EECS 16A Designing Information Devices and Systems I
Fall 2018 Homework 2

This homework is due September 7, 2018, at 23:59.
Self-grades are due September 11, 2018, at 23:59.

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

- `hw2.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 "printout" of your IPython notebook, you will not receive credit for problems that involve coding. Make sure that your results and your plots are visible. Assign the IPython printout to the correct problem(s) on Gradescope.

- `hw2.ipynb`: A single IPython notebook with all of your code in it.

Submit each file to its respective assignment on Gradescope.

1. Lab Grades and Discussion Attendance

Lab grades and discussion attendance will be posted on Gradescope. These will be updated periodically; check the Piazza post for the most recent updates.

For lab grades, please submit a file named `sid.txt` to the "Lab Grades" assignment. This text file should contain only your SID number and nothing else (e.g. no "" marks or extra whitespace). You can find a template in the iPython folder.

For discussion grades, submit the same file `sid.txt` to "Discussion Attendance."

If you use a strange text editor, the autograder will give a strange error message complaining about the unicode format. If this happens, please use a different editor and reupload.

Describe how to find your lab and discussion grades.

Solution: Get the template file from the iPython folder and replace the numbers with your student ID without changing anything else. Submit this file to the assignment "Lab Grades" on Gradescope.

For the discussion grade, submit the same file to "Discussion Attendance."

2. (PRACTICE) Finding Charges from Potential Measurements

Solution: `#modeling` `#matrixNotation`

We have three point charges Q_1, Q_2, and Q_3 whose positions are known, and we want to determine their charges. In order to do that, we take three electric potential measurements U_1, U_2 and U_3 at three different locations. You do not need to understand electric potential to do this problem. The locations of the charges and potentials are shown in Figure 1.

For the purpose of this problem, the following equation is true:

$$U = k\frac{Q}{r}$$

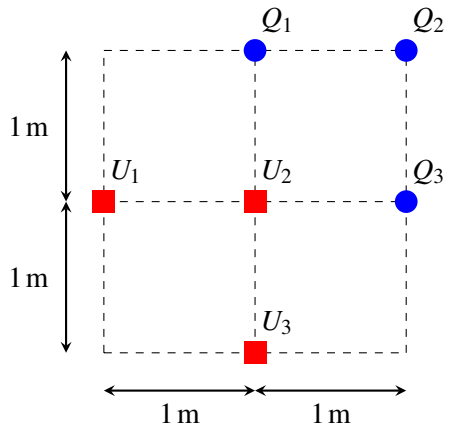


Figure 1: Locations of the charges and potentials.

at a point r meters away (for some fixed physical constant k; this problem does not require the numerical value of k).

Furthermore, the potential contributions from different point charges add up linearly. For example, in the setup of Figure 1, the potential measured at point U_2 is

$$U_2 = k\frac{Q_1}{1} + k\frac{Q_2}{\sqrt{2}} + k\frac{Q_3}{1}.$$

Given that the actual potential measurements in the setup of Figure 1 are

$$U_1 = k\frac{4+3\sqrt{5}+\sqrt{10}}{2\sqrt{5}},$$
$$U_2 = k\frac{2+4\sqrt{2}}{\sqrt{2}},$$
$$U_3 = k\frac{4+\sqrt{5}+3\sqrt{10}}{2\sqrt{5}},$$

write the system of linear equations relating the potentials to charges. Solve the system of linear equations to find the charges Q_1, Q_2, Q_3. You may use your IPython notebook to solve the system.

IPython hint: For constants a_i, b_i, c_i, y_i, you can solve the system of linear equations

$$a_1x_1 + a_2x_2 + a_3x_3 = y_1$$
$$b_1x_1 + b_2x_2 + b_3x_3 = y_2$$
$$c_1x_1 + c_2x_2 + c_3x_3 = y_3$$

in IPython with the following code:

```
import numpy as np
a = np.array([
    [a1, a2, a3],
    [b1, b2, b3],
    [c1, c2, c3]
```

```
])
b = np.array([y1, y2, y3])
x = np.linalg.solve(a, b)
print(x)
```

The square root of a number a can be written as `np.sqrt(a)` in IPython.

Solution:

Let us denote the distance to the ith potential measurement point from the jth source as r_{ij}. By using the Pythagorean theorem (formula for the length of the hypotenuse of a right triangle), we get the following:

$$\begin{aligned} r_{1,1} &= \sqrt{2}, & r_{1,2} &= \sqrt{5}, & r_{1,3} &= 2, \\ r_{2,1} &= 1, & r_{2,2} &= \sqrt{2}, & r_{2,3} &= 1, \\ r_{3,1} &= 2, & r_{3,2} &= \sqrt{5}, & r_{3,3} &= \sqrt{2}. \end{aligned}$$

Then, the potentials U_1, U_2 and U_3 can be expressed as

$$\begin{aligned} k\left(\frac{Q_1}{\sqrt{2}} + \frac{Q_2}{\sqrt{5}} + \frac{Q_3}{2}\right) &= U_1, \\ k\left(Q_1 + \frac{Q_2}{\sqrt{2}} + Q_3\right) &= U_2, \\ k\left(\frac{Q_1}{2} + \frac{Q_2}{\sqrt{5}} + \frac{Q_3}{\sqrt{2}}\right) &= U_3. \end{aligned}$$

Plugging in the values of U_1, U_2 and U_3 and dividing by k, we get the following system of equations:

$$\begin{aligned} \frac{Q_1}{\sqrt{2}} + \frac{Q_2}{\sqrt{5}} + \frac{Q_3}{2} &= \frac{4+3\sqrt{5}+\sqrt{10}}{2\sqrt{5}} \\ Q_1 + \frac{Q_2}{\sqrt{2}} + Q_3 &= \frac{2+4\sqrt{2}}{\sqrt{2}} \\ \frac{Q_1}{2} + \frac{Q_2}{\sqrt{5}} + \frac{Q_3}{\sqrt{2}} &= \frac{4+\sqrt{5}+3\sqrt{10}}{2\sqrt{5}} \end{aligned}$$

The system can be solved by the following IPython code:

```
import numpy as np

r11 = np.sqrt(2); r12 = np.sqrt(5); r13 = 2
r21 = 1; r22 = np.sqrt(2); r23 = 1
r31 = 2; r32 = np.sqrt(5); r33 = np.sqrt(2)

y1 = (4 + 3*np.sqrt(5) + np.sqrt(10)) / (2*np.sqrt(5))
y2 = (2 + 4*np.sqrt(2)) / (np.sqrt(2))
y3 = (4 + np.sqrt(5) + 3*np.sqrt(10)) / (2*np.sqrt(5))

a = np.array([
    [1/r11, 1/r12, 1/r13],
    [1/r21, 1/r22, 1/r23],
```

```
    [1/r31, 1/r32, 1/r33]
])
b = np.array([y1, y2, y3])
x = np.linalg.solve(a, b)
print(x)
```

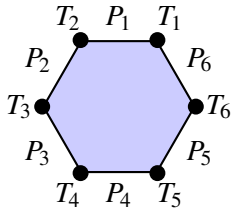
The result is $Q_1 = 1$, $Q_2 = 2$, and $Q_3 = 3$.

3. Figuring Out The Tips

A number of people gather around a round table for a dinner. Between every adjacent pair of people, there is a plate for tips. When everyone has finished eating, each person places half their tip in the plate to their left and half in the plate to their right. In the end, of the tips in each plate, some of it is contributed by the person to its right, and the rest is contributed by the person to its left. Suppose you can only see the plates of tips after everyone has left. Can you deduce everyone's individual tip amounts?

Note: For this question, if we assume that tips are positive, we need to introduce additional constraints enforcing that, and we wouldn't get a linear system of equations. Therefore, we are going to ignore this constraint and assume that negative tips are acceptable.

(a) Suppose 6 people sit around a table and there are 6 plates of tips at the end.



If we know the amounts in every plate of tips (P_1 to P_6), can we determine the individual tips of all 6 people (T_1 to T_6)? If yes, explain why. If not, give two different assignments of T_1 to T_6 that will result in the same P_1 to P_6.

