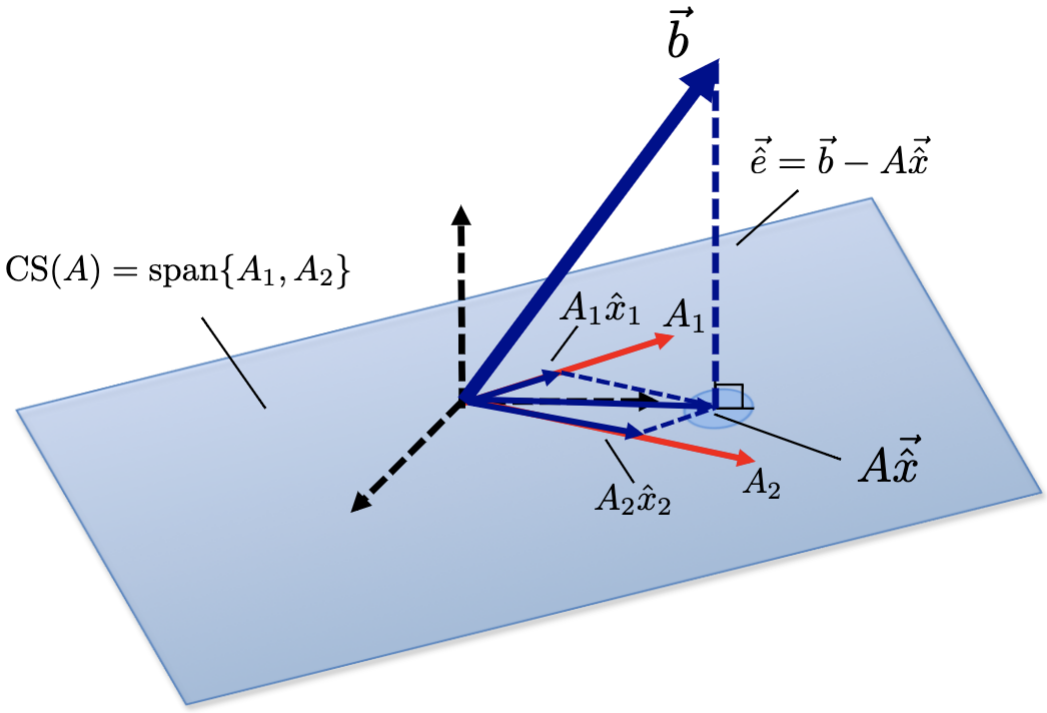
CSM 16A **Designing Information Systems and Devices**

Week 13 Worksheet Solutions

Term: **Spring 2020** *Name:*

Problem 1: Least Squares and Orthogonal Projection



1. Consider that you have some equations of the form $\mathbf{A}\vec{x} = \vec{b}$, however, that there is no solution $\vec{x}$ that solves the equations. What does this tell us about $\vec{b}$ with respect to $\mathbf{A}_1, \mathbf{A}_2$ (the columns of $\mathbf{A}$)?

   Solution: $\vec{b}$ is not in the column space of $\mathbf{A}$. If there was some $\vec{x}$ which solved the equation $\mathbf{A}\vec{x} = \vec{b}$, then we could write $\vec{b} = \mathbf{A}_1 x_1 + \mathbf{A}_2 x_2$, and $\vec{b}$ would have been in the column space of $\mathbf{A}$. However, since there is no such $\vec{x}$, $\vec{b}$ is not in the column space of $\mathbf{A}$.

2. We know that there is no $\vec{x}$ that satisfies the equations exactly, but we still want to solve the equations to get a solution as close as possible.

   Let's say you had 3 choices, $\vec{x}_i, \vec{x}_j, \vec{x}_k$ (these are not drawn on the image). What could you compute in order to determine which of these would be the best choice instead of $\vec{x}$

   Solution: A good idea would be to multiply out $\mathbf{A}\vec{x}_i, \mathbf{A}\vec{x}_j, \mathbf{A}\vec{x}_k$, and see which one results in something closest to $\vec{b}$, and that's the one that we would pick.

3. Suppose that the real vector $\vec{b} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$, and you have two vectors $\vec{x}_1, \vec{x}_2$ that are close to $\vec{b}$, which result in possible $\vec{b}_1 = \begin{bmatrix} 1.5 \\ 1.5 \end{bmatrix}$ and another $\vec{b}_2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$. How do we know which one is *closer* to $\vec{b}$? What if we define *the*

distance between two vectors as the sum of the components of the difference of the vectors.

