EECS 16A Spring 2020 — Designing Information Devices and Systems I — Homework 2

This homework is due Friday, February 7, 2020, at 23:59.
Self-grades are due Monday, February 10, 2020, at 23:59.

Submission Format

Your homework submission should consist of a single PDF file that contains all of your answers (any hand-written answers should be scanned) as well as your IPython notebook saved as a PDF.

Please attach a PDF of your Jupyter notebook for all the problems that involve coding. Make sure the results of your plots (if any) are visible. Please assign the PDF of the notebook to the correct problems on Gradescope — we will be unable to grade the problems without this assignment or submission.

1. Mechanical Gaussian Elimination and Linear Independence

Consider the following system of linear equations:

$$\begin{aligned} 4x+4y+4z+w+v &= 1 \\ x+y+2z+4w+v &= 2 \\ 5x+5y+5z+w+v &= 0 \end{aligned}$$

(a) Find the RREF of the augmented matrix of the system of linear equations.

Solution: There are a variety of ways to row reduce to reach RREF. Here is a very standard way, eliminating downwards to reach REF, then performing backsubstitution:

$$\left[\begin{array}{ccccc|c} 4 & 4 & 4 & 1 & 1 & 1 \\ 1 & 1 & 2 & 4 & 1 & 2 \\ 5 & 5 & 5 & 1 & 1 & 0 \end{array}\right] \xrightarrow{R_1 \leftarrow R_1/4} \left[\begin{array}{ccccc|c} 1 & 1 & 1 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ 1 & 1 & 2 & 4 & 1 & 2 \\ 5 & 5 & 5 & 1 & 1 & 0 \end{array}\right] \xrightarrow[R_3 \leftarrow R_3 - 5R_1]{R_2 \leftarrow R_2 - R_1} \left[\begin{array}{ccccc|c} 1 & 1 & 1 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ 0 & 0 & 1 & \frac{15}{4} & \frac{3}{4} & \frac{7}{4} \\ 0 & 0 & 0 & -\frac{1}{4} & -\frac{1}{4} & -\frac{5}{4} \end{array}\right] \xrightarrow{R_3 \leftarrow -4R_3}$$

$$\left[\begin{array}{ccccc|c} 1 & 1 & 1 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ 0 & 0 & 1 & \frac{15}{4} & \frac{3}{4} & \frac{7}{4} \\ 0 & 0 & 0 & 1 & 1 & 5 \end{array}\right] \xrightarrow[R_1 \leftarrow R_1 - \frac{1}{4}R_3]{R_2 \leftarrow R_2 - \frac{15}{4}R_3} \left[\begin{array}{ccccc|c} 1 & 1 & 1 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 & -3 & -17 \\ 0 & 0 & 0 & 1 & 1 & 5 \end{array}\right] \xrightarrow{R_1 \leftarrow R_1 - R_2} \left[\begin{array}{ccccc|c} 1 & 1 & 0 & 0 & 3 & 16 \\ 0 & 0 & 1 & 0 & -3 & -17 \\ 0 & 0 & 0 & 1 & 1 & 5 \end{array}\right]$$

(b) Which variables are the basic variables? Which variables are the free variables?

Solution: x, z, and w are the basic variables as they have leading coefficients in their column. y and v are the free variables.

(c) Parameterize the solutions to the system of equations in terms of the free variables.

Solution:

We can solve for x, y, z, w, and v in terms of the free variables y and v.

$y = y$
$v = v$
$w = 5 - v$
$z = -17 + 3v$
$x = 16 - 3v - y$

The solutions to the system of equations is the above, for a particular choice of the free variables y and v.

The set of solutions can also be written with vectors: $\left\{ \begin{bmatrix} 16 \\ 0 \\ -17 \\ 5 \\ 0 \end{bmatrix} + \begin{bmatrix} -1 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} y + \begin{bmatrix} -3 \\ 0 \\ 3 \\ -1 \\ 1 \end{bmatrix} v \in \mathbb{R}^5 \middle| y, v \in \mathbb{R} \right\}$

(d) State if the following sets of vectors are linearly independent or dependent. If the set is linearly dependent, provide a linear combination of the vectors that sum to the zero vector.

i $\left\{ \begin{bmatrix} -5 \\ 2 \end{bmatrix}, \begin{bmatrix} 5 \\ 2 \end{bmatrix} \right\}$

Solution: The vectors are linearly independent. A set of two vectors can only be linearly dependent if one of the vectors is a scaled version of the other. $\begin{bmatrix} -5 \\ 2 \end{bmatrix} \neq \alpha \begin{bmatrix} 5 \\ 2 \end{bmatrix}$ no matter the chosen $\alpha \in \mathbb{R}$.

ii $\left\{ \begin{bmatrix} -1 \\ 5 \\ 0 \\ -3 \\ \frac{1}{2} \end{bmatrix}, \begin{bmatrix} -7 \\ 20 \\ 24 \\ -12 \\ 0 \end{bmatrix}, \begin{bmatrix} \frac{1}{2} \\ 0 \\ -4 \\ 0 \\ \frac{1}{3} \end{bmatrix} \right\}$

Solution: This set of vectors is linearly dependent. To find a linear combination that yields $\vec{0}$, find the RREF of the following augmented matrix:

$$\left[\begin{array}{ccc|c} -1 & -7 & \frac{1}{2} & 0 \\ 5 & 20 & 0 & 0 \\ 0 & 24 & -4 & 0 \\ -3 & -12 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{3} & 0 \end{array}\right] \xrightarrow{R_1 \leftarrow -R_1} \left[\begin{array}{ccc|c} 1 & 7 & -\frac{1}{2} & 0 \\ 5 & 20 & 0 & 0 \\ 0 & 24 & -4 & 0 \\ -3 & -12 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{3} & 0 \end{array}\right] \xrightarrow[R_4 \leftarrow R_4 + 3R_1]{R_2 \leftarrow R_2 - 5R_1} \left[\begin{array}{ccc|c} 1 & 7 & -\frac{1}{2} & 0 \\ 0 & -15 & \frac{5}{2} & 0 \\ 0 & 24 & -4 & 0 \\ 0 & 9 & -\frac{3}{2} & 0 \\ \frac{1}{2} & 0 & \frac{1}{3} & 0 \end{array}\right] \xrightarrow[R_2 \leftarrow R_2/-15]{R_5 \leftarrow R_5 - \frac{1}{2}R_1}$$

$$\left[\begin{array}{ccc|c} 1 & 7 & -\frac{1}{2} & 0 \\ 0 & 1 & -\frac{1}{6} & 0 \\ 0 & 24 & -4 & 0 \\ 0 & 9 & -\frac{3}{2} & 0 \\ 0 & -\frac{7}{2} & \frac{7}{12} & 0 \end{array}\right] \xrightarrow[R_4 \leftarrow R_4 - 9R_2]{R_3 \leftarrow R_3 - 24R_2} \left[\begin{array}{ccc|c} 1 & 7 & -\frac{1}{2} & 0 \\ 0 & 1 & -\frac{1}{6} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & -\frac{7}{2} & \frac{7}{12} & 0 \end{array}\right] \xrightarrow[R_1 \leftarrow R_1 - 7R_2]{R_5 \leftarrow R_5 + \frac{7}{2}R_2} \left[\begin{array}{ccc|c} 1 & 0 & \frac{2}{3} & 0 \\ 0 & 1 & -\frac{1}{6} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{array}\right]$$