Solution:

No, this is not possible to determine in general. For example, the following two different assignments of tip amounts for each person:

$$(T_1, T_2, T_3, T_4, T_5, T_6) = (2, 0, 2, 0, 2, 0)$$
$$(T_1, T_2, T_3, T_4, T_5, T_6) = (0, 2, 0, 2, 0, 2)$$

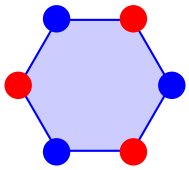
Will both result in $(P_1, P_2, P_3, P_4, P_5, P_6) = (1, 1, 1, 1, 1, 1)$.

If we write down the system of linear equations:

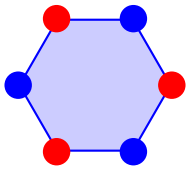
$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \\ T_6 \end{bmatrix} = \begin{bmatrix} 2P_1 \\ 2P_2 \\ 2P_3 \\ 2P_4 \\ 2P_5 \\ 2P_6 \end{bmatrix}$$

We can use Gaussian elimination to reduce the last row to all zeros. Therefore, equations are linearly dependent. However, Gaussian elimination is not needed to solve this question.

Intuitively, we can color each spot on the table alternating between red or blue:

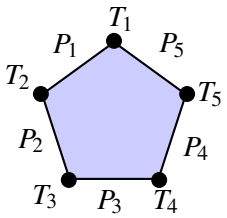


Then supposing that everyone sitting at red spots all tip r dollars and everyone sitting at blue spots all tip b dollars, we find $P_1, \cdots P_6$ dollars on the plates. However, this is no different from this following coloring:



Thus, we see that because of the special symmetry of the six-sided table, it's not possible to deduce everyone's tip.

(b) The same question as above, but what if we have 5 people sitting around a table?



Solution:

Yes. The problem can be reduced to the following system of equations:

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \end{bmatrix} = \begin{bmatrix} 2P_1 \\ 2P_2 \\ 2P_3 \\ 2P_4 \\ 2P_5 \end{bmatrix}$$

We can then run row-reduction on this matrix. First subtract rows 1 and 3 and add rows 2 and 4 to row 5:

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \end{bmatrix} = \begin{bmatrix} 2P_1 \\ 2P_2 \\ 2P_3 \\ 2P_4 \\ P_5 - P_1 + P_2 - P_3 + P_4 \end{bmatrix}$$

Now we subtract row 5 from row 4, row 4 from row 3, row 3 from row 2 and row 2 from row 1:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \end{bmatrix} = \begin{bmatrix} P_1 - P_2 + P_3 - P_4 + P_5 \\ P_2 - P_3 + P_4 - P_5 + P_1 \\ P_3 - P_4 + P_5 - P_1 + P_2 \\ P_4 - P_5 + P_1 - P_2 + P_3 \\ P_5 - P_1 + P_2 - P_3 + P_4 \end{bmatrix}$$

Thus, we have a unique solution for T_1 to T_5 in terms of P_1 to P_5.

Intuitively, unlike the argument for the previous part, since there is an odd number of seats, it's not possible for people to color every alternate seat red or blue.

(c) If n is the total number of people sitting around a table, for which n can you figure out everyone's tip? You do not have to rigorously prove your answer.

Solution:

Note: Although you didn't need to prove your answers rigorously, we will give you a rigorous argument. As long as your answer has the flavor of an argument (or an another equally sound argument), you should give yourself full credit.

For even n, this is not possible. Here is a counterexample: Suppose that all the plates had \$1 in them. There are clearly two different ways that this could have happened.

First:

$$T_n = \begin{cases} 2 & n \text{ odd} \\ 0 & n \text{ even} \end{cases}$$

Second:

$$T_n = \begin{cases} 0 & n \text{ odd} \\ 2 & n \text{ even} \end{cases}$$

Both cases will result in $P_i = 1$ for all i from 1 to n. Therefore, we cannot figure out everyone's tip.

We can determine everyone's tips for all odd n. You can either argue with Gaussiam elimination on a general $n \times n$ matrix where n is odd or use the second argument, which does not use Gaussian elimination:

Gaussian elimination solution:

For odd n, the matrix encoding the system of linear equations is:

$$\left[\begin{array}{cccccc} 1 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 1 & 0 & \cdots & 0 \\ & & \ddots & \ddots & & \\ 0 & 0 & \cdots & 1 & 1 & 0 \\ 0 & 0 & \cdots & 0 & 1 & 1 \\ 1 & 0 & \cdots & 0 & 0 & 1 \end{array}\right] \begin{array}{l} \text{Row } 1 \\ \text{Row } 2 \\ \vdots \\ \text{Row } n-2 \\ \text{Row } n-1 \\ \text{Row } n \end{array}$$

We want to perform Gaussian elimination on this matrix. First, we subtract all odd-numbered rows from row n and add all even-numbered rows to row n. What is row n in the end? Denote the i-th item in row n by $R_{n,i}$. We know that $R_{n,1} = 0$, and for $i = 2, \cdots, n-2$, $R_{n,i} = 1 - 1 = 0$. Since n is odd, row $n-2$ is subtracted from row n, and row $n-1$ is added to row n. Therefore, $R_{n,n-1} = 0$ and $R_{n,n} = 2$. Now we can divide row n by 2, then subtract row n from row $n-1$, row $n-1$ from row $n-2$, and so on, until we get the identity matrix. Therefore we can see that all the rows are linearly independent and we can obtain a unique solution to this system of equations.

Alternate solution:

Suppose that each customer tipping: $T_1 = a_1, \cdots, T_n = a_n$ gives rise to the amount in plates $P_1 = p_1, \cdots, P_n = p_n$. Now suppose there exist another different solution to $T_1, \cdots, T_n$ that gives the same amount in plates. Then for some i two possible values for T_i, a_i and $a_i + \varepsilon$, are possible. Then:

$$\begin{aligned} a_i + a_{i+1} &= 2p_i \\ T_{i+1} &= 2p_i - T_i \\ &= (a_i + a_{i+1}) - (a_i + \varepsilon) \\ &= a_{i+1} - \varepsilon \end{aligned}$$

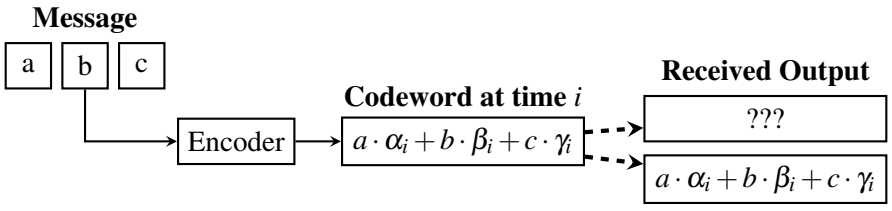
This means that $T_{i+1} = a_{i+1} - \varepsilon$, $T_{i+2} = a_{i+2} + \varepsilon$ and so on. We can then keep going around the circle, noting that there are an odd amount of sign flips in total before we get back to T_i. Therefore, we eventually get $T_i = a_i - \varepsilon$. However, we are assuming that $T_i = a_i + \varepsilon$ in the beginning. Therefore ε must be zero and we conclude that there is one unique solution.

4. Fountain Codes

Consider a sender, Alice, and a receiver, Bob. Alice wants to send a message to Bob, but the message is too big to send all at once. Instead, she breaks her message up into little chunks that she sends across a wireless channel one at a time (think radio transmitter to antenna). She knows some of the packets will be corrupted or erased along the way (someone might turn a microwave on...), so she needs a way to protect her message from errors. This way, even if Bob gets a message missing parts of words, he can still figure out what Alice is trying to say! One coding strategy is to use fountain codes. Fountain codes are a type of error-correcting codes based on principles of linear algebra. They were actually developed right here at Berkeley! The company that commercialized them, Digital Fountain, (started by a **Berkeley grad, Mike Luby**), was later acquired by Qualcomm. In this problem, we will explore some of the underlying principles that make fountain codes work in a very simplified setting.

