IMT574 Problem Set 2: Linear Algebra (100pt)

Your name:

Deadline: Wed Jan 22/Thu Jan 23

Instructions

This is the linear algebra/matrix manipulation problem set. It has three purposes: a) make you better and more confident with matrices in general, matrices in python, and keeping and manipulating different data in matrix form; b) get more used with matrix multiplication and transformation matrices (as mathematical objects and operations); and finally c) learn more matplotlib. The extra credit question also asks you to dig deep into it.

1. Please write clearly! Answer each question in a way that if the code chunks are removed from your document, the result is still readable!
2. Please keep data file in the same folder as your code, and read these w/o any path like `"data.csv"` (or `"./data.csv"`). This makes it much easier to check your code!

Introduction

In the lab we used rotation matrix

$$\mathsf{R}(\alpha) = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix}$$

to rotate images in matrix form. For instance, if an image is given as

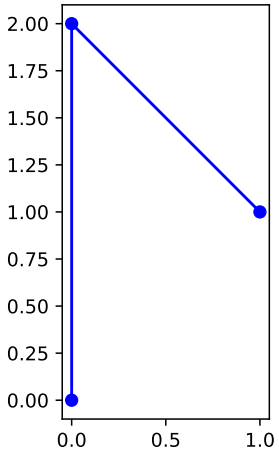
$$\mathsf{A} = \begin{pmatrix} 0 & 0 \\ 0 & 2 \\ 1 & 1 \end{pmatrix},$$

it can be rotated as

$$\mathsf{A}^{\alpha} = \mathsf{A} \cdot \mathsf{R}(\alpha).$$

The image can be plotted by

```
import numpy as np
import matplotlib.pyplot as plt
A = np.array([[0,0], [0,2], [1,1]])
#
fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.plot(A[:,0], A[:,1], marker='o', c='b')
ax.set_aspect('equal')  # we want to keep equal aspect rato
                        # to show rotated images correctly
plt.show()
```



1 Rotate Crazy Hat (30pt)

Your first task here is to rotate Crazy Hat. This is a more complex image, in particular it contains different parts that must not be connected.

1. (5pt) Load data *crazy-hat.tsv*. Inspect it.

   It contains three columns: x, y are coordinates, and *group* is a group label. Only the dots inside each group must be connected. Distinct groups must remain disconnected.

2. (15pt) Plot Crazy Hat (no rotation here). Pay attention not to connect different groups! Use the right colors: hat's outline must be black, eyes

brown, and the mouth bright red (with the line twice as wide as the rest of hat).

Hint: consider looping over unique groups for plotting; consider creating a dict for colors/line widths.

Hint2: I recommend to create a function for plotting Crazy Hat, and doing it in a form that you can rotate it later.

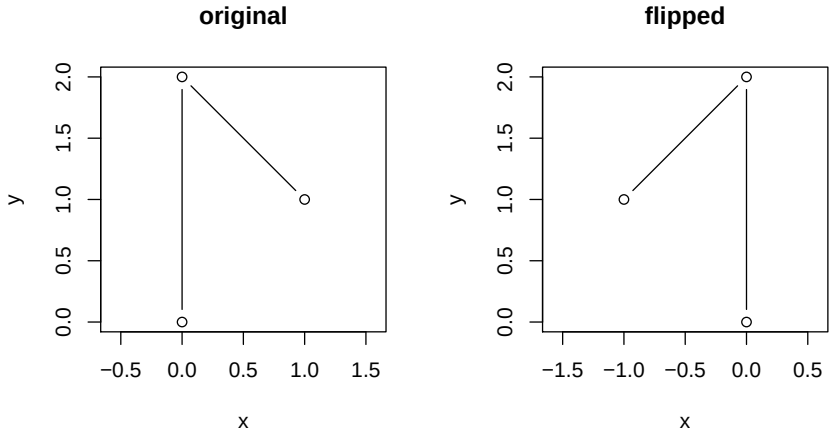
3. (10pt) Now rotate Crazy hat 66 degree *counterclockwise* (and plot it).

2 Manipulate images: Flip and stretch (40pt)

In the previous section we rotated the Crazy Hat image. You next task is to flip it, stretch it, and finally combine all these transformations into a single transformation. If you are not a fan of Crazy Hat, you can also choose another image.

1. (15pt) Construct the "flip-x" matrix F^x that flips (mirrors) the image on y axis. It should mirror (flip) the x components but leave the y components untouched. Demonstrate this with Crazy Hat.