Solution: There are many ways to define distance between 2 vectors. One way is to simply sum up the differences in individual entries. For instance, the sum of the entries in $\vec{b} - \vec{b}_1$ vs. the sum of the entries in $\vec{b} - \vec{b}_2$. The first sum equals 0, while the second one equals 1. Huh? Both of these vectors seemed like they were '1' unit away, but the first one results in a sum of 0. This is because of negative numbers.

$\vec{b}_1$ is -0.5 away in the first element, and 0.5 away in the second. We should interpret this as being 1 away. In other words, we should take the **absolute value of the differences** of the individual elements. It turns out that this is, mathematically, annoying.

4. What is a different approach to solve the issue discussed above?
Hint: Consider the euclidean distance between 2 points p_1 and p_2.

Solution: Another way to get rid of negative numbers is by squaring the element-wise difference between the 2 vectors. Define the 'distance' by the 'sum of the squares of individual elements in the vectors'.

5. Using this definition, how close is $\vec{b}_1$ to $\vec{b}$? How close is $\vec{b}_2$ to $\vec{b}$?

Solution: $\vec{b}_1$: $-0.5^2 + 0.5^2 = 0.5$ and $\vec{b}_2$: $1^2 + 0^2 = 1$. Using this definition, $\vec{b}_1$ is closer, and that's the one we should pick!

6. More generally, we actually don't have a choice of just two or three $\vec{x}$s to pick to get as close to $\vec{b}$ as possible. We have an infinite number of choices. How can we tell which one is the best? Look at the image given at the top of the question, and decide something about $\hat{\vec{x}}$ and $\hat{\vec{e}}$.

Solution: Intuitively, it makes sense that we want to pick an $\hat{\vec{x}}$ which results in the smallest euclidean distance between the 2 vectors; this can also be represented as the length of the vector $\hat{\vec{e}}$, which is the difference between $\vec{x}$ and $\vec{b}$. In general, we have $\hat{\vec{e}} = \vec{b} - \hat{\vec{x}}$, where we consider $\hat{\vec{x}}$ as our choice of "best" vector that creates the smallest possible distance.

$\hat{\vec{e}}$ is exactly the 'difference' that we discussed earlier of the correct $\vec{b}$ and the closest vector that we can get to $\vec{b}$. The norm-squared of $\hat{\vec{e}}$ is precisely the sum of squares of the elements in the difference. We want to minimize $\hat{\vec{e}}$.

7. Let's begin by recalling three facts. Recall that if we want to minimize some quantity squared, it is enough to minimize the quantity itself. Also, recall that given a point and a plane, the shortest line that one can possibly get starting at the point and ending at the plane, is a perpendicular line from the point to the plane. Finally, recall that if a vector $\vec{v}$ is orthogonal to another vector u, their inner product $\langle \vec{v}, \vec{u} \rangle = \vec{v}^T \vec{u} = 0$. Using this information, come up with a method to minimize the norm-squared of $\hat{\vec{e}}$.

Solution: To minimize the norm-squared of $\hat{\vec{e}}$, we just want $\hat{\vec{e}}$ to be perpendicular to the column space of $\mathbf{A}$. This is the shortest-length (or smallest-norm) $\hat{\vec{e}}$ possible. Since finding the shortest-length vector is the same thing as minimizing the norm, which is equivalent to minimizing the squared norm.

To make this vector orthogonal to the column space of $\mathbf{A}$, **it should be orthogonal to every column vector in $\mathbf{A}$**. In other words, if

$$\mathbf{A} = \begin{bmatrix} \uparrow & \uparrow & \dots & \uparrow \\ \vec{a_1} & \vec{a_2} & \dots & \vec{a_n} \\ \downarrow & \downarrow & \downarrow & \downarrow \end{bmatrix},$$

using the definition of orthogonality between two vectors, we know that:

$$\begin{aligned} \left\langle \vec{a_1}, \mathbf{A}\hat{\vec{x}} - \vec{b} \right\rangle = \vec{a_1}^T (\mathbf{A}\hat{\vec{x}} - \vec{b}) &= 0 \\ \left\langle \vec{a_2}, \mathbf{A}\hat{\vec{x}} - \vec{b} \right\rangle = \vec{a_2}^T (\mathbf{A}\hat{\vec{x}} - \vec{b}) &= 0 \\ \dots & \\ \left\langle \vec{a_n}, \mathbf{A}\hat{\vec{x}} - \vec{b} \right\rangle = \vec{a_n}^T (\mathbf{A}\hat{\vec{x}} - \vec{b}) &= 0 \end{aligned}$$