In this problem, we concentrate on the case with transmission erasures, i.e. where a bad transmission causes some parts of the message to be erased. Let us say Alice wants to convey the set of her three favorite ideas covered in EE16A lecture each day to Bob. For this, three real numbers a, b, c can be used to represent three different ideas and convey the 3 element vector $\begin{bmatrix} a \\ b \\ c \end{bmatrix}$ (let us say there are an infinite number of ideas covered in EE16A).

At each time step, she can send one number, which we will call a "symbol" across, so one possible way for her to send the message is to first send a, then send b, and then send c. However, this method is particularly susceptible to losses. For instance, if the first symbol is lost, then Bob will receive $\begin{bmatrix} ? \\ b \\ c \end{bmatrix}$, and he will have no way of knowing what Alice's favorite idea is.



(a) The main idea in coding for erasures is to send redundant information, so that we can recover from losses. Thus, if we have three symbols of information, we might transmit six symbols for redundancy. One of the most naive codes is called the repetition code. Here, Alice would transmit $\begin{bmatrix} a \\ b \\ c \\ a \\ b \\ c \end{bmatrix}$. How many lost symbols can this transmission recover from? Are there specific patterns it cannot handle?

Solution:

This pattern is robust as long as we have one of each point below:

- first or fourth entry
- second or fifth entry
- third or sixth entry

This means that, if we have a channel that erases every third element, we cannot recover the message. If it erases every third element and even more, then we obviously also cannot recover the message since more erasures just make things worse.

(b) A better strategy for transmission is to send linear combinations of symbols. Alice and Bob decide in advance on a collection of 3 element transmission row vectors $\vec{v}_i^T = \begin{bmatrix}\alpha_i & \beta_i & \gamma_i\end{bmatrix}$, $1 \leq i \leq 6$. (The ^{T} is the transpose operator, which converts row vectors to column vectors and vice-versa.) These transmission row vectors define the code: at time i, Alice transmits the scalar

$$k_i = \vec{v}_i^T \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \alpha_i a + \beta_i b + \gamma_i c.$$

Let $\vec{k}$ represent the vector Alice sends to Bob, where

$$\vec{k} = \begin{bmatrix} k_1 \\ k_2 \\ k_3 \\ k_4 \\ k_5 \\ k_6 \end{bmatrix}$$

.

Write $\vec{k}$ as a product of a matrix and a vector.

Solution:

Let $\vec{k}$ be the transmitted message (output of the encoding). Then, we can write the encoding as the operation below:

$$\vec{k} = \begin{bmatrix} \alpha_1 & \beta_1 & \gamma_1 \\ \alpha_2 & \beta_2 & \gamma_2 \\ \alpha_3 & \beta_3 & \gamma_3 \\ \alpha_4 & \beta_4 & \gamma_4 \\ \alpha_5 & \beta_5 & \gamma_5 \\ \alpha_6 & \beta_6 & \gamma_6 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} \alpha_1 a + \beta_1 b + \gamma_1 c \\ \alpha_2 a + \beta_2 b + \gamma_2 c \\ \alpha_3 a + \beta_3 b + \gamma_3 c \\ \alpha_4 a + \beta_4 b + \gamma_4 c \\ \alpha_5 a + \beta_5 b + \gamma_5 c \\ \alpha_6 a + \beta_6 b + \gamma_6 c \end{bmatrix}$$

(c) What are the transmission row vectors $\vec{v}_i^T = \begin{bmatrix}\alpha_i & \beta_i & \gamma_i\end{bmatrix}$, $1 \leq i \leq 6$ that generate the repetition code strategy in part (a)?

Solution:

The vectors are as below:

$$\vec{v}_1^T = \vec{v}_4^T = \begin{bmatrix}1 & 0 & 0\end{bmatrix}$$
$$\vec{v}_2^T = \vec{v}_5^T = \begin{bmatrix}0 & 1 & 0\end{bmatrix}$$
$$\vec{v}_3^T = \vec{v}_6^T = \begin{bmatrix}0 & 0 & 1\end{bmatrix}$$

(d) Suppose now they choose a collection of seven transmission row vectors:

$$\vec{v}_1^T = \begin{bmatrix}1 & 0 & 0\end{bmatrix}, \quad \vec{v}_2^T = \begin{bmatrix}0 & 1 & 0\end{bmatrix}, \quad \vec{v}_3^T = \begin{bmatrix}0 & 0 & 1\end{bmatrix}, \quad \vec{v}_4^T = \begin{bmatrix}1 & 1 & 0\end{bmatrix},$$
$$\vec{v}_5^T = \begin{bmatrix}1 & 0 & 1\end{bmatrix}, \quad \vec{v}_6^T = \begin{bmatrix}0 & 1 & 1\end{bmatrix}, \quad \vec{v}_7^T = \begin{bmatrix}1 & 1 & 1\end{bmatrix}$$

Again, at time i, Alice transmits the scalar $k_i = \vec{v}_i^T \begin{bmatrix}a\\b\\c\end{bmatrix}$.

Bob would like to find the transmitted message. Suppose, using this collection of transmission row vectors, Bob receives $\begin{bmatrix}7 & ? & ? & 3 & 4 & ? & ?\end{bmatrix}^T$. Can you express the problem as a system of linear equations using matrix/vector notation? What was the transmitted message?

Solution:

Since we are using the 1st, 4th, and 5th vectors, we can represent the transmission as the matrix multiplication below:

$$\begin{bmatrix}1 & 0 & 0\\1 & 1 & 0\\1 & 0 & 1\end{bmatrix}\begin{bmatrix}a\\b\\c\end{bmatrix} = \begin{bmatrix}7\\3\\4\end{bmatrix}$$

which can also be written as a system of linear equations

$$\begin{cases} a = 7 \\ a+b = 3 \\ a+c = 4 \end{cases}$$

The solution to this equation is $a = 7, b = -4, c = -3$.

(e) They continue to use the transmission row vectors from part (d). Under what conditions, (i.e. what patterns of losses) can Bob still recover the message?

Solution:

Clearly, at least three $\vec{v}_i$'s are needed to find a, b, c, so Bob cannot recover the message if more than four losses occur. Can he recover when there are exactly four losses? Sometimes yes and sometimes no. For example, $\{\vec{v}_1, \vec{v}_2, \vec{v}_3\}$ are clearly sufficient to recover the message, but $\{\vec{v}_1, \vec{v}_2, \vec{v}_4\}$ are insufficient because $\vec{v}_4 = \vec{v}_1 + \vec{v}_2$.

If fewer than four losses occur, the message can always be recovered. To see this, divide the $\vec{v}_i$'s into three groups:

- $G_1 = \{\vec{v}_1, \vec{v}_2, \vec{v}_3\}$
- $G_2 = \{\vec{v}_4, \vec{v}_5, \vec{v}_6\}$
- $G_3 = \{\vec{v}_7\}$

and consider the following cases when there are three losses (i.e. four successful transmissions):

- All three members from G_1 are transmitted: clearly $\vec{v}_1, \vec{v}_2, \vec{v}_3$ are sufficient to decode.
- Two members from G_1 are transmitted: only one of the vectors in G_2 is linearly dependent on two vectors from G_1 (that is the sum of the two vectors), and $\vec{v}_7$ is linearly independent of any two from G_1, so with four successful transmission, you are guaranteed to have an extra linearly independent vector and will be able to decode.
- Only one member from G_1 is transmitted: then all except one of G_2 and G_3 were successful. If $\vec{v}_7$ was unsuccessful, then all of G_2 succeeded, and G_2 is linearly independent and sufficient to decode. If $\vec{v}_7$ was successful, $\vec{v}_7$ is linearly independent of every set of two vectors from G_2, so we have three linearly independent vectors and can decode.

- Zero members of G_1 are transmitted: that means all of G_2 was successful, which has three linearly independent vectors, so Bob can decode.

We have provided IPython code in the accompanying notebook that reproduces the message in the next subproblem using every combination of 4 vectors. This is just for your benefit, we didn't expect you to necessarily do this yourself.