Choosing the value of the free variable as 6α (chosen to make the coefficients nice) gives the following linear combination that shows linear dependence:

$$-4\alpha \begin{bmatrix} -1 \\ 5 \\ 0 \\ -3 \\ \frac{1}{2} \end{bmatrix} + \alpha \begin{bmatrix} -7 \\ 20 \\ 24 \\ -12 \\ 0 \end{bmatrix} + 6\alpha \begin{bmatrix} \frac{1}{2} \\ 0 \\ -4 \\ 0 \\ \frac{1}{3} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

If you have a set of coefficients that match a specific value of α, give yourself full credit.

iii $\left\{ \begin{bmatrix} 2 \\ 2 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 2 \\ 4 \\ -1 \end{bmatrix}, \begin{bmatrix} 0 \\ -1 \\ 1 \end{bmatrix} \right\}$

Solution: This set of vectors is linearly dependent. Since the subset of the first 3 vectors are linearly independent, they span $\mathbb{R}^3$. Since the fourth vector is in $\mathbb{R}^3$, we are guaranteed some linear combination of the first three vectors that yields the fourth. To find a specific linear combination that shows linear dependence find the RREF of the following augmented matrix:

$$\left[\begin{array}{cccc|c} 2 & 0 & 2 & 0 & 0 \\ 2 & 1 & 4 & -1 & 0 \\ 0 & 1 & -1 & 1 & 0 \end{array}\right] \xrightarrow{R_1 \leftarrow R_1/2} \left[\begin{array}{cccc|c} 1 & 0 & 1 & 0 & 0 \\ 2 & 1 & 4 & -1 & 0 \\ 0 & 1 & -1 & 1 & 0 \end{array}\right] \xrightarrow{R_2 \leftarrow R_2 - 2R_1} \left[\begin{array}{cccc|c} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & -1 & 0 \\ 0 & 1 & -1 & 1 & 0 \end{array}\right] \xrightarrow{R_3 \leftarrow R_3 - R_2}$$
$$\left[\begin{array}{cccc|c} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & -1 & 0 \\ 0 & 0 & -3 & 2 & 0 \end{array}\right] \xrightarrow{R_3 \leftarrow R_3/-3} \left[\begin{array}{cccc|c} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & -1 & 0 \\ 0 & 0 & 1 & -\frac{2}{3} & 0 \end{array}\right] \xrightarrow[R_1 \leftarrow R_1 - R_3]{R_2 \leftarrow R_2 - 2R_3} \left[\begin{array}{cccc|c} 1 & 0 & 0 & \frac{2}{3} & 0 \\ 0 & 1 & 0 & \frac{1}{3} & 0 \\ 0 & 0 & 1 & -\frac{2}{3} & 0 \end{array}\right]$$

Choosing the value of the free variable as 3α (chosen to make the coefficients nice) gives the following linear combination that shows linear dependence:

$$-2\alpha \begin{bmatrix} 2 \\ 2 \\ 0 \end{bmatrix} - \alpha \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} + 2\alpha \begin{bmatrix} 2 \\ 4 \\ -1 \end{bmatrix} + 3\alpha \begin{bmatrix} 0 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

If you have a set of coefficients that match a specific value of α, give yourself full credit.

iv $\left\{ \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ -2 \\ 2 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \right\}$

Solution: Since this set contains $\vec{0}$, it is linearly dependent as we can take the linear combination $\alpha \cdot \vec{0}$ where $\alpha \neq 0$ to get $\vec{0}$.

2. 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.

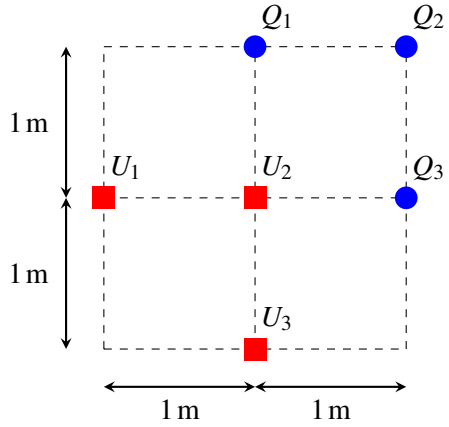


Figure 1: Locations of the charges and potentials.

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

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

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:

$$r_{1,1} = \sqrt{2}, \quad r_{1,2} = \sqrt{5}, \quad r_{1,3} = 2,$$
$$r_{2,1} = 1, \quad r_{2,2} = \sqrt{2}, \quad r_{2,3} = 1,$$
$$r_{3,1} = 2, \quad r_{3,2} = \sqrt{5}, \quad r_{3,3} = \sqrt{2}.$$

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

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

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

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

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. Kinematic Model for a Simple Car

Learning Goal: Many real world systems are not actually linear and have more complex behaviors. However, linear models can approximate non linear systems under certain conditions.

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.

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

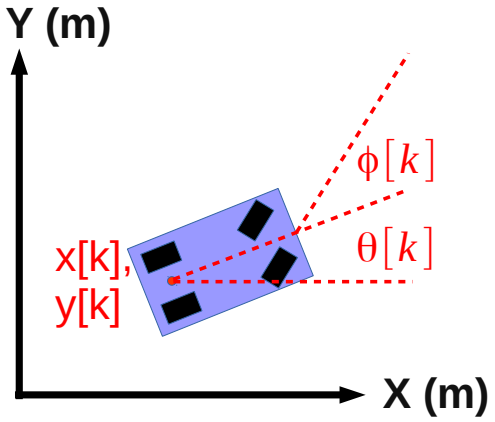


Figure 2: Vehicle Kinematic Model

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:

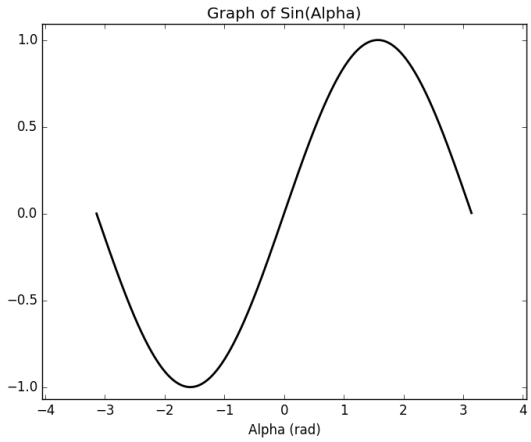


Figure 3: Graph of $sin(\alpha)$

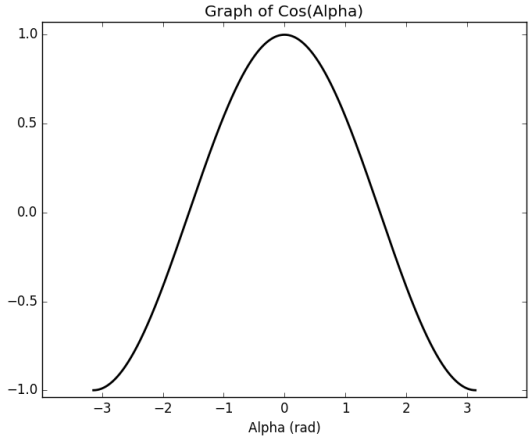


Figure 4: Graph of $cos(\alpha)$

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.

