambda_i+\gamma}$.

Going through a similar process as the previous part, we express $\vec{z} = \mathbf{A}^T\vec{b}$ in the eigenbasis of $(\mathbf{A}^T\mathbf{A}+\gamma\mathbf{I})^{-1}$. This will allow us to understand how different components in $\vec{z}$ are affected by applying the matrix $(\mathbf{A}^T\mathbf{A}+\gamma\mathbf{I})^{-1}$.

$$\begin{aligned}(\mathbf{A}^T\mathbf{A}+\gamma\mathbf{I})^{-1}\vec{z} &= (\mathbf{A}^T\mathbf{A}+\gamma\mathbf{I})^{-1}(\beta_1\vec{v}_1+\beta_2\vec{v}_2+\ldots) &(37)\\ &= \beta_1(\mathbf{A}^T\mathbf{A}+\gamma\mathbf{I})^{-1}\vec{v}_1 + \beta_2(\mathbf{A}^T\mathbf{A}+\gamma\mathbf{I})^{-1}\vec{v}_2+\ldots &(38)\\ &= \frac{\beta_1}{\lambda_1+\gamma}\vec{v}_1 + \frac{\beta_2}{\lambda_2+\gamma}\vec{v}_2+\ldots &(39)\end{aligned}$$

Plugging back in,

$$\begin{aligned}\|\vec{e}\| &= \|\vec{b} - \mathbf{A}\vec{x}\| &(40)\\ &= \left\|\vec{b} - \mathbf{A}\left(\frac{\beta_1}{\lambda_1+\gamma}\vec{v}_1 + \frac{\beta_2}{\lambda_2+\gamma}\vec{v}_2 + \ldots + \frac{\beta_N}{\lambda_N+\gamma}\vec{v}_N\right)\right\| &(41)\end{aligned}$$

Wow that is incredible!!! By adding a small modification to the least squares problem, the prediction error now depends on the $\lambda_i+\gamma$ rather than λ_i on its own. We don't have control over λ_i, but we *do* have control over γ, meaning when we have $\lambda_i \approx 0$, we can control the error by setting $\gamma > 0$!

As one might imagine, there is no free lunch. We cannot set γ to any value, and the next parts of the problem explores how to find the best γ.

(h) In IPython, in **a single plot** display the eigenvalues of $\mathbf{A}^T\mathbf{A}+\gamma\mathbf{I}$ for several values of γ (e.g. 0, 10, 100). What does this do to the eigenvalues of $\mathbf{A}^T\mathbf{A}$?

Solution: The eigenvalues of $\mathbf{A}^T\mathbf{A}+\gamma\mathbf{I}$ are shifted up by γ, i.e. the new eigenvalues $\mu_i = \lambda_i + \gamma$

(i) In IPython, let us now find the "best" γ to improve our testing total prediction error. Evalute the total testing prediction error using different values of γ (use the list in the IPython Notebook). What is the best choice of γ (ie. which gamma minimizes the total testing prediction error)? How does the total testing prediction error using modified least squares with the best choice of γ compare with the total testing prediction error from part (b)?

Solution: See IPython notebook.

Optimal gamma: 9.326033468832199
Achieved Mean Square Prediction Error: 0.3065022811871663

The curve should dip down and then back up. There is a γ that minimizes the total testing prediction error. The total testing prediction error in the part will be less than the total testing prediction error from part (b).

If this problem whets your appetite, awesome! You should take classes in the areas of signal processing, optimization, and machine learning: EE127A, EE120, EE123, CS189, EE221, and EE225A.

6. OMP Exercise

(a) Suppose we have a vector $\vec{x} \in \mathbb{R}^4$. We take 3 measurements of it, $b_1 = \vec{m}_1^T\vec{x} = 4$, $b_2 = \vec{m}_2^T\vec{x} = 6$, and $b_3 = \vec{m}_3^T\vec{x} = 3$, where $\vec{m}_1$, $\vec{m}_2$ and $\vec{m}_3$ are some measurement vectors. In the general case when there are 4 unknowns in $\vec{x}$ and we only have 3 measurements, i.e. system is underdetermined, it is not possible to solve for $\vec{x}$ using Gaussian Elimination. Furthermore, there could be noise in the measurements. However, in this case, we are told that $\vec{x}$ is sparse and has only 2 non-zero entries. In particular,

$$\mathbf{M}\vec{x} \approx \vec{b}$$

$$\begin{bmatrix} - & \vec{m}_1^T & - \\ - & \vec{m}_2^T & - \\ - & \vec{m}_3^T & - \end{bmatrix} \vec{x} \approx \vec{b}$$

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

where exactly 2 of x_1 to x_4 are non-zero. Use Orthogonal Matching Pursuit to estimate x_1 to x_4. In this problem, since we are looking for a solution to the equation $\mathbf{M}\vec{x} = \vec{b}$, we will not cross correlate $\vec{b}$ with the columns of $\mathbf{M}$, instead we will just compute the inner product of $\vec{b}$ with every column of $\mathbf{M}$.