Full credit on this subproblem must identify that this scheme can always recover from three losses and sometimes four. If you did not fully justify why, do not take off credit. We didn't expect an explanation this detailed.

(f) Fountain codes build on the principles explored in this problem. The basic idea used by these codes is that Alice keeps sending linear combinations of symbols until Bob has received enough to decode the message. So at time 1, Alice sends the linear combination using $\vec{v}_1$, at time 2 she sends the linear combination using $\vec{v}_2$ and so on. After each new linear combination is sent, Bob will send back an acknowledgement *if* he can decode her message (i.e. figure out the original $\begin{bmatrix} a \\ b \\ c \end{bmatrix}$ that she intended to communicate). So clearly, the minimum number of transmissions for Alice is 3. If Bob receives the first three linear combinations that are sent, Alice is done in three steps! But because of erasures, she might hear nothing (he hasn't decoded yet). Suppose Alice used

$$\vec{v}_1^T = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}, \quad \vec{v}_2^T = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}, \quad \vec{v}_3^T = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$$

as her first three vectors, but she has still not received an acknowledgement from Bob. Should she choose new vectors according to the strategy in part (a) or the strategy in part (d)? Why?

Solution:

We can see the strategy in part (d) is more robust than that of part (a) since the strategy in part (d) can recover from more patterns of losses than the strategy in part (a) can. (d) can recover from any three losses, whereas (a) cannot.

Real-world fountain-codes are based on the idea of constantly generating new vectors $\vec{v}_i$ according to a known random seed. This way, they constantly provide fresh new mixtures of the underlying information. As soon as the receiver gets enough linearly independent measurements, it can decode. These sorts of ideas are also used in file-sharing and content-dissemination networks. You can learn more about those applications in EE121. More material on error-correcting codes will also appear in EE16B as well as in the third course in this sequence, CS70. In EE16B, you will see a connection between polynomials and subspaces that can be used to build codes robust to erasures. And in CS70, you will learn about finite fields that show how these ideas can work in the digital world without real numbers.

5. Kinematic Model for a Simple Car Building a self-driving car first requires understanding the basic motions of a car. In this problem, we will explore how to model the motion of a car.

There are several models that we can use to model the motion of a car. Assume we use the following kinematic model, given in the following four equations and Figure 2.

$$x[k+1] = x[k] + v[k]\cos(\theta[k])\Delta t \tag{1}$$
$$y[k+1] = y[k] + v[k]\sin(\theta[k])\Delta t \tag{2}$$
$$\theta[k+1] = \theta[k] + \frac{v[k]}{L}\tan(\phi[k])\Delta t \tag{3}$$
$$v[k+1] = v[k] + a[k]\Delta t \tag{4}$$

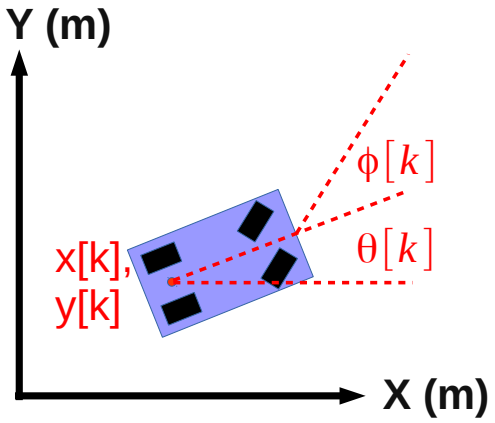


Figure 2: Vehicle Kinematic Model

where

- k, a nonnegative integer, indicates the time step at which we measure the variable (e.g. $v[k]$ is the speed at time step k and $v[k+1]$ is the speed at the following time step)
- $x[k]$ and $y[k]$ denote the coordinates of the vehicle (meters)
- $\theta[k]$ denotes the heading of the vehicle, or the angle with respect to the x-axis (radians)
- $v[k]$ is the speed of the car (meters per second)
- $a[k]$ is the acceleration of the car (meters per second squared)
- $\phi[k]$ is the steering angle input we command (radians)
- Δt is a constant measuring the time difference (in seconds) between time steps $k+1$ and k
- L is a constant and is the length of the car (in meters)

For this problem, let L be 1.0 meter and Δt be 0.1 seconds.

The variables $x[k], y[k], \theta[k], v[k]$ describe the **state** of the car at time step k. The state captures all the information needed to fully determine the current position, speed, and heading of the car. The **inputs** at time step k are $a[k]$ and $\phi[k]$. These are provided by the driver. The current value of these inputs, along with the current state of the vehicle, will determine the state of the vehicle at the next time step.

We note that the problem is nonlinear, due to the sine, cosine and tangent functions, as well as terms including the product of states and inputs.

The purpose of this problem is to show that we can approximate a nonlinear model with a simple linear model and do reasonably well. This is why, despite many systems being nonlinear, linear algebra tools are widely used in practice.

For Parts (b) - (d), fill out the corresponding sections in prob2.ipynb.

(a) We assume that the car has a small heading ($\theta \approx 0$) and that the steering angle is also small ($\phi \approx 0$), where $\approx$ means "approximately equal to." In this case, we could use the following approximations:

$$\sin(\alpha) \approx 0,$$
$$\cos(\alpha) \approx 1,$$
$$\tan(\alpha) \approx 0.$$

where α is the small angle of interest. Here, we use a very simple approximation for small angles; in later classes, you may learn better approximations.

Draw, by hand, graphs of of $\sin(\alpha)$ and $\cos(\alpha)$, for α ranging from $-\pi$ to π. Using these graphs can you justify the approximation we are making for small values of α?

Solution:

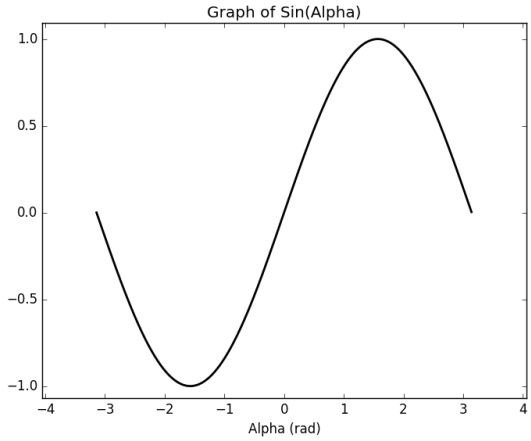


Figure 3: Graph of $sin(\alpha)$

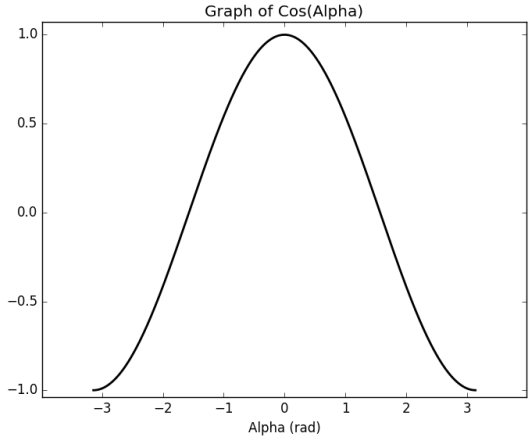


Figure 4: Graph of $cos(\alpha)$

From Fig. 3, we see that sine is close to zero for angles close to zero. From Fig. 4, we see that cosine is close to one for angles close to zero,

(b) Applying the approximation described in the previous part, write down a linear system that approximates the nonlinear vehicle model given above in Equations (1) to (4). In particular, find the 4 x 4 matrix **A** and 4 x 2 matrix **B** that satisfy the equation given below.

$$\begin{bmatrix} x[k+1] \\ y[k+1] \\ \theta[k+1] \\ v[k+1] \end{bmatrix} = \mathbf{A} \begin{bmatrix} x[k] \\ y[k] \\ \theta[k] \\ v[k] \end{bmatrix} + \mathbf{B} \begin{bmatrix} a[k] \\ \phi[k] \end{bmatrix}$$

Solution: In the region where both θ and ϕ are very small, we get that:

$$\begin{aligned} \sin(\theta) &\approx 0, \\ \cos(\theta) &\approx 1, \\ \tan(\phi) &\approx 0. \end{aligned}$$

So the vehicle equations simplify to a linear form:

$$\begin{aligned} x[k+1] &= x[k] + v[k]\Delta t \\ y[k+1] &= y[k] + 0 \\ \theta[k+1] &= \theta[k] + 0 \\ v[k+1] &= v[k] + a[k]\Delta t \end{aligned}$$

This corresponds to the following linear model:

$$\begin{bmatrix} x[k+1] \\ y[k+1] \\ \theta[k+1] \\ v[k+1] \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & \Delta t \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x[k] \\ y[k] \\ \theta[k] \\ v[k] \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ \Delta t & 0 \end{bmatrix} \begin{bmatrix} a[k] \\ \phi[k] \end{bmatrix}$$

Note to graders: Some students may have **A** with element (3,3) set to zero, but otherwise have the right **A** and **B** matrices. Don't take off credit if this is the case.