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

Hint: *Start with simplifying Equations* (1) *to* (4).

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.

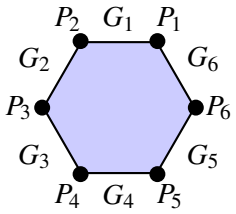
4. Figuring Out The Tips

Learning Objective: *This problem showcases how you can understand general systems of equations by looking at simpler examples. In particular, see if you can generalize your intuition from the case of 5 and 6 guests to a general number of guests.*

A number of guests gather around a round table for a dinner. Between every adjacent pair of guests, 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. Suppose you can only see the amount of tips in each plate after everyone has left. Can you deduce the amount that each individual tipped?

Note: For this question, if we assume that tips are positive, then we need to introduce additional constraint that would make the system of equations no longer linear. We are going to ignore this constraint and assume that negative tips are acceptable.

(a) Suppose six guests sit around a table and there are six plates of tips. If we know the amount of tip in each plate, P_1 to P_6, can we determine each individual's tip amount, G_1 to G_6? If yes, explain why by examining the relationship between the plate values, P_1 to P_6, and guest tips, G_1 to G_6. If not, give two different assignments of G_1 to G_6 that will result in the same P_1 to P_6.



Solution:

No, this is not possible. There exists multiple solutions for G_1 to G_6 given P_1 to P_6. For example, the following two different assignments of tip amounts for each guest:

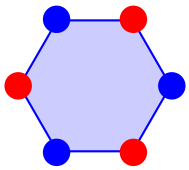
$$(G_1, G_2, G_3, G_4, G_5, G_6) = (2, 0, 2, 0, 2, 0)$$
$$(G_1, G_2, G_3, G_4, G_5, G_6) = (0, 2, 0, 2, 0, 2)$$

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

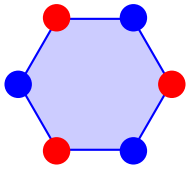
If we write down the system of linear equations:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} G_1 \\ G_2 \\ G_3 \\ G_4 \\ G_5 \\ G_6 \end{bmatrix} = \begin{bmatrix} 2P_1 \\ 2P_2 \\ 2P_3 \\ 2P_4 \\ 2P_5 \\ 2P_6 \end{bmatrix}$$

We can use Gaussian elimination to reduce the last row to all zeros. Therefore, equations are linearly dependent. Intuitively, we can color each spot on the table alternating between red or blue:

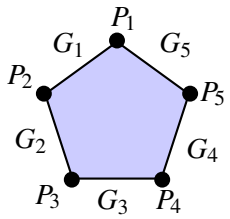


Then supposing that everyone sitting at red spots all tip r dollars and everyone sitting at blue spots all tip b dollars, we find $P_1, \cdots, P_6$ dollars on the plates. However, this is no different from this following coloring:



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

(b) Now lets consider five guests at the table, G_1 to G_5, and we can see the amount of tips in the five plates, P_1 to P_5. In this new setting can you figure out each guests tip values, G_1 to G_5?



Solution:

Yes. Our solution will use Gaussian elimination to conclude that the system of equations is linearly independent and a single solution exists. We start by formulating the system of equations as a matrix:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} G_1 \\ G_2 \\ G_3 \\ G_4 \\ G_5 \end{bmatrix} = \begin{bmatrix} 2P_1 \\ 2P_2 \\ 2P_3 \\ 2P_4 \\ 2P_5 \end{bmatrix}$$

We can run Gaussian elimination on this matrix. In the following order apply the Gaussian elimination operations:

i. subtract row 1 from row 2
ii. subtract row 2 from row 3
iii. subtract row 3 from row 4
iv. subtract row 4 from row 5
v. divide row 5 by a factor of 2

The resulting matrix is:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} G_1 \\ G_2 \\ G_3 \\ G_4 \\ G_5 \end{bmatrix} = \begin{bmatrix} 2P_1 \\ 2P_2 - 2P_1 \\ 2P_3 - 2P_2 + 2P_1 \\ 2P_4 - 2P_3 + 2P_2 - 2P_1 \\ P_5 - P_4 + P_3 - P_2 + P_1 \end{bmatrix} \tag{5}$$

Then in any order apply the Gaussian elimination operations:

i. add row 5 to row 4
ii. subtract row 5 from row 3
iii. add row 5 to row 2
iv. subtract row 5 from row 1

The results is:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} G_1 \\ G_2 \\ G_3 \\ G_4 \\ G_5 \end{bmatrix} = \begin{bmatrix} -P_5 + P_4 - P_3 + P_2 + P_1 \\ P_5 - P_4 + P_3 + P_2 - P_1 \\ -P_5 + P_4 + P_3 - P_2 + P_1 \\ P_5 + P_4 - P_3 + P_2 - P_1 \\ P_5 - P_4 + P_3 - P_2 + P_1 \end{bmatrix} \tag{6}$$

Because we have a full set of pivots columns, a single unique solution exists for G_1 to G_5 in terms of P_1 to P_5.

(c) If n is the total number of guests sitting around a table, for which values of n can you figure out everyone's tip? You do not have to rigorously prove your answer. (**Hint:** consider what is different about parts a and b.)

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 two different ways that this could have happened.

First:

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

Second:

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

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

For odd n, we can determine everyone's tip. Your solution can either argue with Gaussian 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:

$$\begin{bmatrix} 1 & 0 & \cdots & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 1 & 0 & \cdots & 0 \\ & & \ddots & \ddots & & \\ 0 & 0 & \cdots & 1 & 1 & 0 \\ 0 & 0 & \cdots & 0 & 1 & 1 \end{bmatrix} \begin{matrix} \text{Row } 1 \\ \text{Row } 2 \\ \vdots \\ \text{Row } n-2 \\ \text{Row } n-1 \\ \text{Row } n \end{matrix}$$

We want to perform Gaussian elimination on this matrix. First, we subtract each row from the one below it until row $n-1$. That is, we sequentially subtract row 1 from row 2, then row 2 from row 3, and so on and so forth until we subtract row $n-1$ from row n.

