R. Bergmann, E. Çokaj, M. Ludvigsen

Submission Deadline:
Wednesday 28. September, 12:00 (noon)

Project #1

12. September 2022

Notes

This project is an individual project and should be solved that way. You can discuss and work on the problems with other students, but you should write the solutions *in your own words and code.*
Your answer should be contained in a Jupyter notebook that will be uploaded to Inspera.
There is a notebook `project1_supplementary.ipynb` that contains some code that you can and should use for your own answers. The project also comes with a `.zip`-file containing a few `.npy`-files, which is a format for storing numpy arrays, that you will need for the final task of the project. We recommend checking that all the code in `project1_supplementary.ipynb` runs on your machines, and then copy the necessary code over to your own `.ipynb` file which you will hand in. For the rest of this text, we refer to `project1_supplementary.ipynb` as the "handed out code".
All code —also the tests if necessary— should be in individual cells that can just be run (as soon as the necessary functions are defined). Functions should only be used in cells *after* their definition, such that an evaluation in order of the notebook runs without errors. Try to always explain what a chunk of code does *before* the chunk, and discuss the results *after* the chunk.

The project is obligatory and counts 10% on the final grade.

Deadline: Wednesday 28. September, 12:00 (noon)

Introduction

In this project we will work with solving underdetermined linear systems

$$A\boldsymbol{x} = \boldsymbol{y},$$

where $A \in \mathbb{R}^{m\times n}$, $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{y} \in \mathbb{R}^m$, and where $m < n$. Thus, we have more variables in $\boldsymbol{x}$ to determine (n) than we have equations (m); or informally: A is a "wide" matrix. These equations have no or infinitely many solutions, which means the problem is *ill-posed*.

These problem still appear in many applications; we will consider a so-called single channel source separation problem in this project. We will first introduce the necessary background, before stating the tasks. For some supplementary source code, see the accompanying `project1.ipynb` Jupyter notebook. We first start with a few theoretical preliminaries necessary for the tasks. Some of these you already saw in the lecture or some exercises, but we repeat them to make this project self-contained.

Theoretical Background

Positive Definite matrices. A square matrix $A \in \mathbb{R}^{n\times n}$ is *positive definite* if it satisfies

$$\boldsymbol{x}^\mathsf{T} A\boldsymbol{x} > 0, \qquad \text{for all } \boldsymbol{0} \neq \boldsymbol{x} \in \mathbb{R}^n.$$

Note that for $n = 1$ this simplifies to require that $ax^2 > 0$ for $x \neq 0$, i. e. $a > 0$.
Usually the matrices we consider are symmetric as well, i. e. $A^\mathsf{T} = A$, which we denote in short by s.p.d., or even just by SPD.
For symmetric matrices, all eigenvalues are real. You can for example easily convince yourself that the identity matrix I is positive definite. In the following we also write $I_n \in \mathbb{R}^{n\times n}$ to denote the $n \times n$ identity matrix.
A square matrix $A \in \mathbb{R}^{n\times n}$ is *positive semi-definite* if it satisfies

$$\boldsymbol{x}^\mathsf{T} A\boldsymbol{x} \geq 0, \qquad \text{for all } \boldsymbol{x} \in \mathbb{R}^n.$$

Singular Value Decomposition. A square, diagonalizable matrix $A \in \mathbb{R}^{n\times n}$ admits an eigenvalue decomposition

$$A = Q\Lambda Q^{-1},$$

where Q contains (normalised) eigenvectors of A, and $\Lambda = \operatorname{diag}(\lambda_1, \ldots, \lambda_n)$ is a diagonal matrix containing the eigenvalues of A. Fixing the order of the eigenvalues, for example in descending order, this decomposition is unique.
If a square matrix A is normal (i. e. $A^\mathsf{T}A = AA^\mathsf{T}$) we obtain the spectral theorem, that is the matrix Q is orthogonal ($Q^\mathsf{T} = Q^{-1}$), and the decomposition can be written as

$$A = Q\Lambda Q^\mathsf{T}.$$

Note that any symmetric matrix $A^\mathsf{T} = A$ is normal.