While the intention of the problem was that $\theta[k]$ was very small, but not necessarily zero, this was not clear. The answers to part C and D will be unaffected by the value in $\mathbf{A}_{33}$.

(c) Suppose we drive the car so that the direction of travel is aligned with the x-axis, and we are driving nearly straight, i.e. the steering angle is $\phi[k] = 0.0001$ radians. (Driving exactly straight would have the steering angle $\phi[k] = 0$ radians.) The initial state and input are:

$$\begin{bmatrix} x[0] \\ y[0] \\ \theta[0] \\ v[0] \end{bmatrix} = \begin{bmatrix} 5.0 \\ 10.0 \\ 0.0 \\ 2.0 \end{bmatrix}$$

$$\begin{bmatrix} a[k] \\ \phi[k] \end{bmatrix} = \begin{bmatrix} 1.0 \\ 0.0001 \end{bmatrix}$$

You can use these values in the IPython notebook to compare how the nonlinear system evolves in comparison to the linear approximation that you made. The IPython notebook simulates the car for ten time steps. Are the trajectories similar or very different? Why?

Solution: Yes, the linear model is a good approximation. This is since we are looking at a state and input that satisfy the two approximations we made with θ and ϕ. For example, the nonlinear model predicts that the state at time 10 is:

$$\begin{bmatrix} x[10] \\ y[10] \\ \theta[10] \\ v[10] \end{bmatrix} = \begin{bmatrix} 7.449 \\ 10.00 \\ 0.0000245 \\ 3 \end{bmatrix}$$

The state after 10 steps predicted by the linear model is very close.

$$\begin{bmatrix} x[10] \\ y[10] \\ \theta[10] \\ v[10] \end{bmatrix} = \begin{bmatrix} 7.45 \\ 10.0 \\ 0.0 \\ 3 \end{bmatrix}$$

From the plot, we see that both models show the vehicle moving along the x-axis and agree relatively well.

The linear model captures the state evolution well in the case where we drive mostly straight with a tiny heading angle. The models are close because the heading angle is tiny enough that the approximations are valid.

One important thing to note is that if we keep predicting forward in time, the heading angle will increase and the linear model will break down. So we usually use linear models for short predictions near the current state.

(d) Now suppose we drive the vehicle from the same starting state, but we turn left instead of going straight, i.e. the steering angle is $\phi[k] = 0.5$ radians. The initial state and input are:

$$\begin{bmatrix} x[0] \\ y[0] \\ \theta[0] \\ v[0] \end{bmatrix} = \begin{bmatrix} 5.0 \\ 10.0 \\ 0.0 \\ 2.0 \end{bmatrix}$$

$$\begin{bmatrix} a[k] \\ \phi[k] \end{bmatrix} = \begin{bmatrix} 1.0 \\ 0.5 \end{bmatrix}$$

You can use these values in the IPython notebook to compare how the nonlinear system evolves in comparison to the linear approximation that you made. The IPython notebook simulates the car for ten time steps. Are the trajectories similar or very different? Why?

Solution: No, the linear model we found breaks down in this case, because we have a high steering angle (ϕ = 0.5 radians is a steering angle of roughly 30°). The linear model predicts the same next state as in the previous part. The nonlinear model, on the other hand, has a different prediction after ten time steps.

$$\begin{bmatrix} x[10] \\ y[10] \\ \theta[10] \\ v[10] \end{bmatrix} = \begin{bmatrix} 6.88 \\ 11.3 \\ 1.34 \\ 3 \end{bmatrix}$$

The linear model predicts $[7.45, 10, 0, 3]^T$, and is not close to the prediction of the nonlinear model at all. This is because the linear model has no dependence on $\phi[k]$ — notice the column of zeros in the matrix B. Since $\phi[k] = 0.5$ is very different from zero, the approximation breaks down. To approximate this better we would need to approximate sine and tangent better. In the plot, we notice the nonlinear model predicts the car will turn left. The linear model still predicts that the car will go straight.

6. Show It

Let n be a positive integer. Let $\{\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_k\}$ be a set of k linearly dependent vectors in $\mathbb{R}^n$. Show that for any $n \times n$ matrix $\mathbf{A}$, the set $\{\mathbf{A}\vec{v}_1, \mathbf{A}\vec{v}_2, \ldots, \mathbf{A}\vec{v}_k\}$ is a set of linearly dependent vectors. Make sure that you prove this rigorously for all possible matrices $\mathbf{A}$.

Solution:

It is given that $\{\vec{v}_i \in \mathbb{R}^n | i = 0, 1, \ldots, k\}$ is a set of linearly dependent vectors. The definition of a set of linear dependent vectors states that there exist k scalars, $\{\alpha_1, \alpha_2, \ldots, \alpha_k\}$ in $\mathbb{R}$ that are *not all equal to zero simultaneously* (or equivalently *at least one of which is not equal to zero*), such that

$$\alpha_1 \cdot \vec{v}_1 + \alpha_2 \cdot \vec{v}_2 + \cdots + \alpha_k \cdot \vec{v}_k = \vec{0}. \tag{5}$$

By left-multiplying Equation 5 with $\mathbf{A}$, we get

$$\mathbf{A}(\alpha_1 \cdot \vec{v}_1 + \alpha_2 \cdot \vec{v}_2 + \cdots + \alpha_k \cdot \vec{v}_k) = \mathbf{A}\vec{0}.$$

First, note that $\mathbf{A}\vec{0} = \vec{0}$. Moreover, if we distribute $\mathbf{A}$, we get

$$\mathbf{A}(\alpha_1 \cdot \vec{v}_1) + \mathbf{A}(\alpha_2 \cdot \vec{v}_2) + \cdots + \mathbf{A}(\alpha_k \cdot \vec{v}_k) = \vec{0}.$$

From associativity of multiplication, we can bring the matrix $\mathbf{A}$ inside the parentheses and move the vector $\vec{v}$ outside of the parentheses and get

$$(\mathbf{A}\alpha_1)\vec{v}_1 + (\mathbf{A}\alpha_2)\vec{v}_2 + \cdots + (\mathbf{A}\alpha_k)\vec{v}_k = \vec{0}.$$

Since scalar-matrix multiplication is commutative, we can move the scalar α in front of the matrix $\mathbf{A}$ and get

$$(\alpha_1\mathbf{A})\vec{v}_1 + (\alpha_2\mathbf{A})\vec{v}_2 + \cdots + (\alpha_k\mathbf{A})\vec{v}_k = \vec{0}.$$

By using associativity of multiplication again, we can move the scalar α outside of the parentheses and bring the vector $\vec{v}$ into the parentheses and get

$$\alpha_1 \cdot (\mathbf{A}\vec{v}_1) + \alpha_2 \cdot (\mathbf{A}\vec{v}_2) + \cdots + \alpha_k \cdot (\mathbf{A}\vec{v}_k) = \vec{0}. \tag{6}$$

Therefore, the same k scalars, $\{\alpha_1, \alpha_2, \ldots, \alpha_k\}$, show the linear dependence of the vectors $\{\mathbf{A}\vec{v}_1, \mathbf{A}\vec{v}_2, \ldots, \mathbf{A}\vec{v}_k\}$, as requested.

