INFO370 Problem Set: Image rotation

April 16, 2022

Instructions

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. Discussing the solutions and getting help is all right, but you have to solve the problem your own. Do not copy-paste from others!

1 Matrix multiplication (8pt)

1. (6pt) Multiply these matrices/vectors *manually*. Show the math you do while solving these.

   Note: you can do this on paper, just include a picture of your solution.

   a) $\begin{bmatrix} 0 & 7 \\ 5 & 3 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} =$

   b) $\begin{bmatrix} 0 & 7 \\ 5 & 3 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ 0 & -1 \\ 0 & 1 \end{bmatrix} =$

   c) $\begin{bmatrix} 0 & 7 \\ 5 & 3 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 1 \end{bmatrix} =$

2. (2pt) now check your results on computer. You should get the same results.

---

2 Rotate Crazy Hat (21pt)

Images can be rotated by just matrix-multiplying those with *rotation matrix*

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

This matrix rotates an object matrix by angle α *clockwise* if you *post-multiply* the object matrix by it:

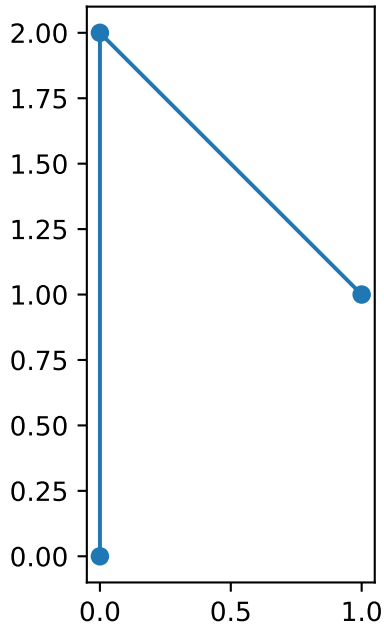
$$\mathsf{A}^{\alpha} = \mathsf{A} \cdot \mathsf{R}(\alpha)$$

where A is a $n \times 2$ matrix of x and y coordinates. Each row of this matrix corresponds to one vector that points to one of the points of this image. For instance, the following matrix gives us a flipped-1-shaped figure: A as

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

The image can be plotted by

```
import numpy as np
import matplotlib.pyplot as plt
# create the image
A = np.array([[0,0], [0,2], [1,1]])
# plot the image
ax = plt.axes()
ax.plot(A[:,0], A[:,1], marker='o')
ax.set_aspect("equal")
plt.show()
```



We ask for a line-point plot (`marker="o"`) and use aspect ratio `"equal"` to ensure that x and y lengths are of equal scale on the figure so that it will not be distorted when rotating.

Hint 1: read lecture notes section 2.3 (2.3.1, 2.3.2, in particular "matrix multiplicaton", 2.3.5). Watch 3Blue1Brown linear algebra videos. Note that while we define the rotation as

$$A^{\alpha} = A \cdot \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix},$$

3Blue1Brown uses the transposed notation where

$$A^{\alpha} = \begin{pmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{pmatrix} \cdot A.$$

These two approaches are not compatible, you have to choose either the one or the other!

Hint 2: remember that numpy trigonometric functions expect angles in radians, not degrees!

Below, your task is to rotate Crazy Hat's image. It is a similar wireframe image, the only difference is that different elements of should not be connected (they belong to different groups). The instructions below suggest an efficient way to achieve it, but you can also get there other ways.

1. (1pt) Read the data points from "crazy_hat.tsv" (available on canvas) from disk. A small excerpt of it looks like this:

```
import pandas as pd
ch = pd.read_csv("../../data/toy/crazy-hat.tsv", sep="\t")
ch.head()

##     x   y    group
## 0   0  22  outline
## 1  16 -18  outline
## 2 -16 -18  outline
## 3   0  22  outline
## 4  -3   7     leye
```

   This contains three variables: `x`, `y`, and *group*. `x` and `y` are the point coordinates (the same as matrix `A` above) while *group* indicates which points should be connected: your should connect points inside the same group (e.g. all four *outline* points in the order they are in the file) but not between different groups (e.g. do not connect *outline* with *leye*).

2. (3pt) Separate data into a coordinate matrix (call this `X`) and group id-s (call this *groups*). You can get matrix out of data frame columns with the `.values` attribute.

   Note that when you rotate the image below, you only have to manipulate `X`, *groups* will remain the same.

3. (5pt) Plot the figure: plot all `x`-`y` pairs and sequentially connect the dots. Pay attention to connect correct groups but not to connect wrong groups.

   Hint: you can do this by a loop over unique group values (remember `np.unique`), and by plotting and connecting only points for this group inside the loop.