For non-square matrices one can generalise this to the so-called *Singular Value Decomposition* (SVD) Let $A \in \mathbb{R}^{m\times n}$, then there exist orthogonal matrices $U \in \mathbb{R}^{m\times m}$, $V \in \mathbb{R}^{n\times n}$ (that is $U^\mathsf{T}U = I_m$ and $V^\mathsf{T}V = I_n$) and there exists a rectangular matrix $\Sigma \in \mathbb{R}^{m\times n}$ only containing nonzero values $\sigma_1, \ldots, \sigma_m$ on the diagonal (remember, that $m < n$), such that

$$A = U\Sigma V^\mathsf{T},$$

The values $\sigma_1, \ldots, \sigma_m$ are called *singular values* of A.

To obtain U and V we can diagonalise $AA^\mathsf{T} \in \mathbb{R}^{m\times m}$ and $A^\mathsf{T}A \in \mathbb{R}^{n\times n}$, respectively, namely

$$AA^\mathsf{T} = U\Sigma\Sigma^\mathsf{T}U^\mathsf{T}, \qquad A^\mathsf{T}A = V\Sigma^\mathsf{T}\Sigma V^\mathsf{T}.$$

This immediately implies that for a symmetric positive (semi-)definite matrix A, the SVD is just the eigenvalue decomposition with $\Lambda = \Sigma^2$, that is, the singular values are the square roots of the eigenvalues.
We can also use the singular values to determine the rank of the matrix. Similarly to eigenvalues for a square matrix, the rank of a matrix is equal to the number of non-zero singular values.
The details of numerical algorithms for calculating the SVD are not of interest us, but the point is that calculating the SVD is *expensive*. We will use `np.linalg.svd` for calculating the SVD, which returns Σ as a vector, not a diagonal matrix. You should most likely also pass the keyword `full_matrices = False` to calculate the so-called *truncated SVD*, which avoids calculating/storing redundant values.

Minimal norm solutions. Now, let us start looking at how to find a meaningful solution to the underdetermined system

$$Ax = y$$

For the case of infinitely many solutions, we need additional assumptions on $\boldsymbol{x}$ in order to obtain unique solutions.
A property that usually makes sense to ask for is the solution with minimal norm. This can be expressed as a *constraint optimization problem*, which is actually a little bit beyond our curriculum, but we try to stick to a suitable intuitive level here.
We define the minimal norm solution as

$$x_{\min} = \operatorname*{arg\,min}_{\boldsymbol{x}\in\mathbb{R}^n} \boldsymbol{x}^\mathsf{T}\boldsymbol{x}, \qquad \text{such that } A\boldsymbol{x} = \boldsymbol{y}. \tag{1}$$

Remember that the arg min here refers to the fact that we are not interested in the minimal function value (i. e. $\|\boldsymbol{x}\|^2$) but in the point $\boldsymbol{x}_{\min}$ where the minimum is attained.
For matrices $A \in \mathbb{R}^{m\times n}$ of full rank ($\operatorname{rank}(A) = m$ since $m < n$) the solution to this problem is even unique.

Example. For $m = 1$, $n = 2$ we could consider $A = \begin{pmatrix}2 & 1\end{pmatrix}$ and $y = 1$. That is

$$\operatorname*{arg\,min}_{\boldsymbol{x}\in\mathbb{R}^2} (x_1 + x_2)^2, \qquad \text{such that } 2x_1 + x_2 = 1,$$

which asks for the point closest (in Euclidean norm) to zero on the line $x_2 = 1 - 2x_1$. In this relatively simple case we can solve the problem by inserting the constraint into the problem, and obtain

$$\operatorname*{arg\,min}_{\boldsymbol{x}\in\mathbb{R}^2 : x_2 = 1-2x_1} \; (x_1 + (1-2x_1))^2 = \begin{pmatrix}1 & -1\end{pmatrix}^\mathsf{T}.$$