Now, note that:

$$\begin{bmatrix} \leftarrow & \vec{a_1}^T & \rightarrow \\ \leftarrow & \vec{a_2}^T & \rightarrow \\ & \vdots & \\ \leftarrow & \vec{a_n}^T & \rightarrow \end{bmatrix} = \mathbf{A}^T,$$

more compactly, this allows us to write all the n equations above as: $\mathbf{A}^T(\mathbf{A}\vec{\hat{x}} - \vec{\hat{b}}) = \vec{0}$. This says $\mathbf{A}^T\mathbf{A}\vec{\hat{x}} = \mathbf{A}^T\vec{\hat{b}}$. The best $\vec{\hat{x}}$ we can find is $(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\vec{b}$.

Problem 2: Least Squares

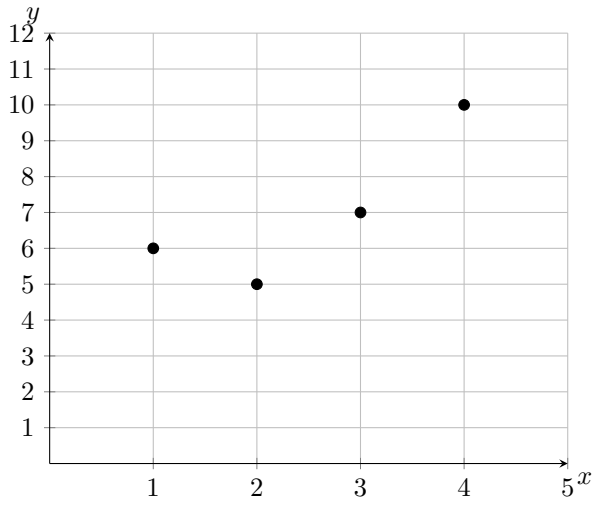
Consider the basic problem of finding some $\vec{x}$ such that $\mathbf{A}\vec{x} = \vec{b}$. If we have an equal number of equations as unknowns, then we are pretty happy.

In many cases, however, we cannot find a solution, as the set of equations that are described by $\mathbf{A}$ and $\vec{b}$ are overdetermined (i.e. there are more equations than there are unknowns).

In general, least squares involves finding the best approximate solution to these overdetermined systems.

Looking at this graphically, we can think of the problem of data-fitting. That is to say that we have many samples, and we want to draw a straight line that goes as close to each point as possible.

1. Looking at the plot below, setup a system of linear equations describing a line going through each point.



Solution: Recall that the generalized equation for a line is $y = mx + c$, where m is the slope of the line and c is a constant shift. Since we have 4 points, we have 4 equations with known (x,y) values, and unknown slope and constants (m and c).

$$6 = (1)m + c$$
$$5 = (2)m + c$$
$$7 = (3)m + c$$
$$10 = (4)m + c$$

2. Put this system of linear equations into matrix-vector form.

Solution:

$\mathbf{A}$: A matrix whose rows contain coefficients for our x_i.

$$\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 2 & 1 \\ 3 & 1 \\ 4 & 1 \end{bmatrix}$$

$\vec{x}$: A vector containing the parameters that we want to be optimizing

$$\vec{x} = \begin{bmatrix} m \\ c \end{bmatrix}$$

$\vec{b}$: A vector containing the "true" y values of our original points, which we will want our $\hat{x}$ to approximate

$$\vec{b} = \begin{bmatrix} 6 \\ 5 \\ 7 \\ 10 \end{bmatrix}$$

Putting this into the standard form $\mathbf{A}\vec{x} = \vec{b}$, we get:

$$\begin{bmatrix} 1 & 1 \\ 2 & 1 \\ 3 & 1 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} m \\ c \end{bmatrix} = \begin{bmatrix} 6 \\ 5 \\ 7 \\ 10 \end{bmatrix}$$

3. Given that $(\mathbf{A}^T\mathbf{A})^{-1} = \begin{bmatrix} 0.2 & -0.5 \\ -0.5 & 1.5 \end{bmatrix}$, use the linear least squares technique you learned earlier on this overdetermined system to solve for a line of best fit

   Solution:

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

$$= \begin{bmatrix} 0.2 & -0.5 \\ -0.5 & 1.5 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 6 \\ 5 \\ 7 \\ 10 \end{bmatrix}$$