   Hint 2: normally data plotting does not care about `x`/`y` scales. You may want to force both to be in the same scale with

```
ax = plt.axes()
ax.set_aspect("equal")
ax.plot(...)
```

4. (4pt) Convert your plotting code into a function that takes two (or more if you wish, like color, image size, etc) arguments: `X` and *groups*. Demonstrate that the function works with Crazy Hat data.

5. (4pt) Now create a function `Rot` that takes the angle α, and *returns* the rotation matrix to rotate image by this angle. Check out `np.sin`, `np.cos`, and remember that $radians = \pi \cdot degrees/180$.

   Note: you can achieve the results here in different ways, but your code will be clean and easy to follow if you follow the advice here: a plotting function that takes in the coordinate matrix, and a function that returns the rotation matrix.

6. (4pt) Now rotate the Crazy Hat figure. Select a few different angles of your choice. Remember that matrix multiplication sign is `@`, not `*`!

   You can first rotate X and thereafter plot the rotated X, or alternatively just plot $\mathsf{X} \cdot \mathtt{Rot}(\alpha)$.

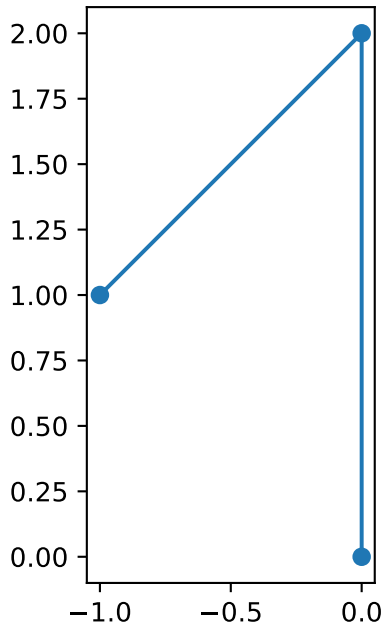
---

3 Linear transformation of images (26pt)

Now we do some more transformations. Matrix multiplication is equivalent to affine transformation of images where all lines remain lines, and the origin remains the the origin.

3.1 Flipping (mirroring) image (13pt)

By now you should be good enough with rotation matrices. Next step will be to flip (mirror) and stretch (or shrink) and image. We construct a "flip-x" $\mathsf{F}^{\mathtt{x}}$ matrix that flips the x-axis (mirrors the object around the y-axis). So if you multiply the flipped-1 with it as $\mathsf{A} \cdot \mathsf{F}^{\mathtt{x}}$, you'll get



Note that all horizontal coordinates have the same value, just negative now.

1. (6pt) Construct the "flip-x" matrix $\mathsf{F}^{\mathtt{x}}$. It should be a 2×2 matrix that inverses the x-components while leaving y-components intact. You can think in terms of generic matrix multiplication.

$$\mathsf{A} = \begin{pmatrix} 0 & 0 \\ 0 & 2 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} \mathsf{f}^{\mathtt{x}}_{11} & \mathsf{f}^{\mathtt{x}}_{12} \\ \mathsf{f}^{\mathtt{x}}_{21} & \mathsf{f}^{\mathtt{x}}_{22} \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 2 \\ -1 & 1 \end{pmatrix}$$

   where $\mathsf{F}^{\mathtt{x}} = \begin{pmatrix} \mathsf{f}^{\mathtt{x}}_{11} & \mathsf{f}^{\mathtt{x}}_{12} \\ \mathsf{f}^{\mathtt{x}}_{21} & \mathsf{f}^{\mathtt{x}}_{22} \end{pmatrix}$ is the "flip-x" matrix.

   Hint: you can take the unit matrix as the point of departure and see which elements you have to modify to get flip-x instead.

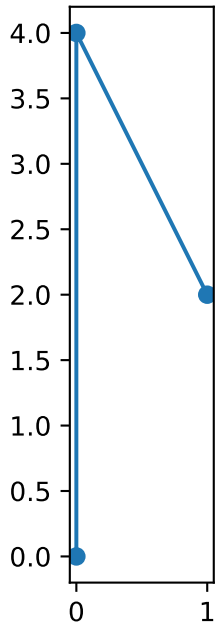
2. (5pt) Why should $\mathsf{F}^{\mathtt{x}}$ be a 2×2 matrix?

   Hint: try something else, e.g. 2×3 or 3×2 matrix. What is the result?

3. (2pt) Demonstrate that your $\mathsf{F}^{\mathtt{x}}$ flips Crazy Hat image. (Note: Crazy Hat is slightly asymmetric.)