■

To better understand what the proof is showing, here is a numerical example.

We will assume that we have 4 vectors, so $k = 4$ and that they are all two dimensional, so $n = 2$.

$$\vec{v}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \vec{v}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \vec{v}_3 = \begin{bmatrix} -1 \\ 1 \end{bmatrix} \vec{v}_4 = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$$

In order to show that these 4 vectors are linearly dependent, we need at least one non-zero scalar that makes the linear combination of these vectors add to zero.

$$\alpha_1 \cdot \vec{v}_1 + \alpha_2 \cdot \vec{v}_2 + \alpha_3 \cdot \vec{v}_3 + \alpha_4 \cdot \vec{v}_4 = \vec{0}.$$

This is one set of scalars that satisfies the linear dependence property

$$\alpha_1 = -2, \alpha_2 = 0, \alpha_3 = 0, \alpha_4 = 1.$$

Another set of scalars could be

$$\alpha_1 = 1, \alpha_2 = -1, \alpha_3 = 1, \alpha_4 = 0.$$

Notice that both of these sets of α have at least one scalar that is not equal to zero, $\alpha_i \neq 0$, and that they make the linear combination of the vectors $\vec{v}_1 \dots \vec{v}_4$ add to zero. Therefore, we have shown that the set of vectors, $\vec{v}_1 \dots \vec{v}_4$, are linearly dependent.

Now we want to see whether or not the set of vectors, $\mathbf{A}\vec{v}_1, \mathbf{A}\vec{v}_2, \mathbf{A}\vec{v}_3, \mathbf{A}\vec{v}_4$, are also linearly dependent. Note, the matrix $\mathbf{A}$ can be any 2×2 matrix.

From the proof above, we know that these vectors will be linearly dependent and that the same scalars, $\alpha_1 \dots \alpha_4$, will make the linear combination of the vectors add to zero. But let's see if that happens in our specific example.

Let's choose the matrix $\mathbf{A}$ and show that the set $\alpha_1 = -2, \alpha_2 = 0, \alpha_3 = 0, \alpha_4 = 1$ does in fact give us a linear combination that adds to zero.

$$\mathbf{A} = \begin{bmatrix} -1 & 1 \\ 0 & 1 \end{bmatrix}$$

Given the matrix $\mathbf{A}$, we can then solve for $\mathbf{A}\vec{v}_i$

$$\mathbf{A}\vec{v}_1 = \begin{bmatrix} -1 \\ 0 \end{bmatrix} \mathbf{A}\vec{v}_2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \mathbf{A}\vec{v}_3 = \begin{bmatrix} 2 \\ 1 \end{bmatrix} \mathbf{A}\vec{v}_4 = \begin{bmatrix} -2 \\ 0 \end{bmatrix}$$

We can now test whether the original set of scalars does in fact make the linear combination of these vectors add to zero.

$$\alpha_1 \cdot (\mathbf{A}\vec{v}_1) + \alpha_2 \cdot (\mathbf{A}\vec{v}_2) + \alpha_3 \cdot (\mathbf{A}\vec{v}_3) + \alpha_4 \cdot (\mathbf{A}\vec{v}_4) =$$

$$-2 \cdot \begin{bmatrix} -1 \\ 0 \end{bmatrix} + 0 \cdot \begin{bmatrix} 1 \\ 1 \end{bmatrix} + 0 \cdot \begin{bmatrix} 2 \\ 1 \end{bmatrix} + 1 \cdot \begin{bmatrix} -2 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Great! We have now shown that the set of scalars α that made the linear combination of vectors $\vec{v}_i$ add to zero also made the linear combination of vectors $\mathbf{A}\vec{v}_i$ add to zero. You can change the numbers of the matrix $\mathbf{A}$ and everything still works. A numerical example like this is a great place to start if you are ever struggling with a proof.

Note: There are alternative and equivalent implications of linear dependence that can be used in the proof (instead of Equation 5). Here are a few of them:

(a) Some vector $\vec{v}_j$ can be represented as a linear combination of the *other* vectors as follows: There exist scalars $\alpha_i, 1 \leq i \leq k, i \neq j$, such that

$$\sum_{\substack{i=1 \\ i \neq j}}^{k} \alpha_i \cdot \vec{v}_i = \vec{v}_j.$$

(In this alternative, the scalars may all be zeros, and the linear combination on the left hand side must exclude $\vec{v}_j$.)

(b) There exists $1 \leq j \leq k$ such that $\vec{v}_j$ can be represented as a linear combination of the $\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_{j-1}$ as follows: There exist scalars $\alpha_1, \alpha_2, \ldots, \alpha_{j-1}$, such that

$$\sum_{i=1}^{j-1} \alpha_i \cdot \vec{v}_i = \vec{v}_j.$$

(In this alternative, the scalars may all be zeros, and the linear combination on the left hand side must exclude $\vec{v}_j$.)

For all of these alternatives, the rest of the proof is similar to the one demonstrated above (multiply both sides by $\mathbf{A}$ and then apply the linearity of matrix multiplication to include the $\mathbf{A}$ in the relevant sum and next to the $\vec{v}_i$) and will result in an equation similar to Equation 6 (that matches the chosen alternative).

Common mistakes included:

- When using the definition provided in Equation 5, not indicating that the at least one of scalars needs to be non-zero.
- When using the definition provided in Equation 5, stating that all scalars need to be non-zero.
- When using any of the alternative implications (a)-(b) above, requiring that at least one scalar be non-zero (or all non-zero).
- When using any of the alternative implications (a)-(b) above, not excluding the vector on the right hand side from the linear combination on the left hand side.

7. Image Stitching

Often, when people take pictures of a large object, they are constrained by the field of vision of the camera. This means that they have two options to capture the entire object:

- Stand as far as away as they need to to include the entire object in the camera's field of view (clearly, we do not want to do this as it reduces the amount of detail in the image)
- (This is more exciting) Take several pictures of different parts of the object and stitch them together like a jigsaw puzzle.

We are going to explore the second option in this problem. Daniel, who is a professional photographer, wants to construct an image by using "image stitching". Unfortunately, Daniel took some of the pictures from different angles as well as from different positions and distances from the object. While processing these pictures, Daniel lost information about the positions and orientations from which the pictures were taken. Luckily, you and your friend Marcela, with your wealth of newly acquired knowledge about vectors and matrices, can help him!

You and Marcela are designing an iPhone app that stitches photographs together into one larger image. Marcela has already written an algorithm that finds common points in overlapping images. **It's your job** to figure out how to stitch the images together using Marcela's common points to reconstruct the larger image.

We will use vectors to represent the common points and these related by a linear transformation. Your idea is to find this linear transformation. For this you will use a single matrix, $\mathbf{R}$, and a vector, $\vec{T}$, that transforms every common point in one image to that same point in the other image. Once you find $\mathbf{R}$ and $\vec{T}$ you will be able to transform one image so that it lines up with the other image.

Suppose $\vec{p} = \begin{bmatrix} p_x \\ p_y \end{bmatrix}$ is a point in one image and $\vec{q} = \begin{bmatrix} q_x \\ q_y \end{bmatrix}$ is the corresponding point in the other image (i.e., they represent the same object in the scene). You write down the following relationship between $\vec{p}$ and $\vec{q}$.

