AMATH 301 – Winter 2023
Homework 8

Due: 11:59pm, March 10, 2023.

Instructions for submitting:

- Coding problems are submitted to Gradescope as a python script/Jupyter notebook (`.py` or `.ipynb` file). (If you use Jupyter Notebook, remove all "magic": the lines that begin with %). You have **8 attempts** (separate submissions to Gradescope) for the coding problems.

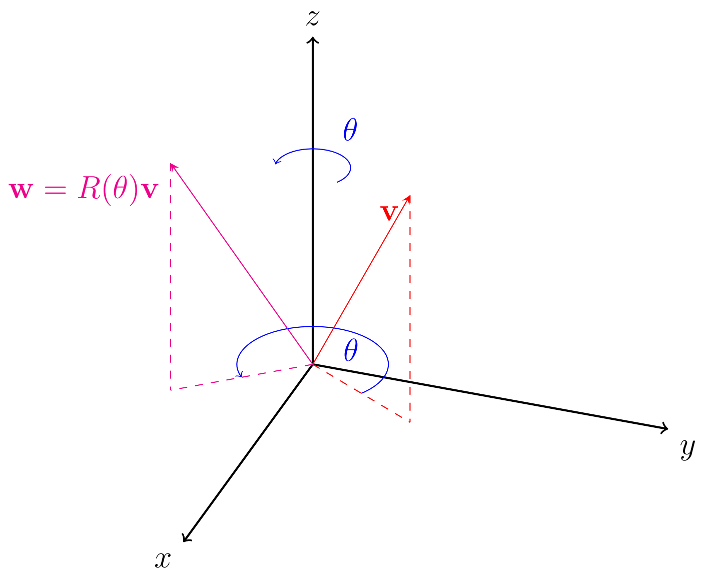
- Writeup problems are submitted to Gradescope as a single `.pdf` file that contains text, plots, and code at the end of the file for reference. Put the problems in order and label each writeup problem. When you submit, **you must indicate which problem is which on Gradescope (including the code that belongs to the problem). Failure to do so will result in a 10% penalty.**. **All code used for each problem should be included at the end of that problem in your .pdf file and be marked as part of the problem on Gradescope. Failure to include code will result in a 25% penalty.**

Coding problems

1. There are a number of applications where one might want to rotate an object in $\mathbb{R}^3$ (3-dimensional space) about a certain axis. Two such examples are (a) in computer graphics where objects are represented on the screen as objects in $\mathbb{R}^3$ (see e.g., Ray Tracing if you are interested) and (b) in aircraft control where rotations about the different axes are called *yaw, pitch,* and *roll* (see e.g., aircraft principle axes if you are interested). Imagine there is a vector $\mathbf{v} = (v_x, v_y, v_z)^T$. To rotate this vector by an angle θ around the z axis, you can multiply $\mathbf{v}$ on the left by the matrix

$$R(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

In other words, $\mathbf{w} = R\mathbf{v}$ is the vector $\mathbf{v} = (v_x, v_y, v_z)^T$ rotated by an angle θ around the z-axis. The following picture might help you understand: the red vector $\mathbf{v}$ is rotated by an angle θ (in blue) to get the new vector $\mathbf{w} = R(\theta)\mathbf{v}$ (magenta).



(a) Define the matrix A to be the matrix that rotates a vector by an angle of $\theta = \pi/4$ around the z axis. (**Hint: a nice way to do this is to define $R(\theta)$ as an anonymous function with one input and which outputs a matrix.**) Save the matrix A to the variable `A1`.

(b) I have a vector $\mathbf{y} = (3, \pi, 4)^T$ which was rotated by the matrix A defined above. Before it was rotated, the vector was called $\mathbf{x}$. I want to know what $\mathbf{x}$ was before rotating it. Setup the equation which is used to find $\mathbf{x}$ and then solve for $\mathbf{x}$ using `scipy.linalg.solve` in python. Save the resulting vector (as a column vector) to the variable `A2`.