Let's see what the matrix looks like after the 1^{st} subtraction ($R_2 - R_1$):

$$\begin{bmatrix} 1 & 0 & \cdots & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & \cdots & -1 \\ 0 & 1 & 1 & 0 & \cdots & 0 \\ & & \ddots & \ddots & & \\ 0 & 0 & \cdots & 1 & 1 & 0 \\ 0 & 0 & \cdots & 0 & 1 & 1 \end{bmatrix} \begin{matrix} \text{Row } 1 \\ \text{Row } 2 \\ \vdots \\ \text{Row } n-2 \\ \text{Row } n-1 \\ \text{Row } n \end{matrix}$$

Then after the 2^{nd} subtraction ($R_3 - R_2$):

$$\begin{bmatrix} 1 & 0 & \cdots & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & \cdots & -1 \\ 0 & 0 & 1 & 0 & \cdots & 1 \\ & & \ddots & \ddots & & \\ 0 & 0 & \cdots & 1 & 1 & 0 \\ 0 & 0 & \cdots & 0 & 1 & 1 \end{bmatrix} \begin{matrix} \text{Row } 1 \\ \text{Row } 2 \\ \vdots \\ \text{Row } n-2 \\ \text{Row } n-1 \\ \text{Row } n \end{matrix}$$

And after the $n-2$ subtraction ($R_{n-1} - R_{n-2}$):

$$\begin{bmatrix} 1 & 0 & \cdots & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & \cdots & -1 \\ 0 & 0 & 1 & 0 & \cdots & 1 \\ & & \ddots & \ddots & & \\ 0 & 0 & \cdots & 0 & 1 & -1 \\ 0 & 0 & \cdots & 0 & 1 & 1 \end{bmatrix} \begin{matrix} \text{Row } 1 \\ \text{Row } 2 \\ \vdots \\ \text{Row } n-2 \\ \text{Row } n-1 \\ \text{Row } n \end{matrix}$$

Can you see the pattern this sequence of subtractions generates? It creates all 1's in the diagonal, alternating 1's and -1's in n^{th} column, and 0's everywhere else. The main thing to notice is that we have -1's on the even rows of the n^{th} column and 1's in the odd rows of the n^{th} column. Since n is odd it follows that $n-1$ will be even and hence we have a -1 in the $a_{n-1,n}$ element of the row reduced matrix.

Finally, we are ready to perform the last subtraction ($R_{n-1} - R_{n-2}$):

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

We have reached an upper triangular form, which means that the system has a unique solution! To find it, we divide row n by a factor of 2. Lastly, we add row n back to all even rows and subtract it from all odd rows, thus performing the back-substituting step of the Gaussian elimination algorithm.

Alternate solution:

Suppose that each customer tipping: $G_1 = a_1, \cdots, G_n = a_n$ gives rise to the amount in plates $P_1 = p_1, \cdots, P_n = p_n$. This means that we have a solution that looks like $[G_1, G_2, \cdots, G_n] = [a_1, a_2, \cdots, a_n]$, and produces the following amount of tips in the plates: $[P_1, P_2, \cdots, P_n] = [p_1, p_2, \cdots, p_n]$. Now suppose that there exists a different solution, that gives rise to exactly the same amount in each and every plate, that is we still have $[P_1, P_2, \cdots, P_n] = [p_1, p_2, \cdots, p_n]$, but at least one of the tips that a guest has given is different, so at least one of a_i's will be different. Without loss of generality we can assume that this "new" solution will differ for the amount the first guest tips. Also without loss in generality we can choose that $G_1' = a_1 + \varepsilon$. Then, writing out again the second equation of the system we will have:

$$\begin{aligned} G_1' + G_2' &= 2p_2 \\ G_2' &= 2p_2 - G_1' \\ &= (a_1 + a_2) - (a_1 + \varepsilon) \\ &= a_2 - \varepsilon \end{aligned}$$

Imagine now, that we keep writing out our equations one-by-one for every plate. What will happen to the individual tips of all guests? Under the assumption that the amount of tips $[p_1, p_2, \cdots, p_n]$ is kept the same on each an every plate, the individual tips of all the guests will change to become $G_i' = a_i - \varepsilon$ for i even, and $G_i' = a_i + \varepsilon$ for i odd. So, we will have $G_n' = a_n + \varepsilon$ for n odd. In that case the first equation in our system becomes:

$$\begin{aligned} G_1' + G_n' &= 2p_1 \\ G_1' &= 2p_1 - G_n' \\ &= (a_1 + a_n) - (a_n + \varepsilon) \\ &= a_1 - \varepsilon \end{aligned}$$

Which contradicts our initial assumption of $G_1' = a_1 + \varepsilon$, if $\varepsilon \neq 0$!! This means, that ε must be zero and we indeed have a unique solution when n is odd.

5. Proof: Linear Dependence in a Square Matrix

(a) Let $\mathbf{A}$ be a $n \times n$ matrix, (i.e. both the columns and rows are vectors in R^n). Suppose we are told that the columns of $\mathbf{A}$ are linearly dependent. Prove, then, that the rows of $\mathbf{A}$ must also be linearly dependent.

(**Hint**: A set of vectors $\{\vec{a}_1,...,\vec{a}_n\}$ is linearly dependent if there exist scalars α_1,...,α_n such that $\alpha_1\vec{a}_1$+...+$\alpha_n\vec{a}_n$=$\vec{0}$ and not all α_i's are equal to zero.)

Solution: There are multiple approaches to this problem. We cover one possible approach here.

First, we express $\mathbf{A}$ in terms of its rows and columns. Let $\{A_1, A_2, \ldots, A_n\}$ represent the columns of $\mathbf{A}$. Similarly, let $\{a_1, a_2, \ldots, a_n\}$ represent the rows of $\mathbf{A}$.

$$\mathbf{A} = \begin{bmatrix} A_1 & A_2 & \ldots & A_n \end{bmatrix} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}$$

We are given that the columns of $\mathbf{A}$ are linearly dependent. This means that there are scalars $\alpha_1, \ldots, \alpha_n$ in $\mathbb{R}$, *of which at least one is nonzero*, such that $\alpha_1 * A_1 + \ldots \alpha_n * A_n = 0$. Let

$$\alpha = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{bmatrix}$$

We can express this fact through matrix multiplication (where $\mathbf{0}$ is the zero vector in $\mathbb{R}^n$):

$$\begin{bmatrix} A_1 & A_2 & \ldots & A_n \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{bmatrix} = \mathbf{0}$$

or equivalently, $\mathbf{A}\alpha = \mathbf{0}$. Recall that $\alpha \neq \mathbf{0}$. This means we can scale the vector α by any constant and still get a different solution to the problem $\mathbf{A}x = \mathbf{0}$.

This implies that there are infinitely many solutions to the problem $\mathbf{A}x = \mathbf{0}$, which we know corresponds to a row of zeros in the reduced form of $\mathbf{A}$ after doing Gaussian elimination.

The presence of a row of zeros in the reduced form suggests that at least one of the rows must be a linear combination of the others and thus the rows are also linearly dependent.

(b) Given some set of vectors $\{\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_n\}$, show the following:

$$\text{span}\{\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_n\} = \text{span}\{\vec{v}_1 + \vec{v}_2, \vec{v}_2, \ldots, \vec{v}_n\}$$

In other words, we can replace one vector with the sum of itself and another vector and not change their span.