Solution:

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

$\vec{b}$:

$$\left\langle \vec{c}_1, \vec{b} \right\rangle = 10$$
$$\left\langle \vec{c}_2, \vec{b} \right\rangle = 7$$
$$\left\langle \vec{c}_3, \vec{b} \right\rangle = -3$$
$$\left\langle \vec{c}_4, \vec{b} \right\rangle = -1$$

Thus, $\vec{c}_1$ is the best matching vector. Now we compute projection of $\vec{b}$ onto $\vec{c}_1$, i.e. $\vec{c}_1 \frac{\langle \vec{b}, \vec{c}_1 \rangle}{\langle \vec{c}_1, \vec{c}_1 \rangle}$ and subtract it from $\vec{b}$:

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

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

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

Thus, we know that $\vec{c}_1$ and $\vec{c}_4$ contribute most to $\vec{b}$. However, 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

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

and the least squares formula gives:

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

Thus, $\vec{x} \approx \begin{bmatrix} 6\frac{1}{3} \\ 0 \\ 0 \\ 2\frac{2}{3} \end{bmatrix}$.

(b) We know that OMP works only when the vector $\vec{x}$ is sparse, which means that it has very few non-zero entries. What if $\vec{x}$ is not sparse in the standard basis but is only sparse in a different basis? What we can do is to change to the basis where $\vec{x}$ is sparse, run OMP in that basis, and transform the result back into the standard basis.

- Suppose we have a $m \times n$ measurement matrix $\mathbf{M}$ and a vector of measurements $\vec{b} \in \mathbb{R}^m$ where $\mathbf{M}\vec{x} = \vec{b}$ and we want to find $\vec{x} \in \mathbb{R}^n$. The basis that $\vec{x}$ is sparse in is defined by basis vectors $\vec{a}_1 \cdots \vec{a}_n$, and we define:

$$\mathbf{A} = \begin{bmatrix} | & & | \\ \vec{a}_1 & \cdots & \vec{a}_n \\ | & & | \end{bmatrix}$$

such that $\vec{x} = \mathbf{A}\vec{x}_a$ and that $\vec{x}_a$ **is sparse**.

- Suppose that we have an OMP function that can compute $\vec{x}'$ for $\mathbf{M}'\vec{x}' = \vec{b}'$ **only when $\vec{x}'$ is sparse**: $\vec{x}' = \text{OMP}(\mathbf{M}', \vec{b}')$.

Assuming that the **change of basis does not significantly affect the orthogonality of vectors**, describe how you would compute $\vec{x}$ using the function OMP using necessary equations.

Solution:

Since

$$\mathbf{M}\vec{x} = \vec{b},$$
$$\mathbf{MA}\vec{x}_a = \vec{b}$$

Using the OMP function we get:

$$\vec{x}' = \text{OMP}(\mathbf{MA}, \vec{b})$$

And the original $\vec{x}$ is thus:

$$\vec{x} = \mathbf{A}\vec{x}_a = \mathbf{A}\left(\text{OMP}(\mathbf{MA}, \vec{b})\right)$$

Out-of-scope remark: In certain cases (choice of $\mathbf{M}$ and $\mathbf{A}$), the assumption that "the change of basis does not significantly affect the orthogonality of vectors" can be formalized. We want it to be the case that if columns of $\mathbf{M}$ are roughly orthogonal, then so are columns of $\mathbf{MA}$. It turns out that if the measurement matrix $\mathbf{M}$ is chosen randomly (similarly to what you have seen in previous HWs) and the basis $\mathbf{A}$ is orthogonal, then not only will the columns of $\mathbf{M}$ be roughly orthogonal but also the columns of $\mathbf{MA}$ will be roughly orthogonal. Therefore, our assumption is justified in some cases. (As an exercise, it is possible to come up with "bad" choices of $\mathbf{M}$ and $\mathbf{A}$ for which the assumption strongly fails).

7. Sparse Imaging

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

However, we saw that an iterative algorithm that does "matching and peeling" can enable reconstruction of a sparse vector 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.)

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

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

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

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

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

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

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

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

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

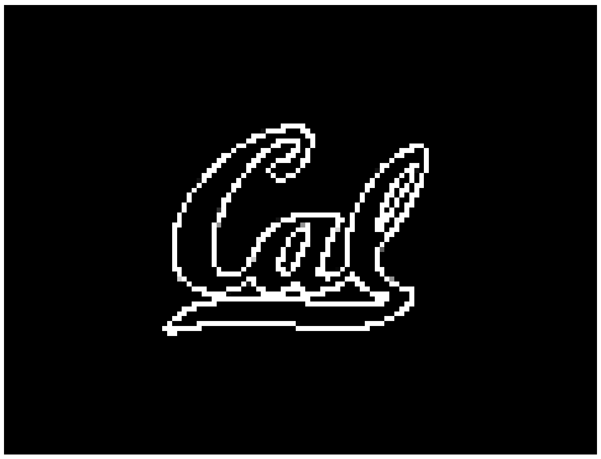
See `sol13.ipynb`.

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

(b) Run the code `rec = OMP((height, width), sparsity, measurements, A)` and see the image being correctly reconstructed from a number of samples smaller than the number of pixels of your figure. What is the image?

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 recover it. An example input image file is also supplied together with the code. Using `smiley.png`, generate an image of size 91×120 pixels of sparsity less than 400 and recover it using OMP with 6500 measurements.

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, as we have more nonzero pixels, they start to contribute in the measurements (in terms of the example given in the lecture, it is like having more users try to transmit their messages at the same time). In order to not be affected by these contributions, we need the signature of each pixel to be 'more orthogonal' to others. In our setting, this is achieved by taking more measurements with different masks.

8. 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.