

## COMPUTATIONAL PHYSICS



### Fourth assignment WS 2022/2023

**Deadline: 22nd of January 2023**

**Final date for the mini-exam: 23rd/24th of January**

The aim of this assignment is to use the knowledge from the previous assignments to solve the exercises using as little guidance as possible. You should complete 2 of the 3 exercises for full points.

**Note:** The assignments contain questions and tasks, labeled (a), (b), etc. For some of these tasks, you are supposed to create and hand in one of the following:

-  a script or a part of a script written in the computer language of your choice;
-  a figure with legends and axis titles.

Other questions primarily serve to guide you through the exercises; you do not need to submit the answers in writing. However, you may be asked these or similar questions during the assessment discussions.

**Note:** Many tasks in the exercises can be performed using an existing library or toolbox. In line with the aim of the course, however, programming your own analysis tools is encouraged. If libraries or toolboxes are used, you are expected to be able to explain in detail how these algorithms work.

**Note:** When collaborating in groups of up to 3 students, you are allowed to hand in identical code. Please list all collaborators, for example in the header.


**Note:** The scripts in the programming exercises should be considered to be intended for general use, and a corresponding coding style will be appreciated. For example, using input parameters as variables, clear presentation of input and output, naming of variables and/or comprehensive comments in the script are among the grading criteria.

# 1 Singular value decomposition of a drawing

We consider two drawings and a painting from Leonardo da Vinci, provided in digital format as `Codice_Atlantico.bmp`, `Vitruvian.bmp` and `Lady_with_Ermine.bmp`.<sup>1</sup> Load one of the images using the Matlab function `imread` or some equivalent function in the programming language of your choice.<sup>2</sup> The input consists of three matrices containing the intensities of the three colors red, green and blue (RGB mode).

## Power method

The aim of this task is to calculate the largest eigenvalues and vectors of the matrix  $B_\nu = A_\nu^\dagger A_\nu$  in order to find a representation of the image  $A_\nu$  for all colors  $\nu = \{r, g, b\}$  by storing only a few vectors instead of the whole matrix (image compression). To do so, we first develop a function that calculates the eigenvectors corresponding to the eigenvalues with the highest absolute values, for which we will use the power method.

- (a)  Implement a function that has the input arguments `B`, `tol`, `maxit`, `x_0` and output arguments `eigenvalue`, `eigenvector` and `residuuum`. The function should perform the von Mises iterations with the optional starting vector  $x_0$ . If  $x_0$  has not been passed, select a random starting vector. The residuum after  $p$  iterations is calculated using the Rayleigh quotient

$$r^{(p)} = \left| \frac{\mathbf{x}^{\dagger(p)} A \mathbf{x}^{(p)}}{\mathbf{x}^{\dagger(p)} \mathbf{x}^{(p)}} - \frac{\mathbf{x}^{\dagger(p-1)} A \mathbf{x}^{(p-1)}}{\mathbf{x}^{\dagger(p-1)} \mathbf{x}^{(p-1)}} \right| = |\lambda^{(p)} - \lambda^{(p-1)}|. \quad (1)$$

The returned eigenvector should be normalized.

---

<sup>1</sup>The used images are in the public domain. The files have been taken from [www.wikipedia.org](http://www.wikipedia.org).

<sup>2</sup>To load `bmp` files in python use the following code: `from PIL import Image, im = Image.open("Codice_Atlantico.bmp") or im = Image.open("Vitruvian.bmp"), pix = im.load(), print im.size, print pix[x,y], im.save("file_out.jpg")`. The images are also provided in `txt` format.

## Power method<sup>a</sup>

We assume that the matrix  $A$  has a dominant eigenvalue with corresponding dominant eigenvectors. Then we choose an initial approximation  $\mathbf{x}^{(0)}$  of one of the dominant eigenvectors of  $A$ . This initial approximation must be a nonzero real-valued vector. Finally, we form the sequence given by

$$\mathbf{x}^{(p)} = A\mathbf{x}^{(p-1)} \quad (2)$$

for  $p = 1 \dots n$ . For large values of  $n$ , this sequence converges to the dominant eigenvector of  $A$ . The corresponding eigenvalue can be found using the Rayleigh quotient

$$\lambda = \frac{\mathbf{x}^{\dagger(n)} A \mathbf{x}^{(n)}}{\mathbf{x}^{\dagger(n)} \mathbf{x}^{(n)}} \quad (3)$$

Details can be found in the lecture notes.