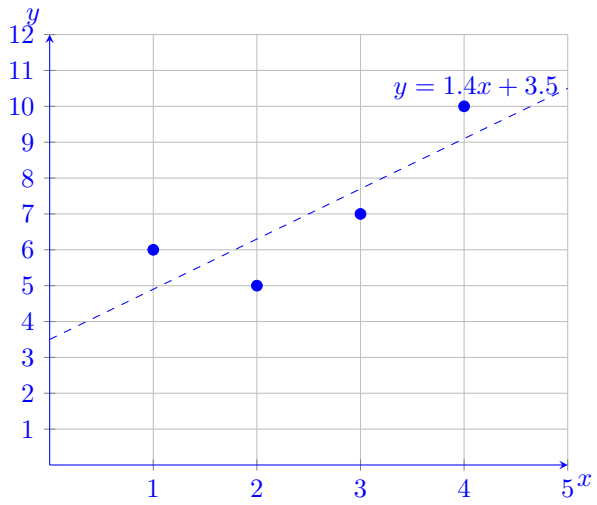
$$= \begin{bmatrix} 0.2 & -0.5 \\ -0.5 & 1.5 \end{bmatrix} \begin{bmatrix} 77 \\ 28 \end{bmatrix}$$

$$\hat{x} = \begin{bmatrix} 1.4 \\ 3.5 \end{bmatrix}$$

   Thus, we see that the line of best fit has a slope of 1.4 and a constant shift of 3.5, and the equation of the line is $y = 1.4x + 3.5$

4. Plot this line in the plot above.

   Solution:



5. C.F. Gauss (the 19th century mathematician of Gaussian elimination fame) used least squares to model the orbits of various celestial bodies around the sun. From Kepler's laws of planetary motion, Gauss constructed the following equation:

$$\alpha x^2 + \beta xy + \gamma y^2 + \delta x + \epsilon y = 1$$

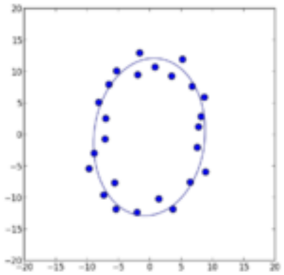
where x and y describe the position of the object in the 2D plane of its orbit around the sun, and the Greek letters are unknown coefficients. A scientist named Piazzi observed the orbit of the dwarf planet Ceres over the period of a month, producing 19 data points of its xy-coordinates. Given this data, how might Gauss have used least squares to recover appropriate values of the coefficients alpha, beta, gamma, delta, epsilon to model the orbit of Ceres?

Solution: The question we need to ask is: what are the variables we need to solve for? In this case, we have points in space: thus, we have values of x and y. Using these two values, we can also easily find x^2, xy, and y^2. So, the unknowns in this story are the coefficients of these variables, or the "weights" of each term. Thus we can set this problem for n measurements in the form of $\mathbf{A}\vec{x} = \vec{b}$:

$$\begin{bmatrix} x_1^2 & x_1y_1 & y_1^2 & x_1 & y_1 \\ x_2^2 & x_2y_2 & y_2^2 & x_2 & y_2 \\ & & \vdots & & \\ x_n^2 & x_ny_n & y_n^2 & x_n & y_n \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ \gamma \\ \delta \\ \epsilon \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$$

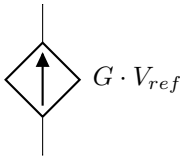
In this case, we have an overdetermined system, which (as we've explored earlier) can be solved using least squares. Simply use the least squares formula to find the best approximation to your weights $\hat{x} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\vec{b}$

Then, you can plot an ellipse of best fit that models the path of the planetary body.



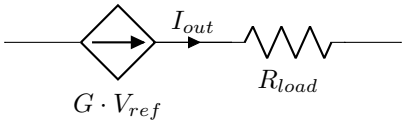
Problem 3: Circuit Design

A Voltage Controlled Current Source looks like this:



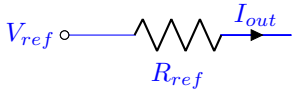
A VCCS is a Dependent Current Source that's controlled by a voltage V_{ref}, and produces a current based on that V_{ref}, which will follow the equation $I_{out} = G \cdot V_{ref}$, for some constant G.

It can be connected to any load resistor (it has a resistance of R_{load}), and guarantees that the current $G \cdot Vref$ will flow through that resistor.