2. Consider the truss bridge diagrammed below. The bridge is composed of steel beams that are connected at joints or *nodes*. Each beam exerts a force, F_j, on the nodes that it touches. When the bridge is at rest, these forces balance. A positive force means that the beam is in tension: it is being pulled apart by the nodes. A negative force means that the beam is being pushed inwards and is in compression.

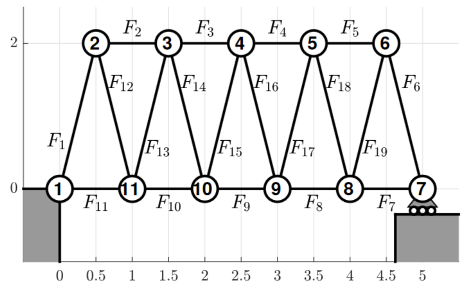


Figure 1: A truss bridge

Cars and trucks will pass over this bridge, and we would like to figure out what forces will act on the bridge as its load changes. Let W_8, W_9, W_{10}, and W_{11} be the weights, measured in Newtons, of vehicles on the bridge located at nodes 8, 9, 10, and 11 respectively.

We can find equations for the unknown forces with Newton's second and third laws of motion.

Don't worry if you don't understand where these formulas come from. You only need to use them to create the matrix equation $Ax = b$.

- x- and y-forces sum to zero on node (2).

$$\frac{-1}{\sqrt{17}}F_1 + F_2 + \frac{1}{\sqrt{17}}F_{12} = 0 \tag{1}$$

$$\frac{-4}{\sqrt{17}}F_1 - \frac{4}{\sqrt{17}}F_{12} = 0 \tag{2}$$

- x and y-forces sum to zero on node (3)

$$-F_2 + F_3 - \frac{1}{\sqrt{17}}F_{13} + \frac{1}{\sqrt{17}}F_{14} = 0 \tag{3}$$

$$\frac{-4}{\sqrt{17}}F_{13} - \frac{4}{\sqrt{17}}F_{14} = 0 \tag{4}$$

- x and y-forces sum to zero on node (4)

$$-F_3 + F_4 - \frac{1}{\sqrt{17}}F_{15} + \frac{1}{\sqrt{17}}F_{16} = 0 \tag{5}$$

$$\frac{-4}{\sqrt{17}}F_{15} - \frac{4}{\sqrt{17}}F_{16} = 0 \tag{6}$$

- x-forces sum to zero on node (5)

$$-F_4 + F_5 - \frac{1}{\sqrt{17}}F_{17} + \frac{1}{\sqrt{17}}F_{18} = 0 \tag{7}$$

$$\frac{-4}{\sqrt{17}}F_{17} - \frac{4}{\sqrt{17}}F_{18} = 0 \tag{8}$$

- x and y-forces sum to zero on node (6)

$$-F_5 + \frac{1}{\sqrt{17}}F_6 - \frac{1}{\sqrt{17}}F_{19} = 0 \tag{9}$$

$$-\frac{4}{\sqrt{17}}F_6 - \frac{4}{\sqrt{17}}F_{19} = 0 \tag{10}$$

- x-forces sum to zero on node (7)

$$-\frac{1}{\sqrt{17}}F_6 - F_7 = 0 \tag{11}$$

- x and y-forces sum to zero on node (8)

$$F_7 - F_8 - \frac{1}{\sqrt{17}} F_{18} + \frac{1}{\sqrt{17}} F_{19} = 0 \tag{12}$$

$$\frac{4}{\sqrt{17}} F_{18} + \frac{4}{\sqrt{17}} F_{19} - W_8 = 0 \tag{13}$$

- x and y-forces sum to zero on node (9)

$$F_8 - F_9 - \frac{1}{\sqrt{17}} F_{16} + \frac{1}{\sqrt{17}} F_{17} = 0 \tag{14}$$

$$\frac{4}{\sqrt{17}} F_{16} + \frac{4}{\sqrt{17}} F_{17} - W_9 = 0 \tag{15}$$

- x and y-forces sum to zero on node (10)

$$F_9 - F_{10} - \frac{1}{\sqrt{17}} F_{14} + \frac{1}{\sqrt{17}} F_{15} = 0 \tag{16}$$

$$\frac{4}{\sqrt{17}} F_{14} + \frac{4}{\sqrt{17}} F_{15} - W_{10} = 0 \tag{17}$$