---

<sup>a</sup>Also known as von Mises iteration. Richard Edler von Mises (1883 - 1953) was an Austrian mathematician.

## Deflation


After having calculated the largest eigenvalue and its eigenvector, the aim of deflation is to get the second largest eigenvector. Deflation transforms the  $n$ -th eigenvalue to zero.

### Deflation

Let  $B$  be a normal  $n \times n$  matrix with eigenvalues  $|\lambda_1| < |\lambda_2| < \dots < |\lambda_n|$  and associated eigenvectors  $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n$ . Then the matrix

$$C = B - \lambda_n \mathbf{v}_n \mathbf{v}_n^{\dagger} \quad (4)$$

has eigenvalues  $0, \lambda_1, \dots, \lambda_{n-1}$  with associated eigenvectors  $\mathbf{v}_n, \mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_{n-1}$ .

- (b)  Write a program to calculate the  $n$  largest eigenvalues and associated eigenvectors using repeated von Mises iteration and deflation. Calculate the  $n = 10$  largest eigenvalues and eigenvectors of the matrix  $B_\nu = A_\nu^\dagger A_\nu$  and plot the residuum depending on the iteration number of the first three calculations in a semilogarithmic plot. You can use the value  $10^{-5}$  for the residuum as the stop criterion.

## Singular value decomposition

Now that we are able to calculate the eigenvalues and eigenvectors of a non-degenerate normal matrix we approach our final goal: singular value decomposition of Leonardo's drawing.

### Singular Value Decomposition (SVD)

Every square<sup>a</sup> matrix  $A$  can be represented as

$$A = U\Lambda V^\dagger, \quad (5)$$

with  $U$  and  $V$  being unitary matrices and  $\Lambda$  being a diagonal matrix containing the so-called singular values, which are the square roots of the non-zero eigenvalues of the matrix  $A^\dagger A$ . The three matrices can be calculated in the following way:


- Matrix  $V$  consists of the orthonormal<sup>b</sup> eigenvectors of the matrix  $B = A^\dagger A$ .
- The eigenvalues of  $B$  are denoted by the diagonal matrix  $D$ :  $BV = VD$ . For the diagonal matrix  $\Lambda$  the following relation holds:  $\Lambda = D^{\frac{1}{2}}$ .
- The matrix  $U$  can be calculated by  $U = AVD^{-\frac{1}{2}}$ .

By taking only the  $n$  largest eigenvalues of  $B$  you can calculate the matrices  $V_n$ ,  $\Lambda_n$  and  $U_n$ . The final  $n$ -SVD results is

$$A_n = U_n \Lambda_n V_n^\dagger. \quad (6)$$

<sup>a</sup>Note that a more general description for rectangular matrices exists.

<sup>b</sup>Check for orthonormality - if necessary, orthonormalization can be performed using the Gram-Schmidt or modified Gram-Schmidt algorithms.

- (c)  Calculate a  $n$ -SVD representation of Leonardo's drawing for all three color channels for  $n = \{1, 5, 10, 20\}$ . Save the approximated image as `Leonardo_n.jpg` (or another graphics format of your choice) using the corresponding number for the value of  $n$ . If you use the Matlab function `imwrite`, cast the three-dimensional matrix to the variable type `uint8` or divide the matrix by 255.<sup>3</sup>
- (d) **Bonus-Task:** Increase the value for  $n$  until the resulting  $n$ -SVD representation of Leonardo's drawing is reasonably sharp. How do the values of  $n$  compare to the initial drawing.

