project 1:

eigenvalues, eigenvectors, and prototypes

solution(s) due:

November 28, 2019 at **12:00** via email to **bauckhag@bit.uni-bonn.de**

practical advice

Please note that your instructor is a proponent of open science and open education. Therefore, **MATLAB implementations will not be accepted** in this course.

The problem specifications you'll find below assume that you use python / numpy / scipy for your implementations. They also assume that you have imported the following

```
import numpy as np
import scipy.linalg as la
import matplotlib.pyplot as plt
```

problem specification:

task 1.1: spectral- and singular value decompositions
First of all, download the file

```
faceMatrix.npy
```

from the Data folder on the Google site that accompanies the course. It contains a data matrix $\boldsymbol{X} \in \mathbb{R}^{361\times 2429}$ whose columns $\boldsymbol{x}_i$ represent tiny face images of size 19×19 pixels.

To read this matrix into memory, you may use

```
X = np.load('faceMatrix.npy').astype('float')
```

To have a look at one of the images it contains, say $\boldsymbol{x}_{15}$, you may use the following snippet

```
x = X[:,14].reshape(19,19)
plt.imshow(x, cmap='gray')
plt.xticks([])
plt.yticks([])
plt.show()
```

Having read matrix $\boldsymbol{X}$ into memory, here is what you are supposed to do

1. normalize $\boldsymbol{X}$ such that its column mean is $\mathbf{0}$
2. compute $\boldsymbol{C} = \boldsymbol{X}\boldsymbol{X}^T$
3. compute the spectral decomposition $\boldsymbol{C} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T$ using `la.eig`
4. compute the spectral decomposition $\boldsymbol{C} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T$ using `la.eigh`
5. compare the eigenvalues you obtain from both these approaches; what do you observe ?
6. compute the singular value decomposition $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$ using `la.svd`
7. square the resulting singular values and compare them to the eigenvalues you found above; what do you observe ?
8. for the fun of it, create a plot of the three spectra you just computed; do you get any warnings or error messages ?

task 1.2: timing spectral- and singular value decompositions
Proceed just as above, but now measure the run-time of the methods `la.eig`, `la.eigh`, and `la.svd`.

Here, it is highly recommended you resort to python's `timeit` module. Examples for its proper use can be found in

> C. Bauckhage, "NumPy / SciPy Recipes for Image Processing: Avoiding for Loops over Pixel Coordinates", technical report, 2018
> (available at researchgate.net)

What do you observe ? Do your result so far suggest any preference for any of the methods considered up to this point ?

task 1.3: the QR algorithm
Given matrix $\boldsymbol{C}$ as computed above, implement the following algorithm

```
$\boldsymbol{C}_0 = \boldsymbol{C}$

for $t = 1, \ldots, 10$
    compute the QR decomposition of $\boldsymbol{C}_{t-1}$ to obtain matrices $\boldsymbol{Q}_t$ and $\boldsymbol{R}_t$

    given matrices $\boldsymbol{Q}_t$ and $\boldsymbol{R}_t$, compute

        $\boldsymbol{C}_t = \boldsymbol{R}_t \boldsymbol{Q}_t$
```

Run your implementation and have a look at the diagonal entries of the resulting matrix $\boldsymbol{C}_{10}$; compare them against the spectra you computed above; what do you observe ?

Note that

$$\boldsymbol{C}_t = \boldsymbol{R}_t \boldsymbol{Q}_t = \boldsymbol{Q}_t^{-1} \boldsymbol{Q}_t \boldsymbol{R}_t \boldsymbol{Q}_t = \boldsymbol{Q}_t^{-1} \boldsymbol{C}_{t-1} \boldsymbol{Q}_t = \boldsymbol{Q}_t^{T} \boldsymbol{C}_{t-1} \boldsymbol{Q}_t$$

Can you use this insight to explain the result you get form running the QR algorithm ?

If you want to impress your professor, perform run-time measurements for your implementation of the QR algorithm as well.

task 1.4: fastmap
Implement the fastmap algorithm which we discussed in the lecture and run it on matrix $\boldsymbol{X}$.