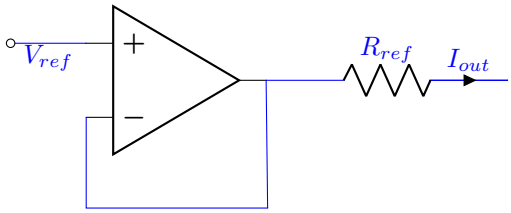
1. In order to create a VCCS, we'll need some way to turn an input voltage into a current. What's the simplest way we can accomplish this?
   Hint: What's a circuit element that relates voltage and current?

   Solution: Use a resistor! The circuit would look like this, where $I_{out} = V_{ref}/R_{ref}$:



2. We'll also need some way to isolate the input voltage from the previous parts of the circuit that produces our input voltage. In other words, if the input voltage to our VCCS has some resistors or other components connected to it, we don't want that to affect the relation between V_{ref} and I_{out} in our VCCS. What design component can we use to do this?

   Solution: Place a buffer between the input voltage V_{ref} and the resistor R_{ref}.



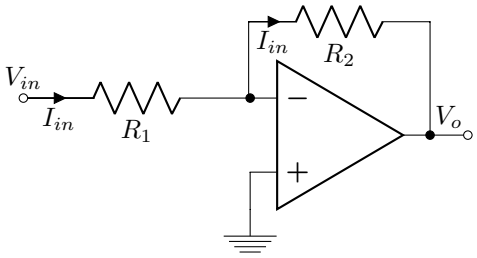
3. Now we have V_{ref} converted to I_{out} using the formula $G \cdot V_{ref}$. What is G in terms of R_{ref}?

   Solution: By Ohm's Law, the voltage dropped across the resistor is the current through it times the resistance: $I_{out} = \dfrac{V_{ref}}{R_{ref}} = GV_{ref} \implies G = \dfrac{1}{R_{ref}}$

4. Are we done? Are there any problems with the current that we're producing?
   (Hint: What happens when we place our R_{load} at I_{out}?)

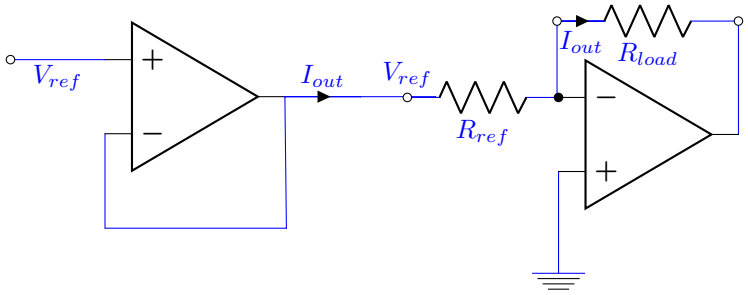
   Solution: The value of the current produced changes from $\frac{V_{ref}}{R_{ref}}$ to $\frac{V_{ref}}{R_{ref}+R_{load}}$, so our current source doesn't work the way it should.

5. We can keep the current at the value we want it to be at by using one of the properties of an inverting op amp: the fact that no matter what, the current flowing through the $V-$ to V_{out} branch is equal to the input current to $V-$ (This follows from the fact that no current flows into $V-$).



Where can we place our previous circuit and R_{load} to take advantage of this? Draw the entire circuit configuration for your VCCS.

Solution:



Problem 4: (Challenging) Disease Diagnosis

Consider a set of patients. Patient i can be represented by an attribute vector $\vec{x}^{(i)} = \begin{bmatrix} x_1^{(i)} \\ x_2^{(i)} \end{bmatrix}$ as well as a known label $y^{(i)} \in \{-1, +1\}$ indicating whether they have a disease D. We want to design a simple classifier that can use the information from the data we have in order to predict whether a patient with attributes $x_1^{(i)}$ and $x_2^{(i)}$ and unknown diagnosis status has the disease. To do this, we will use our knowledge of least squares linear regression. We would like to design a linear function

$$f\left(\vec{x}^{(i)}\right) = \vec{w}^T \vec{x}^{(i)}$$

that takes in a vector $\vec{x}^{(i)}$ for a patient i and computes $y^{(i)} = \text{sign}\left(f(\vec{x}^{(i)})\right)$ to predict whether the patient has the disease.

Here, $\vec{w}$ represents the vecotr containing all the weights (coefficients) for each of the attribute entries in the vector $\vec{x}^{(i)}$.

Note: $\text{sign}(x) = \begin{cases} +1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases}$

1. In order to make our classifier as accurate as possible, we've found the following 4 error functions that we can choose from to minimize. choose **one that best suits our model** and **explain why the other three don't work as well** in this scenario.