In the more general case we can use the so-called *Lagrange function*, which in our case for (1) reads

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) = \boldsymbol{x}^\mathsf{T} x + \boldsymbol{\lambda}^\mathsf{T}(A\boldsymbol{x} - \boldsymbol{y}), \tag{2}$$

where $\boldsymbol{\lambda} \in \mathbb{R}^n$ are the so-called *Lagrange multipliers.*

The idea is that if we minimize $\mathcal{L}$ with respect to $\boldsymbol{x}$ and *simultaneously maximize* with respect to $\boldsymbol{\lambda}$, then this requires $A\boldsymbol{x} - \boldsymbol{y}$ to become the zero vector. Hence we are looking for a so-called *saddle point of* $\mathcal{L}$ to fulfill both the optimality in norm as well as the constraint.

A solution to this optimization problem satisfies (1)

$$\nabla_x \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) = 0 \quad \text{ and } \quad \nabla_\lambda \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) = 0, \tag{3}$$

so we obtain in our case again two linear systems to solve. One can even compute the solution from this, see Task 1.c), which is

$$\boldsymbol{x}_{\min} = A^\mathsf{T}(AA^\mathsf{T})^{-1}\boldsymbol{y} \tag{4}$$

Computing $\boldsymbol{x}_{\min}$ is the main part of task 1 in this project.
While in general we can just use `np.linalg.inv` this approach is often infeasible for large problems. A second way is of course to use the generic `np.linalg.solve`, but indeed we know much more about our problem due to the specific structure of $\boldsymbol{x}_{\min}$, for example that the matrix AA^T is symmetric and positive-semidefinite, see also Task 1.b). However, don't be surprised if the code you produce is slower than the NumPy methods. These are well written programs that clever people have spent tens of years optimizing.
Using these properties we can spare both time and memory in the computation, since in most cases, it is not advisable to actually compute the inverse of a matrix, unless of course the inverse is easy to find like for diagonal or orthogonal matrices.
We will consider a method based on the QR decomposition in Task 1.

Task 1

1.a) Let $m = 2$ and $n = 3$, that is, we consider linear systems of the form $A\boldsymbol{x} = \boldsymbol{y}$ with $A \in \mathbb{R}^{2\times 3}$, $\boldsymbol{x} \in \mathbb{R}^3$, and $\boldsymbol{y} \in \mathbb{R}^2$.
Give two examples of A and $\boldsymbol{y}$ such that

(1) the linear system does not have a solution,

(2) the linear system has infinitely many solutions.

1.b) Let $A \in \mathbb{R}^{m\times n}$, $m < n$. Show that both $A^\mathsf{T}A$ and AA^T are symmetric, and that both are positive semi-definite.
What can you say about the invertibility of these two matrices if A has full rank, that is $\operatorname{rank}(A) = m$?

Hint. Use the identity $(AB)^\mathsf{T} = B^\mathsf{T}A^\mathsf{T}$, the fact that you can write $\boldsymbol{x}^\mathsf{T}\boldsymbol{x} = \|\boldsymbol{x}\|^2$ as well as properties of the norm.

1.c) Compute the two gradients $\nabla_{\boldsymbol{x}}\mathcal{L}(\boldsymbol{x},\boldsymbol{\lambda})$ and $\nabla_{\boldsymbol{\lambda}}\mathcal{L}(\boldsymbol{x},\boldsymbol{\lambda})$ from (3) of the Lagrange function $\mathcal{L}(\boldsymbol{x},\boldsymbol{\lambda})$ from (2). Setting both gradients to zero, you can plug the first equation into the second. Use this to show that the solution to (1) is given by