Solution:

Suppose $\vec{q} \in \text{span}\{\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_n\}$. For some scalars a_i:

$$\vec{q} = a_1\vec{v}_1 + a_2\vec{v}_2 + \cdots + a_n\vec{v}_n = a_1(\vec{v}_1 + \vec{v}_2) + (-a_1 + a_2)\vec{v}_2 + \cdots + a_n\vec{v}_n$$

We can change the scalar values to adjust for the combined vectors. Thus, we have shown that $\vec{q} \in \text{span}\{\vec{v}_1 + \vec{v}_2, \vec{v}_2, \ldots, \vec{v}_n\}$. Therefore, we have $\text{span}\{\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_n\} \subseteq \text{span}\{\vec{v}_1 + \vec{v}_2, \vec{v}_2, \ldots, \vec{v}_n\}$. Now, we must show the other direction. Suppose we have some arbitrary $\vec{r} \in \text{span}\{\vec{v}_1 + \vec{v}_2, \vec{v}_2, \ldots, \vec{v}_n\}$. For some scalars b_i:

$$\vec{r} = b_1(\vec{v}_1 + \vec{v}_2) + b_2\vec{v}_2 + \cdots + b_n\vec{v}_n = b_1\vec{v}_1 + (b_1 + b_2)\vec{v}_2 + \cdots + b_n\vec{v}_n.$$

Thus, we have shown that $\vec{r} \in \text{span}\{\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_n\}$. Therefore, we have $\text{span}\{\vec{v}_1 + \vec{v}_2, \vec{v}_2, \ldots, \vec{v}_n\} \subseteq \text{span}\{\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_n\}$. Combining this with the earlier result, the spans are thus the same.

6. Image Stitching

Learning Objective: *This problem is similar to one that students might experience in an upper division computer vision course. Our goal is to give students a flavor of the power of tools from fundamental linear algebra and their wide range of applications.*

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 away as they need 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.**

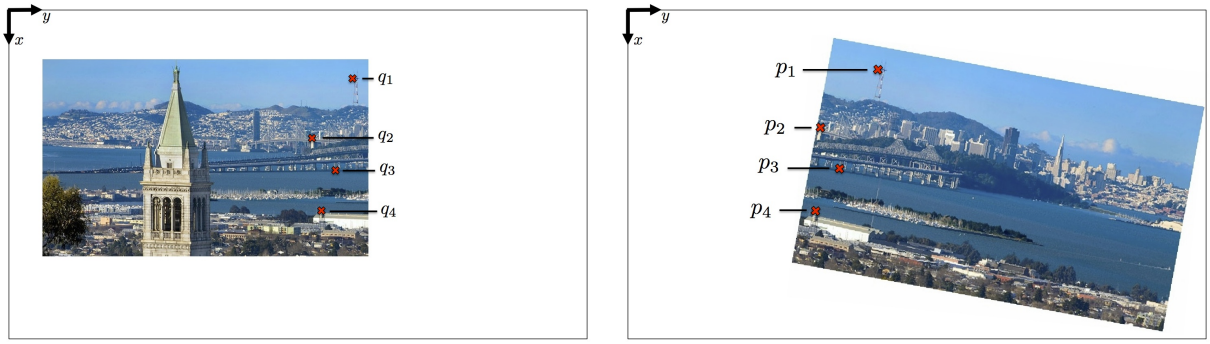


Figure 5: Two images to be stitched together with pairs of matching points labeled.

We will use vectors to represent the common points which are 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 their corresponding 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 $\vec{q}$.

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

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,

$$\vec{v}_2 = \begin{bmatrix} 2 & 2 \\ -2 & 2 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix}. \tag{9}$$

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 scaled, 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 each pair of points' elements: 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 based on what you found in part b). Transform these linear equations into a matrix equation, so that we can 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 $\vec{T} = \begin{bmatrix} -150 \\ -250 \end{bmatrix}$.

7. Homework Process and Study Group

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

Solution:

I worked on this homework with...

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

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

EECS16A: Homework 2

Problem 2: Finding Charges from Potential Measurements

In [1]:
```
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)
```

```
[1. 2. 3.]
```

Problem 3: Kinematic Model for a Simple Car

This script helps to visualize the difference between a nonlinear model and a corresponding linear approximation for a simple car. What you should notice is that the linear model is similar to the nonlinear model when you are close to the point where the approximation is made.

First, run the following block to set up the helper functions needed to simulate the vehicle models and plot the trajectories taken.

In [1]:

```
''' Problem/Model Setup'''
import numpy as np
import matplotlib.pyplot as plt

# Vehicle Model Constants
L = 1.0  # length of the car, meters
dt = 0.1 # time difference between timestep (k+1) and timestep k, seconds

''' Nonlinear Vehicle Model Update Equation '''
def nonlinear_vehicle_model(initial_state, inputs, num_steps):
    x     = initial_state[0] # x position, meters
    y     = initial_state[1] # y position, meters
    theta = initial_state[2] # heading (wrt x-axis), radians
    v     = initial_state[3] # speed, meters per second

    a = inputs[0]            # acceleration, meters per second squared
    phi = inputs[1]          # steering angle, radians

    state_history = []       # array to hold state values as the time step
    state_history.append([x,y,theta,v]) # add the initial state (i.e. k = 0

    for i in range(0, num_steps):
        # Find the next state, at time k+1, by applying the nonlinear model
        x_next     = x     + v * np.cos(theta) * dt
        y_next     = y     + v * np.sin(theta) * dt
        theta_next = theta + v/L * np.tan(phi) * dt
        v_next     = v     + a * dt

        # Add the next state to the history.
        state_history.append([x_next,y_next,theta_next,v_next])

        # Advance to the next state, at time k+1, to get ready for next loop
        x = x_next
        y = y_next
        theta = theta_next
        v = v_next

    return np.array(state_history)

''' Linear Vehicle Model Update Equation '''
def linear_vehicle_model(A, B, initial_state, inputs, num_steps):
    # Note: A should be a 4x4 matrix, B should be a 4x2 matrix for this lin

    x     = initial_state[0] # x position, meters
    y     = initial_state[1] # y position, meters
    theta = initial_state[2] # heading (wrt x-axis), radians
    v     = initial_state[3] # speed, meters per second

    a = inputs[0]            # acceleration, meters per second squared
    phi = inputs[1]          # steering angle, radians

    state_history = []       # array to hold state values as the time step
    state_history.append([x,y,theta,v]) # add the initial state (i.e. k = 0

    for i in range(0, num_steps):
        # Find the next state, at time k+1, by applying the nonlinear model
```

```
        state_next = np.dot(A, state_history[-1]) + np.dot(B, inputs)

        # Add the next state to the history.
        state_history.append(state_next)

        # Advance to the next state, at time k+1, to get ready for next loo
        state = state_next

    return np.array(state_history)

''' Plotting Setup'''
def make_model_comparison_plot(state_predictions_nonlinear, state_predictio
    f = plt.figure()
    plt.plot(state_predictions_nonlinear[0,0], state_predictions_nonlinear[
    plt.plot(state_predictions_nonlinear[:,0], state_predictions_nonlinear[
    plt.plot(state_predictions_linear[:,0], state_predictions_linear[:,1],
    plt.legend(loc='upper left')
    plt.xlim([4, 8])
    plt.ylim([9, 12])
    plt.show()
```

Part B

Task: Fill in the matrices A and B for the linear system approximating the nonlinear vehicle model under small heading and steering angle approximations.