   - Linear Error: $\vec{e} = \vec{y}^{(i)} - f(\vec{x}^{(i)})$
   - Absolute Error: $\vec{e} = |\vec{y}^{(i)} - f(\vec{x}^{(i)})|$
   - Squared Error: $\vec{e} = \left(\vec{y}^{(i)} - f(\vec{x}^{(i)})\right)^2$
   - Sign Error: $\vec{e} = \begin{cases} +1, \text{if } \vec{y}^{(i)} = f(\vec{x}^{(i)}) \\ -1, \text{if } \vec{y}^{(i)} \neq f(\vec{x}^{(i)}) \end{cases}$

   Solution: Out of the existing options we have, squared error is the best option because it efficiently computes the deviation of our prediction from the actual labels for classifying the patients. The reasons why the other error functions don't work as well are as follows:

   - Linear Error doesn't work well because the errors are signed (take for example one point that has an error of +100 and one point that has an error of -100 (the sum of those errors is 0 despite the fact that neither of the points was perfectly described)).
   - Absolute Error doesn't work well because we can't easily work with absolute expressions algebraically.
   - Sign Error, a reasonable model, again doesn't work well because it is a piecewise function and makes optimization on all the data points much trickier.
   - **One more note,** both the absolute error and the sign error function are harder to interpret because they are **not continuous** at every point. It is easier for us to interpret the model and optimize if we have a **differentiable error function**.

   Optional: In real life, since this is a binary (two-class) classification task, we actually have some other error functions that are even better suited. You are not required to know this for the class, but in case you are interested, *cross entropy* error function is most commonly used for training binary classification models:

   $$l(\beta) = \log L(\beta) = \sum_{i=1}^{n} y_i \log h_\beta(\vec{x}_i) + (1 - y_i) \log(1 - h_\beta(\vec{x}_i)),$$

   where $(\vec{x}_i, y_i)$ are our data points, and h_β is the logistic (or sigmoid) function $h_\beta(x) = \frac{1}{1+e^{-x}}$.
   One intuitive reason we don't use squared error for a binary classification task is because it treats the outputs of our model as something continuous, while in reality, in the context of this problem, we only have 2 discrete

outputs, -1 (does not have the disease D) or +1 (has the disease D). This makes further interpreting the model and extracting information from the performance of our model nearly impossible.

2. Suppose we have a brand new set of data points as shown in the table below along with a column representing the actual diagnosis results: (+1) for positive diagnosis and (-1) for negative diagnosis. Given that we've found the weight vector $\vec{w}$ of our linear function $f(\vec{x}^{(i)})$ to be $\begin{bmatrix}2\\3\end{bmatrix}$, predict whether each patient has the disease by filling in the table with the prediction **Yes, No,** or **Inconclusive**. What is the accuracy of our linear classifier?

Patient	Attribute x_1	Attribute x_2	Diagnosis Result	Prediction from the Classifier
1	1	0	(+1)	
2	-6	1	(-1)	
3	-5	-5	(+1)	
4	9	-6	(-1)	

Note: As a refresher, here is how we will make the final prediction based on the value of our linear function:

- When $f(\vec{x}^{(i)}) > 0$, the classifier outputs **Yes**.
- When $f(\vec{x}^{(i)}) < 0$, the classifier outputs **No**.
- When $f(\vec{x}^{(i)}) = 0$, the classifier outputs **Inconclusive**.

Solution: Recall 0 is the boundary for whether a patient has the disease. $w^T\vec{x} = 2x_1 + 3x_2$. Plugging in the values of x_1 and x_2 for each row and comparing them against the boundary value (0), we obtain the following classification:

Patient	Attribute x_1	Attribute x_2	Diagnosis Result	Prediction from the Classifier
1	1	0	(+1)	Yes
2	-6	1	(-1)	No
3	-5	-5	(+1)	No
4	9	-6	(-1)	Inconclusive

We've correctly classified 2 out of the 4 patients, hence the accuracy of our model is $\frac{2}{4} = 50\%$

3. Applying the same model, we have found a different linear classifier $f(\vec{x}^{(j)}) = \vec{w}^T\vec{x}^{(j)}$ for another disease T with 2 attributes from each patient. Surprisingly, the disease has an astonishingly high accuracy of 95 % on all the patients we have trained our classifier on so far.