$$\boldsymbol{x}_{\min} = A^\mathsf{T}(AA^\mathsf{T})^{-1}\boldsymbol{y}$$

1.d) We further want to use the QR decomposition to solve (4) Show that $\boldsymbol{x}_{\min} = QR^{-\mathsf{T}}\boldsymbol{y}$

using the reduced QR decomposition of $A^\mathsf{T} = QR$, where $Q \in \mathbb{R}^{n\times m}$ is orthogonal and $R \in \mathbb{R}^{m\times m}$ is upper triangular. [1] Note that for R we have $R^{-\mathsf{T}} = (R^\mathsf{T})^{-1} = (R^{-1})^\mathsf{T}$. Implement a function that calculates the QR decomposition using one of the methods explained in lectures. Gram-Schmidt (or even better, Modified Gram-Schmidt[2]), is most likely the easiest one to implement.

1.e) Explain how to calculate $\boldsymbol{x}_{\min}$ without inverting R^T. Implement a function that calculates $\boldsymbol{x}_{\min}$ given the matrices Q, R and the left hand side $\boldsymbol{y}$. You can use the forward/backward substitution methods from Exercise 1, or even `scipy.linalg.solve_triangular`. You can for example test your algorithm on the system you proposed in task 1.a).

1.f) We now want to test our method numerically, and in particular compare against NumPy implementations. In the handed out code you will find a function `generate_A` that generates $m \times n$ matrices A. Given a generated A, we want to try to solve the problem for 100 hundred different $\boldsymbol{y}$. In the handed out code is a also a chunk that generates 100 $\boldsymbol{x}$ which are componentwise i.i.d standard normal distributed using `np.random.normal(0.0,1.0,(100,n))`, which creates a matrix containing 100 vectors of size n. We can then create the left hand sides by calculating $\boldsymbol{y} = A\boldsymbol{x}$. In the handed out code these are stored in the arrays `X` and `Y`.

Compare the following two implementation approaches

1) `np.linalg.solve`, which consists of precalculating AA^T, then for each $\boldsymbol{x}$ solve

$$AA^\mathsf{T}\boldsymbol{z} = \boldsymbol{y}, \qquad \boldsymbol{x}_{\min} = A^\mathsf{T}\boldsymbol{z}.$$

2) The QR method which you implemented in the previous tasks.

Using for example `%%timeit`, measure how much time each method takes to precalculate the necessary matrices/decompositions, and independently of this how much time it takes to calculate $\boldsymbol{x}_{\min}$ for the 100 different $\boldsymbol{y}$.
Which method performs the best timewise?
Also, for at least one of the test examples, calculate $\|\boldsymbol{x}_{\min} - \boldsymbol{x}_{\text{true}}\|$, where $\boldsymbol{x}_{\text{true}}$ is the

[1] Remember a thin $n \times m$ matrix A with $n > m$ has a QR decomposition $A = QR = Q\begin{bmatrix}\tilde{R}\\0\end{bmatrix} = \tilde{Q}\tilde{R}$, where $Q \in \mathbb{R}^{n\times n}, R \in \mathbb{R}^{n\times m}$. What we want to calculate in this task is the reduced (or truncated) $\tilde{Q}, \tilde{R}$, as the full matrices Q and R contain redundant information.

[2] https://en.wikipedia.org/wiki/Gram\T1\textendashSchmidt_process

"true" solution, that is the one you generated and are stored in `X`.
Is there any difference in the numerical accuracy of the different methods?
Are you able to recover a solution in a meaningful way?