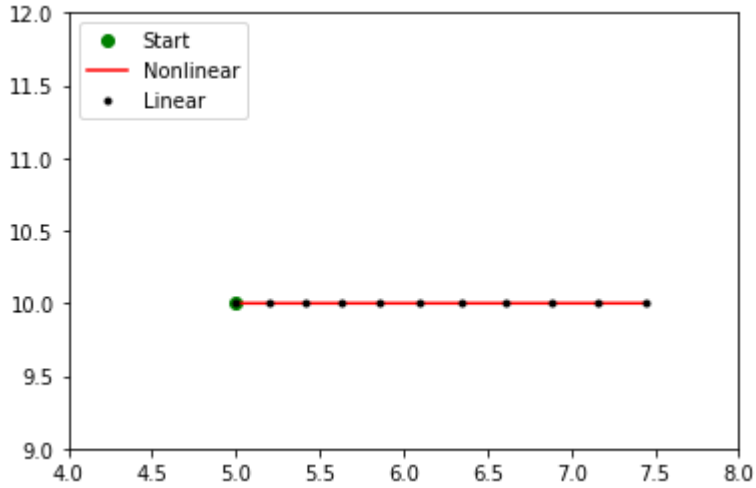
In [2]:

```
A = np.array([[1,0,0,dt],
              [0,1,0,0],
              [0,0,1,0],
              [0,0,0,1]])

B = np.array([[0,0],
              [0,0],
              [0,0],
              [dt,0]])
```

Part C

Task: Fill out the state and input values from Part C and look at the resulting plot. The plot should help you to visualize the difference between using a linear model and a nonlinear model for this specific starting state and input.

In [5]:
```
x_init  = 5.0
y_init  = 10.0
theta_init = 0.0
v_init     = 2.0
a_input    = 1.0
phi_input  = 0.0001

state_init = [x_init, y_init, theta_init, v_init]
state_predictions_nonlinear = nonlinear_vehicle_model(state_init, [a_input,
state_predictions_linear = linear_vehicle_model(A, B, state_init, [a_input,

make_model_comparison_plot(state_predictions_nonlinear, state_predictions_l

# The problem didn't ask you for the actual values, but it is useful to see

print(state_predictions_nonlinear[10])
print(state_predictions_linear[10])

```

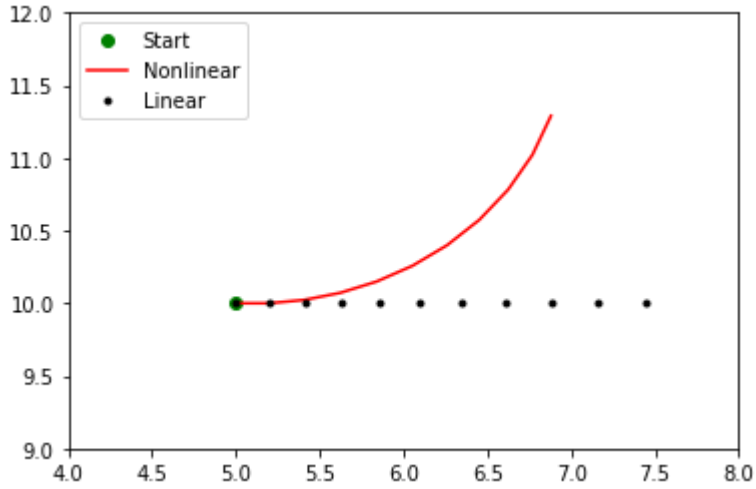



```
[7.44999998e+00 1.00002697e+01 2.45000001e-04 3.00000000e+00]
[ 7.45 10.    0.    3.  ]
```

Part D

Task: Fill out the state and input values from Problem D and look at the resulting plot. The plot should help you to visualize the difference between using a linear model and a nonlinear model for this specific starting state and input.

In [6]:
```
x_init  = 5.0
y_init  = 10.0
theta_init = 0.0
v_init     = 2.0
a_input    = 1.0
phi_input  = 0.5

state_init = [x_init, y_init, theta_init, v_init]
state_predictions_nonlinear = nonlinear_vehicle_model(state_init, [a_input,
state_predictions_linear = linear_vehicle_model(A, B, state_init, [a_input,

make_model_comparison_plot(state_predictions_nonlinear, state_predictions_l

# The problem didn't ask you for the actual values, but it is useful to see

print(state_predictions_nonlinear[10])
print(state_predictions_linear[10])
```




```
[ 6.87984693 11.28998941  1.3384411   3.        ]
[ 7.45 10.    0.    3.  ]
```

Problem 6: Image Stitching

This section of the notebook continues the image stiching problem. Be sure to have a `figures` folder in the same directory as the notebook. The `figures` folder should contain the files:

```
Berkeley_banner_1.jpg
Berkeley_banner_2.jpg
stacked_pieces.jpg
lefthalfpic.jpg
righthalfpic.jpg
```

Note: This structure is present in the provided HW2 zip file.

Part D

Run the next block of code before proceeding

In [7]:

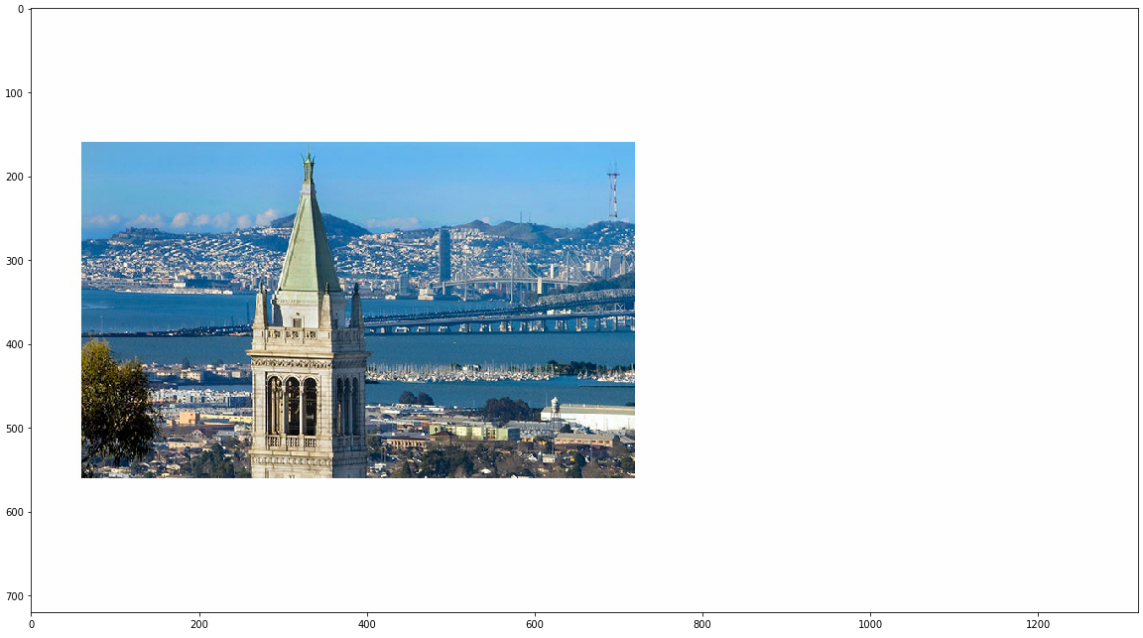
```
import numpy as np
import numpy.matlib
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from numpy import pi, cos, exp, sin
import matplotlib.image as mpimg
import matplotlib.transforms as mtransforms


#%matplotlib inline

#loading images
image1=mpimg.imread('figures/Berkeley_banner_1.jpg')
image1=image1/255.0
image2=mpimg.imread('figures/Berkeley_banner_2.jpg')
image2=image2/255.0
image_stack=mpimg.imread('figures/stacked_pieces.jpg')
image_stack=image_stack/255.0


image1_marked=mpimg.imread('figures/lefthalfpic.jpg')
image1_marked=image1_marked/255.0
image2_marked=mpimg.imread('figures/righthalfpic.jpg')
image2_marked=image2_marked/255.0

def euclidean_transform_2to1(transform_mat,translation,image,position,LL,UL
    new_position=np.round(transform_mat.dot(position)+translation)
    new_position=new_position.astype(int)


    if (new_position>=LL).all() and (new_position<UL).all():
        values=image[new_position[0][0],new_position[1][0],:]
    else:
        values=np.array([2.0,2.0,2.0])

    return values

def euclidean_transform_1to2(transform_mat,translation,image,position,LL,UL
    transform_mat_inv=np.linalg.inv(transform_mat)
    new_position=np.round(transform_mat_inv.dot(position-translation))
    new_position=new_position.astype(int)

    if (new_position>=LL).all() and (new_position<UL).all():
        values=image[new_position[0][0],new_position[1][0],:]
    else:
        values=np.array([2.0,2.0,2.0])

    return values

def solve(A,b):
    try:
        z = np.linalg.solve(A,b)
    except:
        raise ValueError('Rows are not linearly independent. Cannot solve s
    return z
```