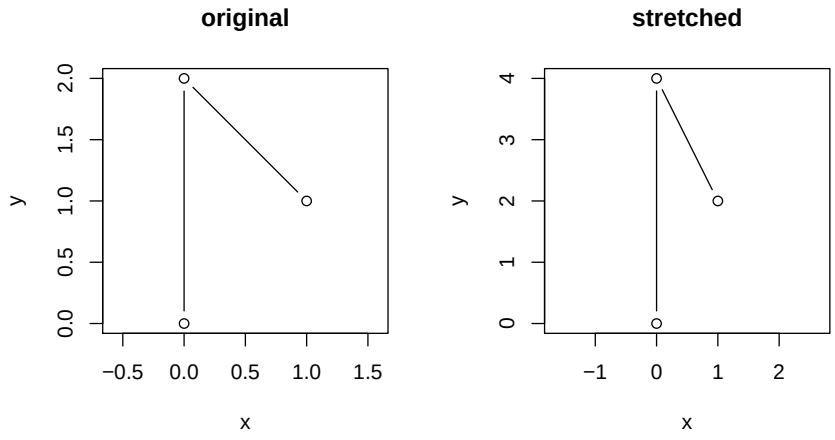
   Here is an example how the flipped inverse-1 did in the lab will look like:



2. (15pt) Create the "stretch-y" matrix $S^y(s)$. This matrix should stretch (or squish) the y components by amount s while leaving x components

untouched. For instance, if $s = 2$, the object should grow twice as tall, but no wider than the original. Demonstrate it with Crazy Hat.

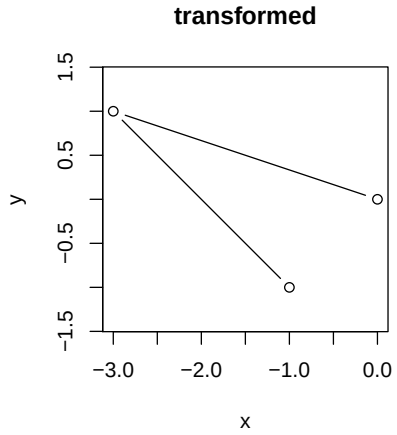
Here is an example what should happen with $2\times$ stretch:



Now you have developed three simple tools for image manipulation: rotation, flipping and stretching. All these tools can easily be chained.

3. (5pt) Do the following operations after one another: rotate it by 45 degrees counterclockwise, flip it, stretch it $2\times$, and rotate it back.

   The inverse-1, if doing this, should look like

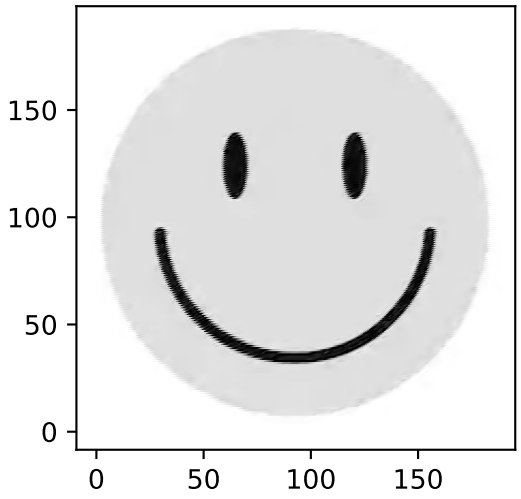


4. (5pt) finally, consider all these transformations as a single linear transformation. Print the corresponding transition matrix.

3 Manipulate bitmap images (30pt)

Now when you are done with the wireframe images it is time to take a look at bitmap images and rotate these in a similar fashion. Below is an example code to print a bitmap image using `plt.scatter`. There are other ways to plot images but this way fits best with matrix transformations.

```
from matplotlib.image import imread
img = imread('img/smiley.png')
# note: rows x columns x colors (not width x height!)
R = img[:,:,1]  # select color channel 2, G
## create coordinate matrix
xx, yy = np.meshgrid(np.arange(R.shape[1]),
                     np.arange(R.shape[0], 0, -1))
X = np.stack((xx.ravel(), yy.ravel()), axis=1)
fig = plt.figure(figsize=(3,3))
ax = fig.add_subplot(1,1,1)
ax.scatter(X[:,0], X[:,1], c=R.ravel(), marker='.', cmap="Greys_r")
ax.set_aspect('equal')
plt.show()
```



Your tasks:

1. (0pt – nothing to be written) Understand the example code above. Consult documentation of the corresponding functions, and also read Transforming Bitmap Images into Coordinate Matrix Form in the lecture notes (I try to improve the text the next few days).

2. (5pt) Read the image *dont-own-tv.png* and display it in an analogous fashion. (We will just stay with grayscale images).

   Hint: start with the smaller file *dont-own-tv-small.png*, just in case the full image is slow. If the large one is too slow, you can complete the assignment with just the small one.

   You may also experiment with different color channels (or try to plot color figure).

3. (15pt) Now stretch the image vertically (using the stretch-y matrix you did above) so that it will be square-shaped.

   Hint: you can compute the correct stretch factor using the original image shape your read with `imread`.

4. (10pt) Zoom onto just the red arrowhead from the image (extract a slice that just contains the arrowhead), and plot it in a rotated form so that the arrow points exactly left.

Note: you may have to experiment a bit to find good bounding box for the arrowhead.

Extra credit (2 extra credit pt) when plotting the arrow, create a red colormap in a way that the image will be printed in levels of red, not in grays.

Hint: check out `LinearSegmentedColormap` and it's method `from_list` in `matplotlib.colors`.