   (a) In order to further validate our model, we obtain a new batch of data on 50 more patients along with their diagnosis results. What can we infer about the accuracy of our classifier's performance on those 50 patients? Circle one below and provide your justification.

   At most 95% Exactly 95% At least 95% Cannot be determined

   Solution: Cannot be determined. It is possible that we overfit our classifier on all the patients we have so far, and the new testing data will bring down the accuracy of our model. It might also be possible this particular disease T exhibits a linear correlation among the given set of features.

   (b) We want a more comprehensive model and decides to add more attributes of each patient to our linear classifier. What can we most likely infer about the accuracy of our classifier on the original dataset? Circle one below and provide your justification.

   At most 95% Exactly 95% At least 95% Cannot be determined

   Solution: At least 95%. As the complexity of the model increases, the bias of the model will decrease. This means the accuracy of our classifier on the original dataset (which we are trying to optimize over)

is also guaranteed **to be at least as good as the previous model**.

Note: In the solution above, we mentioned the term "bias". In this class, you can think of bias as the length of the residual vector $\|\vec{r}\| = \|\vec{y} - f(X)\|$. It addresses **how much our prediction deviates from what we expect the results to be**, and it is closely related with the accuracy of a model on the dataset.

(**The following part is optional**) Another interesting fact to note is that, as the complexity of a model increases (with more features), while we are able to make better predictions on the original dataset we use to train our model, the "variance" of the model also increases as a result. You can think of "variance" as something that describes **how much the model specifically fits onto the training dataset and deviates from a more general and robust pattern**. A helpful example to think about is trying to fit a degree-10 polynomial on a set of points that, when plotted out, resemble much more the shape of a parabola. While the degree-10 polynomial might fit almost every single point in the original dataset perfectly. If we add in any new point, the model will likely not predict as well as if we had used a quadratic model.

Now that we are done with choosing the best error (cost) function for our linear classifier and are satisfied with its accuracy on the dataset, we have decided to dig more into the actual process of diagnosing the disease. Surprisingly, it turns out that part of the diagnosis process can be further refined using the least squares method we have learned in class!

To give a bit more context, part of the diagnosis process requires measuring neural impulses from the patient. For the simplicity of this problem, we assume the neural impulses can be measured through a sensor that converts the impulses to **discrete-time signals**. When converting the electric impulses to samples of signals and transmitting these samples to the measurement device, some samples got lost or corrupted in the process. To complicate the problem even more, the missing samples may be randomly distributed through out the signal. Our task is to fill in missing values **based on the available uncorrupted data** in order to conceal these errors (this process is formally known as *error concealment*).

4. To help us reformulate this problem as a least squares problem, we need to derive the matrix properly representing our data. The following subparts will guide you through this process.

   (a) Suppose we originally have a total of 5 samples in the signal, and during the transmission process, the 3rd and 4th samples have become unusable (either lost or corrupted). In other words, given a length 5 vector $\vec{s}$ where each entry represents a sample of the signal, $\vec{s}$ is reduced down to a length 3 vector $\vec{r}$ because the original 3rd and 4th entries can no longer be used.

$$\vec{s} = \begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \end{bmatrix} \longrightarrow \begin{bmatrix} s_1 \\ s_2 \\ s_5 \end{bmatrix} = \vec{r}.$$

   Find a matrix H that when applied to $\vec{s}$, can result in $\vec{r}$. In other words, find H such that $H\vec{s} = \vec{r}$.

   Solution: To accommodate the length of our original signal, we start with a 5×5 identity matrix I_5, and we can see that when we take out the 3rd and 4th rows in I_5, the resulting matrix, when applied on $\vec{s}$, exactly eliminates the 3rd and 4th entries as well! Hence,

$$S = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

(b) Now, let's think about the reversed process. In the context of this question, we end up receiving the signal only containing the original 1st, 2nd and 5th entries, but we would like to recover our original signal. To "re-expand" the vector back to length 5, for now, we will fill all the missing entries with 0. In other words, we have:

$$\vec{r} = \begin{bmatrix} s_1 \\ s_2 \\ s_5 \end{bmatrix} \longrightarrow \begin{bmatrix} s_1 \\ s_2 \\ 0 \\ 0 \\ s_5 \end{bmatrix} = \vec{s}'.$$

Find a matrix H' such that $H'\vec{r} = \vec{s}'$, and show that $H' = H^T$.