We will stick to a simple example and just consider stitching two images (if you can stitch two pictures, then you could conceivably stitch more by applying the same technique over and over again).

Daniel decided to take an amazing picture of the Campanile overlooking the bay. Unfortunately, the field of view of his camera was not large enough to capture the entire scene, so he decided to take two pictues and stich them together.

The next block will display the two images.

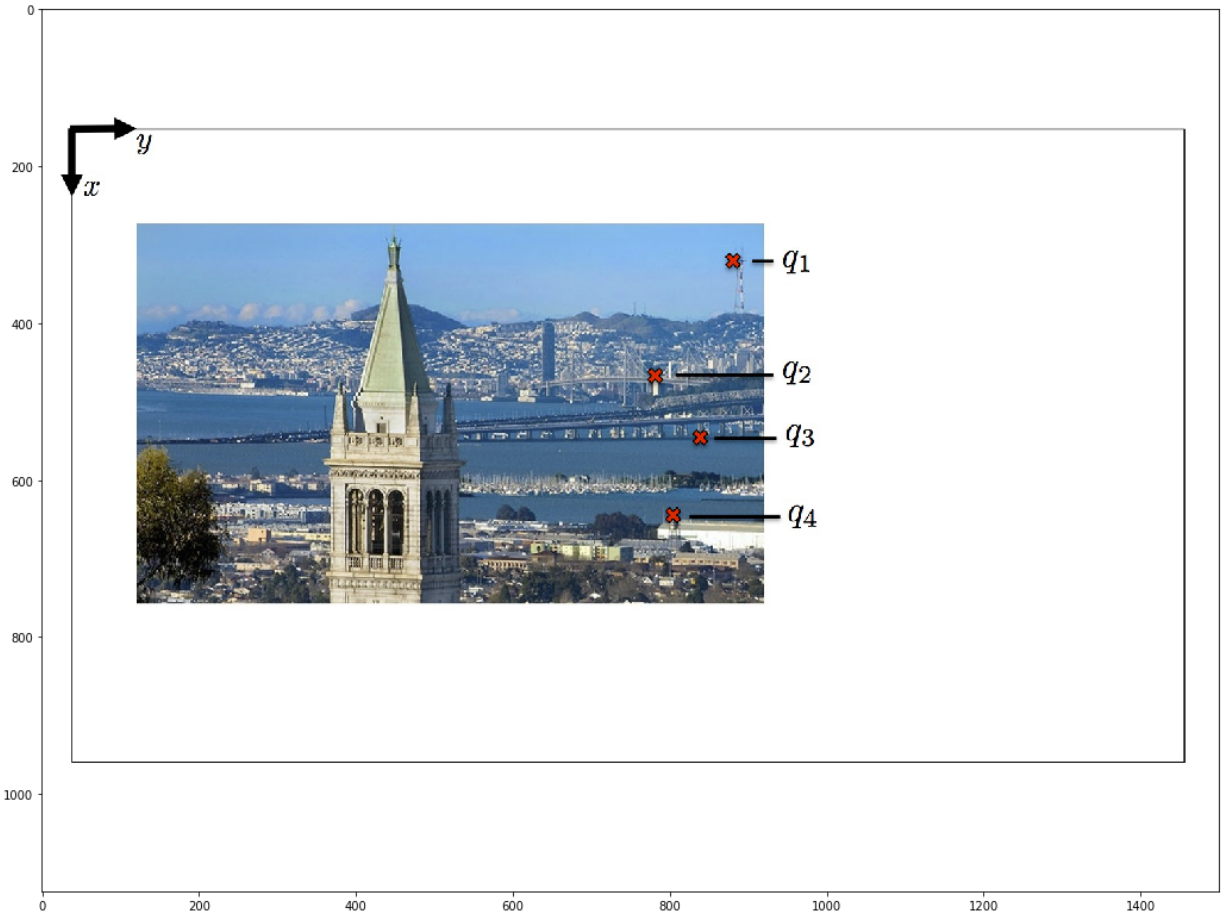
In [8]:
```
plt.figure(figsize=(20,40))

plt.subplot(311)
plt.imshow(image1)

plt.subplot(312)
plt.imshow(image2)

plt.subplot(313)
plt.imshow(image_stack)

plt.show()
```