- x and y-forces sum to zero on node (11)

$$F_{10} - F_{11} - \frac{1}{\sqrt{17}} F_{12} + \frac{1}{\sqrt{17}} F_{13} = 0 \tag{18}$$

$$\frac{4}{\sqrt{17}} F_{12} + \frac{4}{\sqrt{17}} F_{13} - W_{11} = 0 \tag{19}$$

(a) Write this linear system as a matrix equation $\boldsymbol{Ax} = \mathbf{b}$ where $\boldsymbol{x}$ is a vector of the forces:

$$\boldsymbol{x} = \begin{pmatrix} F_1 \\ F_2 \\ \vdots \\ F_{19} \end{pmatrix}.$$

This is done by putting the first equation in the first row, the second in the second row, etc.

(b) Set $W_8 = 12000$, $W_9 = 9200$, $W_{10} = 9000$, and $W_{11} = 19200$, then create the matrix `A` and vector `b`. **Note:** This matrix can be hard to input and is prone to mistakes. I have given you part of the code in the template, but not all of it because it is goot practice to create larger matrices. Instead of directly typing in every element of the matrix, you should create a 19×19 matrix full of zeros (using `zeros`) and then edit the required entries of this matrix, *e.g.,* `A[i,j] = ...`. This will lead to the least amount work on your end and will be easiest to debug. It will also be helpful to define some other variable, `s = 1/sqrt(17)` which will be used several times when creating the matrix. **You can test if you input `A` correctly by checking that `np.linalg.norm(A)` returns `5.92105` and `sum(sum(A))` returns** `-3.1828` (rounded in both cases).

(c) Save $\boldsymbol{A}$ to the variable `A3`. This will tell if you defined `A` correctly.

(d) Solve $\boldsymbol{Ax} = \mathbf{b}$ for $\boldsymbol{x}$ by using scipy.linalg.solve. Save the answer $\boldsymbol{x}$ to the variable `A4` (as a column vector). This will test if you defined both `A` and `b` correctly in your script.

(e) Find the largest force (since both negative and positive forces can be "large", the "largest force" should be interpreted as largest *in absolute value*) of the vector $\boldsymbol{x}$. Save its absolute value to the variable `A5`. You should use the commands `np.amax` and `np.abs`.

(f) We now want to think about *stress-testing* the bridge to figure out the maximal weight for a vehicle at node 10. To do this, suppose that we add weight to the truck at position 10 in increments of 5 Newtons until the bridge collapses. Each bridge member can withstand 44000 Newtons of compression or tension (i.e., positive or negative forces.) Therefore, the bridge will collapse when the absolute value of the largest force is **larger** than 44000. Find the smallest weight of the truck at position 10 for which the bridge collapses. Save your answer (the weight at which the bridge collapses) to the variable `A6`. Find which force is the one that exceeded 44000 Newtons. If the force is F_j, save the value of j to the variable `A7`.

Note that `np.argmax(b)` gives the index corresponding to the largest value of `b` and that index 0 corresponds to force 1 (since python indexes starting at 0).

Note: Two forces exceed 44000 Newtons at the same time. Record the smaller of the two value of j to `A7`.

Note 2: You can do this in two ways: using a big for loop (that iterates many times), or, you can look up how to use a "while" loop.

3. (Note that this problem goes along with Writeup Problem 1, you may want to look at that problem too before completing the coding here. We will talk about this problem in class on Monday, March 6.) In order to complete this problem in Thonny, you will need to import *opencv-python.*

In this problem, we will use the SVD to efficiently compress an image. To do this problem you will need to download the file `olive.jpg` from Canvas. When you have the image `olive.jpg`, you can load the image using

```
A = cv2.imread('olive.jpg', 0)
```

The variable `A` will now be a 4032×3024 matrix (`A.shape = (4032, 3024)`).