Solution: Observing that in H, we zeroed out the 3rd and 4th entries in $\vec{s}$ through columns of zeros in the 3rd and 4th columns of H. Similarly, we can reverse engineer the process by using rows of zeros instead in the 3rd and 4th rows. In fact, it turns out that:

$$H' = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} = H^T.$$

Hence, we can see that $H' = H^T$.

(c) Using the matrix H' we have found in the previous part, we now want to find the "*complement*" of H' (we will call it H'_c). We define the H'_c as the matrix that, when applied, expands a length 2 vector $\vec{r}_c$ containing the two missing entries in $\vec{s}$ to a length 5 vector padded with zeros. In other words, we have:

$$\vec{r}_c = \begin{bmatrix} s_3 \\ s_4 \end{bmatrix} \longrightarrow \begin{bmatrix} 0 \\ 0 \\ s_3 \\ s_4 \\ 0 \end{bmatrix} = \vec{s}'_c.$$

Find the matrix H'_c such that $H'_c\vec{r}_c = \vec{s}'_c$. For the consistency in notation for the rest of this problem, we will also re-express $H'_c = H_c^T$ (to align with H^T from the previous part).

Solution: Again, using a similar approach from the previous two parts, we can find the complement of H' to be:

$$H'_c = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} = H_c^T.$$

(d) Now that we have found the matrix H^T and its complement H_c^T, we would like to re-express our original signal vector $\vec{s}$. Rewrite the original signal vector $\vec{s}$ in terms of H^T, H_c^T, $\vec{r}$, and $\vec{r}_c$. Within this expression, what is the **unknown term** we are trying to recover?

Solution: We can rewrite our original signal vector $\vec{s}$ through the following decomposition:

$$\vec{s} = \begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \end{bmatrix} = \begin{bmatrix} s_1 \\ s_2 \\ 0 \\ 0 \\ s_5 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ s_3 \\ s_4 \\ 0 \end{bmatrix} = \vec{s}' + \vec{s}'_c = H^T\vec{r} + H_c^T\vec{r}_c.$$

Therefore, we can see that:

$$\vec{s} = H^T\vec{r} + H_c^T\vec{r}_c.$$

Within this expression, we can see that $\vec{r}_c$ is the term we are trying to recover because it contains the missing/corrupted 3rd and 4th entries from our original signal vector.

5. To complete the reformulation of this problem in a least squares setting, we would like to find an expression representing the cost of our model. Suppose we have an orthogonal square matrix D. Given our received signal $\vec{r}$ (has a length of 5; contains missing/corrupted entries as well), our original signal $\vec{s}$, and the matrix D, consider the following expression as the cost function we would like to minimize:

$$C(\vec{s}, \vec{r}) = \|\vec{s} - \vec{r}\|^2 + \|D\vec{s}\|^2$$

Specifically, for the term $\|D\vec{s}\|$, show that $\|D\vec{s}\| = \|\vec{s}\|$ and explain how it helps control the minimization process as when its value increases v.s. decreases.

Solution: Breaking the expression down into two separate norms, we can see that the first term, $\|\vec{s} - \vec{r}\|^2$, represents the squared difference between our received signal containing corrupted terms and the original signal. The second term, $\|D\vec{s}\|^2 = (D\vec{s})^T(D\vec{s}) = \vec{s}^T(D^TD)\vec{s} = \vec{s}^T\vec{s} = \|\vec{s}\|^2$, which actually just represents the squared norm of the original signal $\vec{s}$ itself! As a quick reminder, for an orthogonal square matrix D, $D^TD = I$.

Intuitively, we would like our received and corrupted signal $\vec{r}$ to be as close to the predicted original signal $\vec{s}$ as possible, so this gives rise to the first term $\|\vec{s} - \vec{r}\|^2$. The reason for having $\|D\vec{s}\|^2$ is a bit more subtle:

- By minimizing $\|D\vec{s}\|^2$, we are minimizing the overall expression, allowing $\|\vec{s} - \vec{r}\|^2$ to be closer to 0, and consequently $\vec{s}$ to be closer to $\vec{r}$ (which matches our original objective with the first term).
- If the discrepancy between $\vec{s}$ and $\vec{r}$ increases, this will also cause $\|D\vec{s}\|^2$ to increase. This is undesirable because we will have an overall higher cost, and in the process of minimizing the cost function, this will lead us to avoid such a choice of $\vec{s}$ as well.

To sum up, the term $\|D\vec{s}\|^2$ "*controls*" how the overall cost expression changes by rewarding (less cost) when it is performing well; and penalizing (more cost) when it is performing poorly.