### Gram-Schmidt orthonormalization

The Gram-Schmidt algorithm can be used to project a set of vectors onto an orthonormal basis. Starting with a set of vectors  $\mathbf{u}$ , the orthogonal vectors  $\mathbf{v}$  are calculated from

$$\mathbf{v}_k = \mathbf{u}_k - \sum_{j=0}^{k-1} \frac{\mathbf{u}_k \cdot \mathbf{v}_j}{\mathbf{v}_j \cdot \mathbf{v}_j} \mathbf{v}_j, \quad (7)$$

which can then be normalized to create orthonormal vectors. For better numerical stability, a modified Gram-Schmidt method can be used, which calculates the orthogonal vector in steps instead of in a single sum,

$$\begin{aligned} \mathbf{v}_k^{(1)} &= \mathbf{u}_k - \frac{\mathbf{u}_k \cdot \mathbf{v}_1}{\mathbf{v}_1 \cdot \mathbf{v}_1} \mathbf{v}_1 \\ \mathbf{v}_k^{(2)} &= \mathbf{v}_k^{(1)} - \frac{\mathbf{v}_k^{(1)} \cdot \mathbf{v}_2}{\mathbf{v}_2 \cdot \mathbf{v}_2} \mathbf{v}_2 \\ &\vdots \\ \mathbf{v}_k &\equiv \mathbf{v}_k^{(k-1)} = \mathbf{v}_k^{(k-2)} - \frac{\mathbf{v}_k^{(k-2)} \cdot \mathbf{v}_{k-1}}{\mathbf{v}_{k-1} \cdot \mathbf{v}_{k-1}} \mathbf{v}_{k-1}. \end{aligned} \quad (8)$$





The final equation can again be normalized.

---

<sup>3</sup>Useful Matlab functions are `imwrite` and `uint8`.

## 2 Simulation of the solar system

Use a suitable numerical integration scheme (for example: a Runge-Kutta scheme, Crank-Nicolson, Verlet) to perform following tasks. You are allowed to use scripts written for previous exercises, but make sure you critically reflect on their suitability for this exercise.





- (a)  Determine the orbits of the following systems
- Mercury to Mars + sun
  - Jupiter to Neptune + sun
  - Mercury to Neptune + sun
- (b)  Create the following plots
- orbitals of the planets and the sun
  - the distance  $r(t)$  of the planets (+ sun) to the center of mass
- (c)  Check for
- conservation of energy
  - conservation of angular momentum
  - agreement of your results with the orbital periods from literature
- (d)  **Bonus-Task** Create an animation of the solar system

**Hints:** The initial conditions can be taken from [`https://ssd.jpl.nasa.gov/horizons/app.html#`](https://ssd.jpl.nasa.gov/horizons/app.html#/).

## 3 Abbe theory of image formation

In this task we want to deal with the Fourier transformation one more time. For detailed information on the underlying math we refer the reader to the literature. The important part for our considerations is, that the intensity distribution and the corresponding diffraction pattern are associated through a Fourier transformation. For some help the reader can take a look on the following module [`https://diffractio.readthedocs.io/en/latest/examples.html`](https://diffractio.readthedocs.io/en/latest/examples.html). The reader is free to use the built-in FFT and IFFT functions.

- (a) Write a function `optical_mask`, which is able to create the intensity distribution of a given mask.

- (b)  Write a function `diffraction_pattern`, which calculates the diffraction pattern of the output from your `optical_mask` function.
- (c)  Write a function `diffraction_mask`, which is able to filter different orders of your diffraction pattern.
- (d)  Write a function `intensity_distribution`, which transforms your diffraction pattern back into the initial intensity distribution.
- (e)  Use your functions to plot the following images
- The intensity distribution from your `optical_mask` function
  - The diffraction pattern from your `diffraction_pattern` function
  - The diffraction pattern after applying your `diffraction_mask` to it
  - The intensity distribution you get from the modified diffraction pattern.

Do so for at least three different optical masks and 5 different diffraction masks. Possible diffraction masks could be ones, removing the highest-order diffraction peaks, the lowest-order diffraction peak, even-ordered diffraction peaks and so on.

- (f) **Bonus-Task:** Modify one of your optical masks to include some dust on it. Try to “clean” the dirty mask by means of removing the contribution of this dust to the diffraction pattern.

**Hint:** Feel free to code your function in a way that it generates the masks from an image you provide (like the ones in TeachCenter). **Feel free to express your Christmas joy within your masks.**