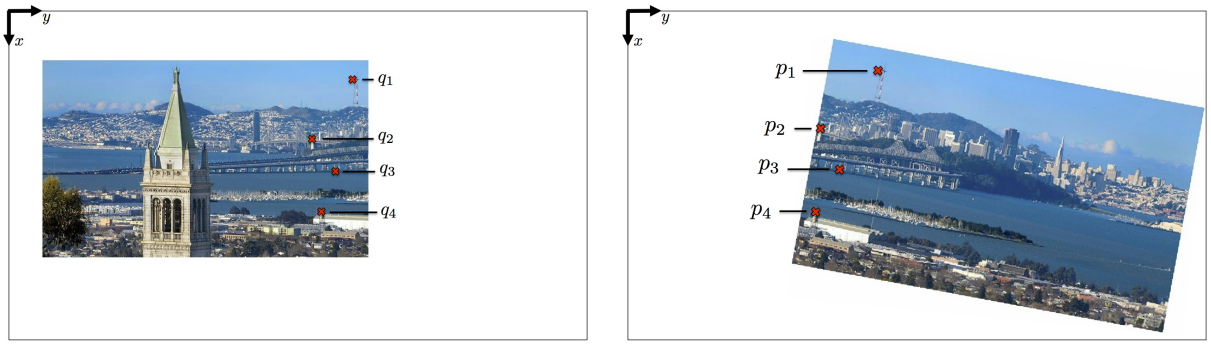


Figure 5: Two images to be stitched together with pairs of matching points labeled.

$$\begin{bmatrix} q_x \\ q_y \end{bmatrix} = \underbrace{\begin{bmatrix} \mathbf{R}_{xx} & \mathbf{R}_{xy} \\ \mathbf{R}_{yx} & \mathbf{R}_{yy} \end{bmatrix}}_{\mathbf{R}} \begin{bmatrix} p_x \\ p_y \end{bmatrix} + \underbrace{\begin{bmatrix} T_x \\ T_y \end{bmatrix}}_{\vec{T}} \tag{7}$$

This problem focuses on finding the unknowns (i.e. the components of $\mathbf{R}$ and $\vec{T}$), so that you will be able to stitch the image together.

(a) To understand how the matrix $\mathbf{R}$ and vector $\vec{T}$ transform a vector, $\vec{v}_0$, consider this similar equation,

$$\vec{v}_2 = \begin{bmatrix} 2 & 2 \\ -2 & 2 \end{bmatrix} \vec{v}_0 + \vec{v}_1. \tag{8}$$

Using $\vec{v}_0 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, $\vec{v}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, what is $\vec{v}_2$? On a single plot, draw the vectors $\vec{v}_0, \vec{v}_1, \vec{v}_2$ in two dimensions. Describe how $\vec{v}_2$ is transformed from $\vec{v}_0$ (e.g. rotated, scaled, shifted).

Solution: Plugging in the given vectors and performing the matrix vector multiplication,

$$\begin{bmatrix} 2 \\ 2 \end{bmatrix} = \begin{bmatrix} 2 & 2 \\ -2 & 2 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix}. \tag{9}$$

We get $\vec{v}_2 = \begin{bmatrix} 3 \\ 3 \end{bmatrix}$.

Looking at the plotted vectors, it is observable that $\vec{v}_2$ is a rotated and translated version of $\vec{v}_0$.

(b) Multiply Equation (7) out into two scalar linear equations. What are the known values and what are the unknowns in each equation? How many unknowns are there? How many independent equations do you need to solve for all the unknowns? How many pairs of common points $\vec{p}$ and $\vec{q}$ will you need in order to write down a system of equations that you can use to solve for the unknowns?

Solution:

We can rewrite the above matrix equation as the following two scalar linear equations:

$$q_x = p_x R_{xx} + p_y R_{xy} + T_x$$
$$q_y = p_x R_{yx} + p_y R_{yy} + T_y$$

Here, the known values are the elements of the pair of points: q_x, q_y, p_x, p_y, and 1. The unknowns are elements of $\mathbf{R}$ and $\vec{T}$: R_{xx}, R_{xy}, R_{yx}, R_{yy}, T_x, and T_y. There are 6 unknowns, so we need a total of 6 equations to solve for them. For every pair of points we add, we get two more equations. Thus, we need 3 pairs of common points to get 6 equations.

(c) What is the vector of unknown values? Write out a system of linear equations that you can use to solve for the unknown values (you should use multiple pairs of points $\vec{p}$'s and $\vec{q}$'s to have enough equations). Transform these linear equations to in matrix equation, so that we could solve for the vector of unknown values.

Solution:

The vector of unknowns:

$$\begin{bmatrix} R_{xx} \\ R_{xy} \\ R_{yx} \\ R_{yy} \\ T_x \\ T_y \end{bmatrix} \tag{10}$$

The system of linear equations:

$$R_{xx}p_{1x} + R_{xy}p_{1y} + T_x = q_{1x} \tag{11}$$
$$R_{yx}p_{1x} + R_{yy}p_{1y} + T_y = q_{1y} \tag{12}$$
$$R_{xx}p_{2x} + R_{xy}p_{2y} + T_x = q_{2x} \tag{13}$$
$$R_{yx}p_{2x} + R_{yy}p_{2y} + T_y = q_{2y} \tag{14}$$
$$R_{xx}p_{3x} + R_{xy}p_{3y} + T_x = q_{3x} \tag{15}$$
$$R_{yx}p_{3x} + R_{yy}p_{3y} + T_y = q_{3y} \tag{16}$$
$$\tag{17}$$

We will label the 3 pairs of point we select as:

$$\vec{q}_1 = \begin{bmatrix} q_{1x} \\ q_{1y} \end{bmatrix}, \quad \vec{p}_1 = \begin{bmatrix} p_{1x} \\ p_{1y} \end{bmatrix} \qquad \vec{q}_2 = \begin{bmatrix} q_{2x} \\ q_{2y} \end{bmatrix}, \quad \vec{p}_2 = \begin{bmatrix} p_{2x} \\ p_{2y} \end{bmatrix} \qquad \vec{q}_3 = \begin{bmatrix} q_{3x} \\ q_{3y} \end{bmatrix}, \quad \vec{p}_3 = \begin{bmatrix} p_{3x} \\ p_{3y} \end{bmatrix}$$

We write the system of linear equations in matrix form.

$$\begin{bmatrix} p_{1x} & p_{1y} & 0 & 0 & 1 & 0 \\ 0 & 0 & p_{1x} & p_{1y} & 0 & 1 \\ p_{2x} & p_{2y} & 0 & 0 & 1 & 0 \\ 0 & 0 & p_{2x} & p_{2y} & 0 & 1 \\ p_{3x} & p_{3y} & 0 & 0 & 1 & 0 \\ 0 & 0 & p_{3x} & p_{3y} & 0 & 1 \end{bmatrix} \begin{bmatrix} R_{xx} \\ R_{xy} \\ R_{yx} \\ R_{yy} \\ T_x \\ T_y \end{bmatrix} = \begin{bmatrix} q_{1x} \\ q_{1y} \\ q_{2x} \\ q_{2y} \\ q_{3x} \\ q_{3y} \end{bmatrix}$$

(d) In the IPython notebook `prob2.ipynb`, you will have a chance to test out your solution. Plug in the values that you are given for p_x, p_y, q_x, and q_y for each pair of points into your system of equations to solve for the matrix, $\mathbf{R}$, and vector, $\vec{T}$. The notebook will solve the system of equations, apply your transformation to the second image, and show you if your stitching algorithm works. You are not responsible for understanding the image stitching code or Marcela's algorithm.

Solution:

The parameters for the transformation from the coordinates of the first image to those of the second image are $\mathbf{R} = \begin{bmatrix} 1.1954 & .1046 \\ -.1046 & 1.1954 \end{bmatrix}$ and $\vec{T} = \begin{bmatrix} -150 \\ -250 \end{bmatrix}$.

(e) We will now explore when this algorithm fails. Marcela's algorithm gives us a new set of corresponding points between the images (Figure 6), but the new points are collinear (i.e. we can draw a straight line between them in the image). Marcela suspects that you will run into an issue when solving for your unknown vector. In the IPython notebook `prob2.ipynb`, try to plug in the new corresponding points and solve for new unknown values. What happens?