Tikhonov regularisation. Instead of trying to solve $A\boldsymbol{x} = \boldsymbol{y}$ by using constraints, we can also consider the least squares solution

$$\operatorname*{arg\,min}_{\boldsymbol{x}\in\mathbb{R}^n} \lVert A\boldsymbol{x} - \boldsymbol{y}\rVert^2.$$

Note that the minimum is not necessarily unique, and there can be infinitely many solutions. Furthermore, if there is no solution for the linear system of equations, we still might get solutions here, maybe even a unique one.
By looking at the gradient one can derive that a solution $\boldsymbol{x}_{\mathrm{ls}}$ satisfies $A^{\mathsf{T}}A\boldsymbol{x}_{\mathrm{ls}} = A^{\mathsf{T}}\boldsymbol{y}$. While this is (again) a linear system, we already learned that this does not yet yield a unique solution for underdetermined systems.
Instead of constraining our solution, we could alternatively introduce a weight $\mu > 0$ to “balance” between a solution of $A\boldsymbol{x} = \boldsymbol{y}$ and a small norm of $\boldsymbol{x}$, i. e. we obtain the so-called *Tikhonov regularisation* or Ridge regression (commonly used in statistics)

$$\boldsymbol{x}_{\mathrm{r}} = \operatorname*{arg\,min}_{\boldsymbol{x}\in\mathbb{R}^n} \lVert A\boldsymbol{x} - \boldsymbol{y}\rVert^2 + \mu\lVert\boldsymbol{x}\rVert^2. \tag{5}$$

For ease of notation let’s introduce $T(\boldsymbol{x}) = \lVert A\boldsymbol{x} - \boldsymbol{y}\rVert^2 + \mu\lVert\boldsymbol{x}\rVert^2$.

Task 2

2.a) A solution of the Tikhonov regularisation (5) satisfies $\nabla_{\boldsymbol{x}}T(\boldsymbol{x}) = 0$. Use this to show that a solution $\boldsymbol{x}_{\mathrm{r}}$ of (5) satisfies $(A^{\mathsf{T}}A + \mu I)\boldsymbol{x}_{\mathrm{r}} = A^{\mathsf{T}}\boldsymbol{y}$.
Show that $A^{\mathsf{T}}A + \mu I$ is SPD and therefore invertible.

2.b) Using the SVD $A = U\Sigma V^{\mathsf{T}}$, show that $\boldsymbol{x}_{\mathrm{r}}$ satisfies $\boldsymbol{x}_{\mathrm{r}} = VDU^{\mathsf{T}}\boldsymbol{y}$ where D is a diagonal matrix depending on the singular values Σ and the parameter μ.

Hint. For two diagonal matrices $D_1, D_2 \in \mathbb{R}^{n\times n}$ we have $VD_1V^{\mathsf{T}} + VD_2V^{\mathsf{T}} = V(D_1 + D_2)V^{\mathsf{T}}$.

2.c) Write a function `Tikhonov_solveSVD(U,S,Vt,mu)` that computes $\boldsymbol{x}_{\mathrm{r}}$ based on A and μ using the SVD approach from 2.b). Note that the singular values Σ should be stored in a vector, not an actual diagonal matrix.

2.d) We now want to do another numerical experiment. This time we are not interested in solving for many different left hand sides $\boldsymbol{y}$, but rather many different regularisation parameters μ. We use the same function as we did in Task 1 for generating a matrix A. Generate 100 logarithmically spaced values of $\mu \in [10^{-6}, 10^{6}]$ using for example `np.logspace(5,-6,num = 100)` (you should iterate starting with the largest values of μ), and calculate $\boldsymbol{x}_r$ for each of these μ. We denote the solution for a given μ as $\boldsymbol{x}_r(\mu)$. We want to test two approaches:

1) Using the SVD. Remember the SVD only has to be calculated once, and we can then find $\boldsymbol{x}_r$ for many different μ.

2) Using `np.linalg.solve` and precalculating $A^\mathsf{T}A$ and $A^\mathsf{T}\boldsymbol{y}$. When you do this you need to select a much coarser grid for the values of μ, for example 5 to 10 values. The reason is that this approach is painfully slow. You might also see some numerical instability for small μ.