Compare the approximative principal component you obtain from this approach the the left singular vectors you obtain from applying the SVD to matrix $\boldsymbol{X}$.

A good idea for doing this is to visualize / plot them as tiny images; figure out how to create a plot that shows not just a single tiny image but several, say 25, at the same time.

task 1.5: finding maximally different images
So far, you only had to do what the problem specifications asked you to do; in this task, you need to get creative:

Working with the *unnormalized* matrix $\boldsymbol{X}$, think of / invent an algorithm that selects $k > 2$ out of its n columns such that the selected data vectors are as far apart as possible. Mathematically, this problem is expressed as follows: given $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$, solve

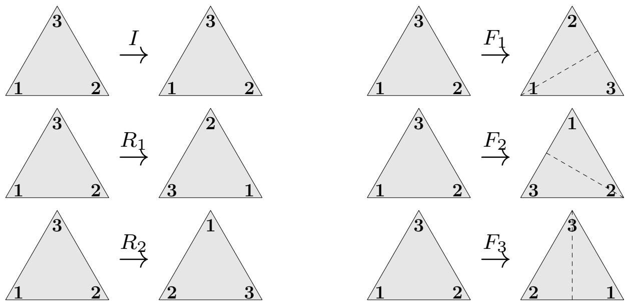
$$
\begin{aligned}
S^* = \underset{S \subset X}{\operatorname{argmax}} & \sum_{\boldsymbol{x}_i \in S} \sum_{\boldsymbol{x}_j \in S} \left\| \boldsymbol{x}_i - \boldsymbol{x}_j \right\|^2 \\
\text{s.t.} \quad & |S| = k
\end{aligned}
$$

Test your algorithm for several choices of k, say $k \in \{25, 50, 100\}$; visualize the extracted columns in terms of tiny images

NOTE: this problem is actually way more difficult than it may appear at first sight; in fact, it is NP-complete and an *efficient* algorithm that is *guaranteed* to find the optimal S^* for very large sets X and arbitrary choices of k remains elusive to this date.

bonus task: the dihedral group D_3
This task is not mandatory but intended for those who like to puzzle and put new found knowledge to work. (If you decide to give this task a try, then really try to solve it yourself . . . those who do not like to- or cannot think for themselves may of course just go to wikipedia and look up the answer to the questions below . . . but what would you gain from that?)

The following are the six symmetries (three rotations and three reflections) of a unilateral triangle:



Using the symbols / notation introduced in this figure, the dihedral group D_3 is given by

$$D_3 = \big(G, \circ\big)$$

where

$$G = \big\{I, R_1, R_2, F_1, F_2, F_3\big\}$$

and $\circ$ denotes the composition operator.

Can you set up the Cayley table for this group ? If so, does you result suggest that D_3 is an Abelian group or not ?

general hints and remarks

- Send all your solutions (code, plots, slides) in a ZIP archive to

  bauckhag@bit.uni-bonn.de

- The goal of this project is to provide a gentle introduction to scientific python. There are numerous resources on the web related to python programming. Numpy and Scipy are more or less well documented and Matplotlib, too, comes with tons of tutorials. Play with the code that is provided. Most of the above tasks are trivial to solve, just look around for ideas as to how it can be done.
- **note:** if you insist on using a language other than python, you have to figure out elementary functions/toolboxes in these languages by yourself. **Implementations in MATLAB will not be accepted.**
- Remember that you have to complete all practical projects (and the tasks therein) to be eligible to the written exam at the end of the semester. Your grades (and credits) for this course will be decided based on the exam only, but –once again– you have to succeed in the projects to get there.
- Not handing in a solution implies failing the course.
- Your project work needs to be *satisfactory* to count as a success. Your code and results will be checked and your presentation needs to be convincing.
- If your solutions meets the above requirements and you can demonstrate that they work in practice, it is a *satisfactory* solution.
- A *good* to *very good* solution requires additional efforts especially w.r.t. to elegance and readability of your code. If your code is neither commented nor well structured, your solution is not good! A very good solution requires additional efforts towards the quality of your project presentation in the colloquium. Your presentation should be well timed, consistent, and convincing. Striving for very good solutions should always be your goal!