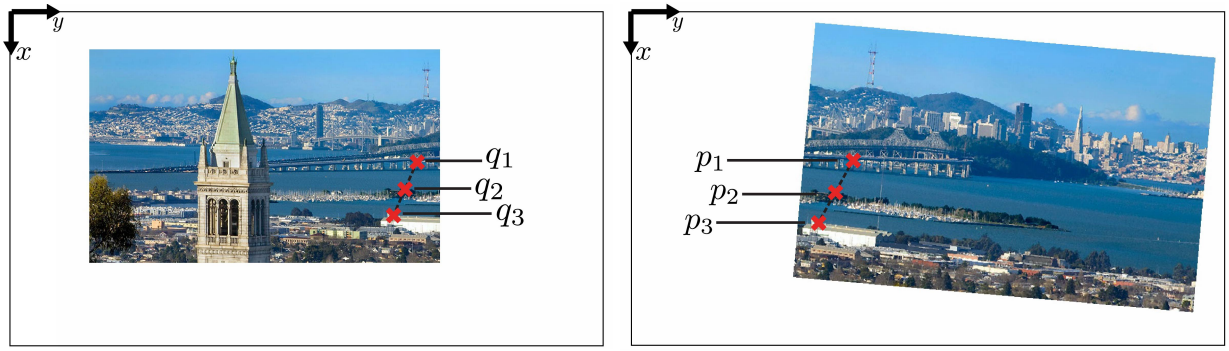


Figure 6: Two images to be stitched together with collinear pairs of matching points labeled.

Solution: Once you plug in the new collinear points, the IPython Notebook will return an error. The error message is:

Could not invert matrix. Rows are not linearly independent. :)

You will learn more about matrix inversion in the coming weeks.

If you did not get the error but expected to get an error and explained why, then this is okay and you can receive full credit. Depending the version of numpy, you may or may not have gotten the error.

(f) Now more generally, show that if $\vec{p}_1, \vec{p}_2, \vec{p}_3$ are *collinear*, the system of equations you created in part (c) will have an infinite number of solutions.

Fact: $\vec{p}_1, \vec{p}_2, \vec{p}_3$ are collinear *if and only if* $(\vec{p}_2 - \vec{p}_1) = k(\vec{p}_3 - \vec{p}_1)$ for some $k \in \mathbb{R}$.

Solution:

Note: A general 2D affine transformation is a transformation from points $\vec{p}$ to points $\vec{q}$ of the form of Equation 7. It is a linear transformation (given by matrix $\mathbf{R}$) followed by a translation given by the vector $\vec{T}$.

The algorithm fails when the points $\vec{p}_1, \vec{p}_2, \vec{p}_3$ are not distinct or when they are collinear.

To show that the collinear case leads to an underdetermined system, write the system (with respect to the matrix $\mathbf{R}$) as:

$$\begin{cases} \vec{q}_1 = \mathbf{R}\vec{p}_1 + \vec{T} \\ \vec{q}_2 = \mathbf{R}\vec{p}_2 + \vec{T} \\ \vec{q}_3 = \mathbf{R}\vec{p}_3 + \vec{T} \end{cases}$$

By subtracting the first equation from the last two, this is equivalent to the following system:

$$\begin{cases} \vec{q}_1 = \mathbf{R}\vec{p}_1 + \vec{T} \\ \vec{q}_2 - \vec{q}_1 = \mathbf{R}(\vec{p}_2 - \vec{p}_1) \\ \vec{q}_3 - \vec{q}_1 = \mathbf{R}(\vec{p}_3 - \vec{p}_1) \end{cases}$$

Then, by collinearity, this is equivalent to the system:

$$\begin{cases} \vec{q}_1 = \mathbf{R}\vec{p}_1 + \vec{T} \\ \vec{q}_2 - \vec{q}_1 = \mathbf{R}k(\vec{p}_3 - \vec{p}_1) \\ \vec{q}_3 - \vec{q}_1 = \mathbf{R}(\vec{p}_3 - \vec{p}_1) \end{cases}$$

Note that the last two equations are now linearly dependent. To see this, remember that the variables are the entries of the matrix $\mathbf{R} = \begin{bmatrix} R_{xx} & R_{xy} \\ R_{yx} & R_{yy} \end{bmatrix}$, and all other elements $(\vec{p}_i, \vec{q}_i, k)$ are constants. Define $\vec{p}_{31} = \vec{p}_3 - \vec{p}_1$. The third equation is a constraint on linear combinations of entries in $\mathbf{R}$ – specifically, it is a constraint on $\mathbf{R}\vec{p}_{31}$. The second equation is a constraint on $\mathbf{R}(k\vec{p}_{31})$. However, since $\mathbf{R}(k\vec{p}_{31}) = k(\mathbf{R}\vec{p}_{31})$, these two constraints are linearly dependent. Therefore the system is underdetermined.

This problem can also be solved by writing out the linear equations explicitly in matrix form, as you did in part (b). The system becomes:

$$\underbrace{\begin{bmatrix} p_{1x} & p_{1y} & 0 & 0 & 1 & 0 \\ 0 & 0 & p_{1x} & p_{1y} & 0 & 1 \\ p_{2x} & p_{2y} & 0 & 0 & 1 & 0 \\ 0 & 0 & p_{2x} & p_{2y} & 0 & 1 \\ p_{3x} & p_{3y} & 0 & 0 & 1 & 0 \\ 0 & 0 & p_{3x} & p_{3y} & 0 & 1 \end{bmatrix}}_{\mathbf{A}} \begin{bmatrix} R_{xx} \\ R_{xy} \\ R_{yx} \\ R_{yy} \\ T_x \\ T_y \end{bmatrix} = \begin{bmatrix} q_{1x} \\ q_{1y} \\ q_{2x} \\ q_{2y} \\ q_{3x} \\ q_{3y} \end{bmatrix}$$

Subtracting row 1 from rows 4 and 5 and then subtracting row 2 from rows 4 and 6, the matrix $\mathbf{A}$ becomes:

$$\begin{bmatrix} p_{1x} & p_{1y} & 0 & 0 & 1 & 0 \\ 0 & 0 & p_{1x} & p_{1y} & 0 & 1 \\ p_{2x} - p_{1x} & p_{2y} - p_{1y} & 0 & 0 & 0 & 0 \\ 0 & 0 & p_{2x} - p_{1x} & p_{2y} - p_{1y} & 0 & 0 \\ p_{3x} - p_{1x} & p_{3y} - p_{1y} & 0 & 0 & 0 & 0 \\ 0 & 0 & p_{3x} - p_{1x} & p_{3y} - p_{1y} & 0 & 0 \end{bmatrix}$$

Then, by collinearity, we have $p_{2x} - p_{1x} = k(p_{3x} - p_{1x})$ and so on, so the above matrix is:

$$\begin{bmatrix} p_{1x} & p_{1y} & 0 & 0 & 1 & 0 \\ 0 & 0 & p_{1x} & p_{1y} & 0 & 1 \\ k(p_{3x} - p_{1x}) & k(p_{3y} - p_{1y}) & 0 & 0 & 0 & 0 \\ 0 & 0 & k(p_{3x} - p_{1x}) & k(p_{3y} - p_{1y}) & 0 & 0 \\ p_{3x} - p_{1x} & p_{3y} - p_{1y} & 0 & 0 & 0 & 0 \\ 0 & 0 & p_{3x} - p_{1x} & p_{3y} - p_{1y} & 0 & 0 \end{bmatrix}$$

Now, clearly rows 3 and 5 are linearly dependent (and, in fact, so are rows 4 and 6). Therefore the linear system is underdetermined.

Geometrically: Consider a transformation that simply scales a vector along the x-axis. For example, $\vec{T} = 0$ and $\mathbf{R} = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$. Say we pick all $\vec{p}_i$ to lie along the y-axis. Then, applying this transformation will give $\vec{q}_i = \vec{p}_i$ (since the y-axis is unaffected by this scaling). However, only given q_i, this transformation has the same effect as the identity transformation (that is, where $\vec{T} = \vec{0}$ and $\mathbf{R}$ is the identity). Therefore, the transformation cannot be determined from the q_i (both the identity transformation and the x-axis scaling lead to the same q_i). This argument can be extended to the case when all p_i are collinear but not necessarily along the x-axis.

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.