(a) The image is stored as a matrix because it corresponds to the pixels (4032 pixels in rows, 3024 pixels in columns). Calculate the total number of pixels stored in the image and save the result to the variable `A8`.

(b) Perform a singular value decomposition on the matrix $\mathbf{A}$ using `np.linalg.svd(A, full_matrices=False`. Save the matrix U to the variable `A9`, the array Σ to the variable `A10`, and the matrix `V` to the variable `A11` (note that it is V, not V^T that you are saving here, do `.T` to *transpose* in python).

(c) Create a 15×1 array that contains the fifteen largest singular values of A. Store this vector to the variable `A12`.

(d) The proportion of the total energy that is contained in the rank-1 approximation of $\mathbf{A}$ (defined in Week 10) is the largest singular value divided by the sum of all of the singular values. Calculate the proportional energy of the rank-1 approximation and save it to the variable `A13`.

(e) The proportion of the total energy that is contained in the rank-15 approximation of $\mathbf{A}$ is the sum of the fifteen largest singular values divided by the sum of all of the singular values. Calculate the proportional energy of the rank-15 approximation and save it to the variable `A14`.

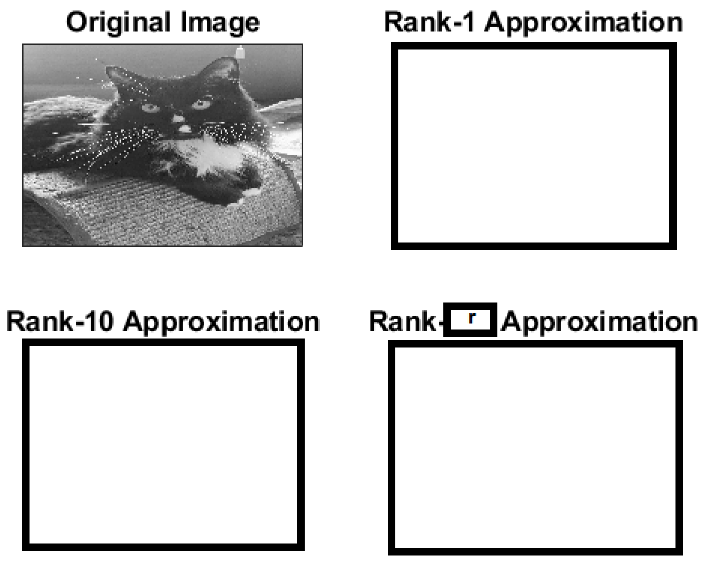
(f) Find the smallest value of r such that the energy of the rank-r approximation is greater than or equal to 0.75 (75%) of the total energy. Save the value of r to the variable `A15`. It may be helpful to look up the command `np.where` for this problem.

Writeup problems

1. This problem is connected to Coding Problem 3. In this problem we compare the true image to the low-rank approximations of the image. This will be demonstrated in full on Tuesday and in the activity on Wednesday, Week 9. Recall from class that you can plot a grayscale image which is stored as a matrix `A` by using the commands

```
ax.imshow(A, cmap='gray')
```

(a) Create a 2×2 grid of axes using `fig, ax = plt.subplots(2,2)`. Update each of the axes to include the full image, the rank-1 approximation, rank-15 approximation, and the rank-r approximation, using the r from `A14`. They should be arranged in the following order and have titles (for the rank-r title you may write the actual value of r instead of using the letter r). To make a title, use, e.g., `ax[0,0].set_title('Original Image')`. Your image may look a little bit different from what we see below.



(b) Calculate the total number of pixels in the full image (`A8`). Then calculate how many values you would need to store in order to store the rank-r approximation (with the value of r from `A15`). State both numbers in your write up.

(c) State the ratio of points used in the full image to the number of points in the rank-r approximation (r from `A15`). What does this tell you about the efficiency of storing the two images (the full image an the one that stores 75% of the image energy)?