3.2 Stretch image (13pt)

Next, let's stretch the image instead. We'll contruct "stretch-$\mathtt{y}$" matrix that stretches the image vertically. E.g, when we stretch the flipped-1 vertically by factor of 2 we should see



Note how the stretched version is much taller but still still as wide as the original version. Obviously, you can shrink the image in a similar fashion.

1. (5pt) Create the "stretch-$\mathtt{y}$" matrix $\mathsf{S}^{\mathtt{y}}(\mathtt{s})$ where $\mathtt{s}$ is the "stretch factor", 2 in the example above.

   Exactly as a above, it should be a 2×2 matrix that stretches the $\mathtt{y}$-components while leaving $\mathtt{x}$-components untouched. And you can think in terms of generic matrix multiplication:

$$\mathsf{A} = \begin{pmatrix} 0 & 0 \\ 0 & 2 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} \mathtt{s}^{\mathtt{y}}_{11} & \mathtt{s}^{\mathtt{y}}_{12} \\ \mathtt{s}^{\mathtt{y}}_{21} & \mathtt{s}^{\mathtt{y}}_{22} \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 4 \\ 1 & 2 \end{pmatrix}$$

   where $\mathsf{S}^2 = \begin{pmatrix} \mathtt{s}^{\mathtt{y}}_{11} & \mathtt{s}^{\mathtt{y}}_{12} \\ \mathtt{s}^{\mathtt{y}}_{21} & \mathtt{s}^{\mathtt{y}}_{22} \end{pmatrix}$ is the "stretch-$\mathtt{y}$" matrix.

   Again, you can take the unit matrix as the point of departure and see which elements you have to modify to make it to stretch-$\mathtt{y}$.

   This matrix should stretch (or compress) the $\mathtt{y}$ components by amount $\mathtt{s}$ while leaving $\mathtt{x}$ components untouched. Demonstrate it with your object.

   Hint: it is more practical to create a function that takes stretch factor $\mathtt{s}$ as input and returns the corresponding stretch matrix.

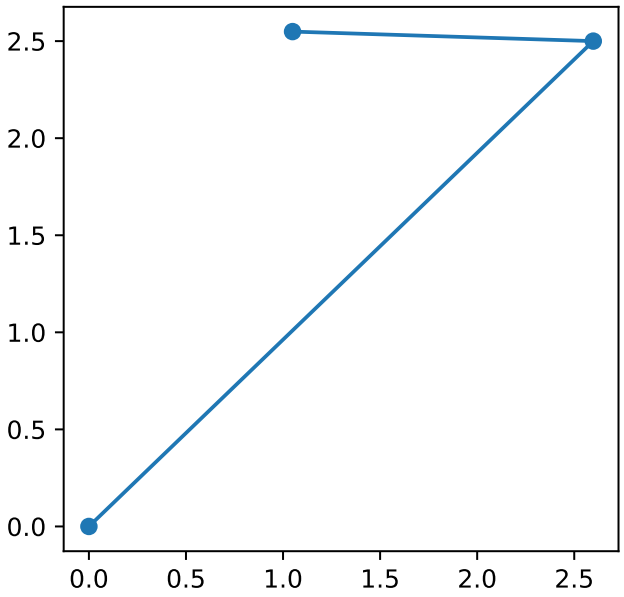
2. (2pt) Demonstrate that your matrix works correctly by stretching Crazy Hat vertically by factor 2.

3. (7pt) Finally, let's combine these transformation. Use Crazy Hat to perform the following operations: flip along the 45°-line, and stretch it 2× along the 45°-line (see the example below). I have in mind the 45° line that to SE.

   Note: this task is a bit confusing because "flip" can mean two things. First, it can be understood as moving the points along the 45° line from one side to the other. This is the meaning we used when talking about "flip-x", F^{x}, above. But one can also understand it as if we have to move the points to the opposite side of the line. We use the first meaning here, and would have called the other "flip-y".

   Hint: instead of creating a new flip-45-deg matrix, you can rotate your image so that the 45° line now aligns with the x-axis, flip over x and rotate it back. Try this!

   Example: here is the flipped-1 a) flipped along the 30°-line; and b) stretched 2× perpendicular to the same line. Try to understand this first on paper, and afterwards to replicate it on computer.