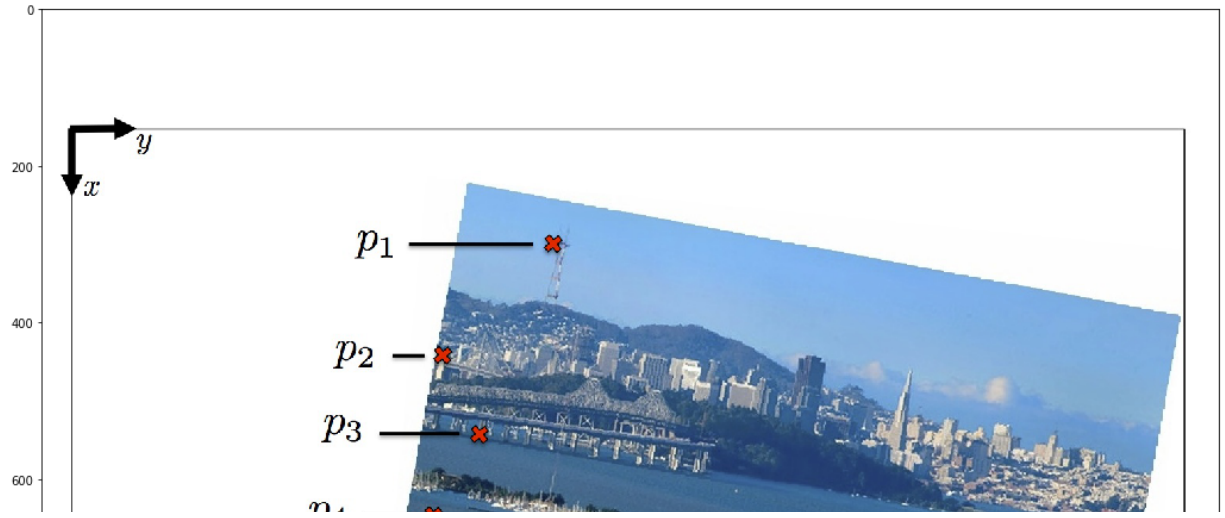


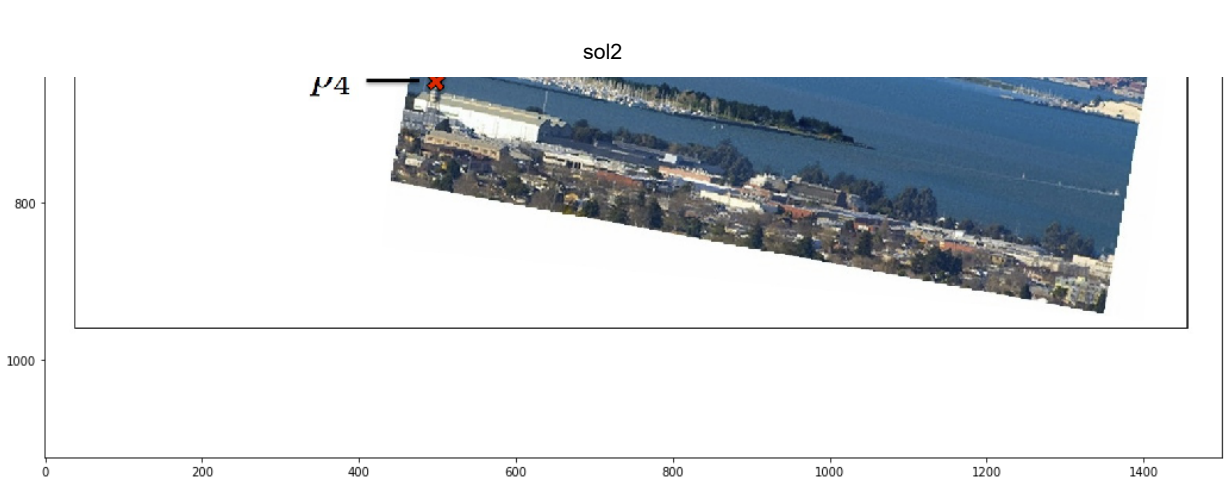
Once you apply Marcela's algorithm on the two images you get the following result (run the next block):

In [9]:
```
plt.figure(figsize=(20,30))

plt.subplot(211)
plt.imshow(image1_marked)

plt.subplot(212)
plt.imshow(image2_marked)
```

Out[9]: <matplotlib.image.AxesImage at 0x1f842c4e6a0>







As you can see Marcela's algorithm was able to find four common points between the two images. These points expressed in the coordinates of the first image and second image are

$$\vec{p}_1 = \begin{bmatrix} 200 \\ 700 \end{bmatrix} \quad \vec{p}_2 = \begin{bmatrix} 310 \\ 620 \end{bmatrix} \quad \vec{p}_3 = \begin{bmatrix} 390 \\ 660 \end{bmatrix} \quad \vec{p}_4 = \begin{bmatrix} 460 \\ 630 \end{bmatrix}$$

$$\vec{q}_1 = \begin{bmatrix} 162.2976 \\ 565.8862 \end{bmatrix} \quad \vec{q}_2 = \begin{bmatrix} 285.4283 \\ 458.7469 \end{bmatrix} \quad \vec{q}_3 = \begin{bmatrix} 385.2465 \\ 498.1973 \end{bmatrix} \quad \vec{q}_4 = \begin{bmatrix} 465.7892 \\ 455.0132 \end{bmatrix}$$

It should be noted that in relation to the image the positive x-axis is down and the positive y-axis is right. This will have no bearing as to how you solve the problem, however it helps in interpreting what the numbers mean relative to the image you are seeing.

Using the points determine the parameters $R_{11}, R_{12}, R_{21}, R_{22}, T_x, T_y$ that map the points from the first image to the points in the second image by solving an appropriate system of equations. Hint: you do not need all the points to recover the parameters.

In [10]:
```
# Note that the following is a general template for solving for 6 unknowns j
# You do not have to use the following code exactly.
# All you need to do is to find parameters R_11, R_12, R_21, R_22, T_x, T_y
# If you prefer finding them another way it is fine.

A = np.array([[200, 700, 0, 0, 1, 0],
              [0, 0, 200, 700, 0, 1 ],
              [310, 620, 0, 0, 1, 0],
              [0, 0, 310, 620, 0, 1 ],
              [460, 630, 0, 0, 1, 0],
              [0, 0, 460, 630, 0, 1 ]])

b = np.array([[162.2976], [565.8862], [285.4283], [458.7469], [465.7892], [

A = A.astype(float)
b = b.astype(float)

# solve the linear system for the coefficiens
z = solve(A,b)

#Parameters for our transformation
R_11 = z[0,0]
R_12 = z[1,0]
R_21 = z[2,0]
R_22 = z[3,0]
T_x  = z[4,0]
T_y  = z[5,0]
```

Stitch the images using the transformation you found by running the code below.

Note that it takes about 40 seconds for the block to finish running on a modern laptop.

In [11]:
```
matrix_transform=np.array([[R_11,R_12],[R_21,R_22]])
translation=np.array([T_x,T_y])

#Creating image canvas (the image will be constructed on this)
num_row,num_col,blah=image1.shape
image_rec=1.0*np.ones((int(num_row),int(num_col),3))

#Reconstructing the original image

LL=np.array([[0],[0]]) #lower limit on image domain
UL=np.array([[num_row],[num_col]]) #upper limit on image domain

for row in range(0,int(num_row)):
    for col in range(0,int(num_col)):
        #notice that the position is in terms of x and y, so the c
        position=np.array([[row],[col]])
        if image1[row,col,0] > 0.995 and image1[row,col,1] > 0.995 and imag
            temp = euclidean_transform_2to1(matrix_transform,translation,im
            image_rec[row,col,:] = temp
        else:
            image_rec[row,col,:] = image1[row,col,:]


plt.figure(figsize=(20,20))
plt.imshow(image_rec)
plt.axis('on')
plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for flo
ats or [0..255] for integers).
```

In [ ]:
```

```