For each value of μ, store the error $\|\boldsymbol{x}_\mathrm{r}(\mu) - \boldsymbol{x}_\mathrm{true}\|$, where $\boldsymbol{x}_\mathrm{true}$ is the "true" solution provided in the code as well and plot the errors as a function of μ. You can also plot $\boldsymbol{x}_\mathrm{min}$ as a horizontal line for comparison. You should also time how long it takes to do the numerical experiment for the different methods, but make sure you don't include the timing of the plotting. Which method is the most computationally efficient? What μ gives the best reconstruction of $\boldsymbol{x}_\mathrm{true}$?

Single Channel Source Separation. We now move on to a more practical problem. As an application we consider the so-called single channel source separation problem. Assume we know that our measured signal $\boldsymbol{y} \in \mathbb{R}^n$ consists of two separate channels, $\boldsymbol{u} \in \mathbb{R}^n$ and $\boldsymbol{v} \in \mathbb{R}^n$, i. e.

$$\boldsymbol{u} + \boldsymbol{v} = A\boldsymbol{x} = \begin{pmatrix} I & I \end{pmatrix} \begin{pmatrix} \boldsymbol{u} \\ \boldsymbol{v} \end{pmatrix} = \boldsymbol{y}$$

In this case our unknown $\boldsymbol{x} \in \mathbb{R}^{2n}$ consist of the two stacked channel sources $\boldsymbol{u}, \boldsymbol{v}$ and we have $m = 2n$.
A real-life application of this is the separation of audio sources. For example, if $\boldsymbol{u}$ is some vocal track and $\boldsymbol{v}$ some instrumentals, we would like to separate those from a common (single) channel recording $\boldsymbol{y}$. Another application is to split images, which is what we will do in the next task.

We will solve the single channel source separation for images from the MNIST dataset. In

there images are represented as matrices with values in $[0, 1]$ stored in a matrix. For solving the source separation we reshape the image (matrix) to a vector.
The images are of size 28×28 so the vector has $n = 784$ elements. Throughout the rest of the project, we will assume that $\boldsymbol{u}$ is an image of a digit 0 and $\boldsymbol{v}$ is an image of a digit 1. Example images are shown in Figure 1.