---

4 Rotate Krazy Kat (27 points)

Here your task is to rotate Krazy Kat's image.[1] Unlike Crazy Hat, this is a bitmap (png) image. You can plot it with the following code:

```
from matplotlib.image import imread
import matplotlib.pyplot as plt
import numpy as np
```

[1]By George Herriman - This file has been extracted from another file: Krazy Kat 1918-12-17 original.jpg, Public Domain, https://commons.wikimedia.org/w/index.php?curid=81653285. See also Krazy Kat's Wikipedia entry.

```
img = imread('img/krazy-kat.png')
                            # note: rows * columns (not width * height!)
print("shape:", img.shape)
## create coordinate matrix
xx, yy = np.meshgrid(np.arange(img.shape[1]),
                     np.arange(img.shape[0], 0, -1))
X = np.column_stack((xx.ravel(), yy.ravel()))
fig = plt.figure(figsize=(3,3))
ax = fig.add_subplot(1,1,1)
ax.scatter(X[:,0], X[:,1], c=img.ravel(), marker='.',
           cmap="Greys_r")
ax.set_aspect('equal')
plt.show()
```

There are two versions of Krazy Kat: 400×400 and 100×100. I recommend to start with the small one (much faster), and use the large one only for the final version.

Note: as Krazy Kat images are black-and white, you get a 2-D image matrix (called `img` in the code snippet above) when you load. In case of color images you get 3-D image tensor instead. Keep this in mind if you want to try other images.

1. (5×3pt) Get the code to run and understand what it does. In particular
   (a) Get the code to run
   (b) What is the matrix `X` that is made by stacking?
   (c) How many rows does `X` have? What does it mean, why does it have this many rows it has?
   (d) A few lines of `X` may look like:

$$\mathsf{X} = \begin{pmatrix} \dots & \dots \\ 36 & 430 \\ 37 & 430 \\ 38 & 430 \\ \dots & \dots \end{pmatrix}$$

   What do these three lines denote?
   (e) What is `img` matrix? If may contain values like $(0.9882353, 0.99215686, 0.99607843, 1)$. What do these values denote?

   Hint: you may want to consult lecture notes section 2.3.5, in particular the section about bitmap images.

2. (8pt) Now use matrix multiplication to rotate this image by a few different angles.

   Note: use *matrix rotation* we did above, various image libraries will not count!

   Suggestion: when you use the code like shown above, it may result in large and slow pdf (or html) files. You can make the pdf faster by:
   (a) instead of *plt.show*, save the image into a jpg or png file.
   (b) put the resulting saved file into your pdf/html.

3. (4pt) Use also other transforms (F^{x} and S^{y}) to manipulate the image.

---

5 Linear Regression (18pt)

You final task is to implement linear regression. This is mathematically very easy–the solution is

$$\hat{\boldsymbol{\beta}} = (\mathsf{X}^\mathsf{T}\,\mathsf{X})^{-1}\,\mathsf{X}^\mathsf{T}\,\mathbf{y}. \tag{1}$$

This involves a few matrix products, and one inverse $(\mathsf{X}^\mathsf{T}\,\mathsf{X})^{-1}$. So computing is easy. You will probably spend some time on creating the design matrix X. We do it as simple as we can and take Boston housing data (because all variables are numeric and no cleaning is needed).

1. (1pt) Load boston housing data.
2. (3pt) Run a linear regression model in the form

   $$medv_\mathrm{i} + \beta_0 + \beta_1 \cdot rm_\mathrm{i} + \beta_2 \cdot lstat_\mathrm{i} + \beta_3 \cdot zn_\mathrm{i} + \epsilon_\mathrm{i} \tag{2}$$

   Use an existing library, such as *statsmodels* or R's *lm* to do this, and display the results table.

   (You are welcome to pick different variables, feel free to include all of them, just pick more than one.)
3. (8pt) Next, the most complex task–create the *design matrix* X for this regression model. You have to create a matrix that contains four columns: a) column of ones, and b) three columns of the variables you use in the model. See lecture notes 4.1.10 *Theoretical considerations*, in particular subsections "Linear Regression: The matrix approach" and "Solving the Linear Regression Model". Example 4.1.12 shows how to create a design matrix.

   The task is actually not complex at all–one line of code is enough, just it may need you to understand the concept of design matrix and learn how to concatenate vectors. This may take a while if you have little experience with matrices.

   Hint: in numpy you can create a column of ones with `np.ones()` and add a column of ones to a matrix with `np.column_stack((ones, X))`. In R you can do `rbind(1, X)`.

   In numpy you can transform a data frame to matrix using `.values` attribute, in R use `as.matrix`.
4. (2pt) Create the outcome vector $\mathbf{y}$. Just transform the *medv* column to a vector. (In R it will be vector anyway.)
5. (3pt) Compute the solution–your estimate for the parameter vector β using (1).

   Hint: numpy inverts matrix with function `np.linalg.inv`, in R you can use `solve`.
6. (1pt) Compare the results–your computed $\boldsymbol{\beta}$ estimate should be exactly the same that you see in the linear regression table.

Finally...

How much time did you spend on this PS?