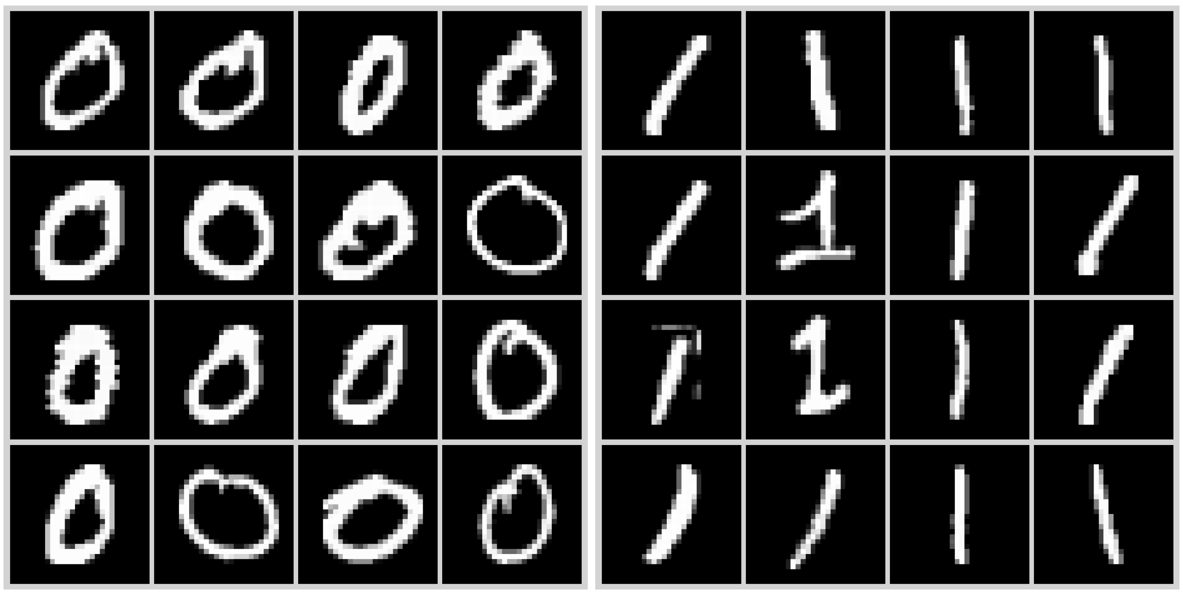

Figure 1: To the left are 16 zero-digits and to the right are 16 one-digits from the MNIST dataset.

Figure 2 shows an example of a mix of two images. Our task is now: given this mixed image,

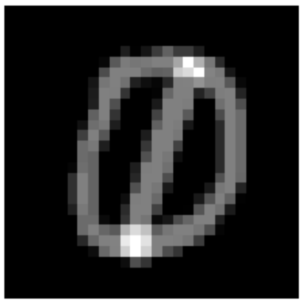

Figure 2: An example of a mixed image $\boldsymbol{y} = \boldsymbol{u} + \boldsymbol{v}$.

can we recover the individual image of a zero-digit and an individual image of a one-digit?

Solving the single channel source separation using learnt norm. As we mentioned in the introduction to this project, we need additional assumptions on $\boldsymbol{u}$ and $\boldsymbol{v}$ in order to have a chance at solving the problem. We will assume that these two signal types can be described by/are close to two different linear subspaces

$$Z_0 = \operatorname{span}\{\boldsymbol{w}_1^{(0)}, \dots, \boldsymbol{w}_d^{(0)}\} \qquad Z_1 = \operatorname{span}\{\boldsymbol{w}_1^{(1)}, \dots, \boldsymbol{w}_d^{(1)}\},$$

where d is the number of basis vectors. We assume that the basis vectors form an orthogonal basis[3].

What do we mean by "close to" linear subspaces? One of the most natural ways of measuring this is the length of the distance between the vector and the orthogonal projection onto the space. Suppose we have the space $Z = \operatorname{span}\{\boldsymbol{w}_1, \dots \boldsymbol{w}_d\}$, with $\boldsymbol{w} \in \mathbb{R}^n$ and $d < n$, we define a matrix containing the basis vectors

$$W = \begin{bmatrix} \boldsymbol{w}_1 & \dots & \boldsymbol{w}_d \end{bmatrix}.$$

The projection onto Z can be calculated by the least squares solution

$$\operatorname{proj}_Z(\boldsymbol{y}) = W\boldsymbol{x}, \qquad \text{where } \boldsymbol{x} = \arg\min_{\boldsymbol{x}\in\mathbb{R}^d} \|W\boldsymbol{x} - \boldsymbol{y}\|^2.$$

Because W is assumed orthogonal, this projection is simply $\operatorname{proj}_Z(\boldsymbol{y}) = WW^\top \boldsymbol{y}$. The distance between a vector $\boldsymbol{y}$ and its orthogonal projection onto the space is then

$$\|B\boldsymbol{y}\|^2 = \boldsymbol{y}^\top B^\top B\boldsymbol{y}, \quad B = I - WW^\top.$$

This means that we can use this matrix B to create a so-called semi-norm

$$\|\boldsymbol{x}\|_B^2 = \|B\boldsymbol{x}\|^2 = x^\top B^\top B\boldsymbol{x} = \boldsymbol{x}^\top B\boldsymbol{x}.$$

This satisfies all the qualities of a norm, except that there exists $\boldsymbol{x} \neq \mathbf{0}$ so that $\|\boldsymbol{x}\|_B = 0$, which is where the prefix "semi" comes from. If B is SPD, this is a proper norm.
The question then is how do we obtain the basis vectors W_0 and W_1 for our problem? It is more or less impossible to handcraft these, so we need to apply some machine learning. Specifically, we will find the basis vectors as the left singular vectors of two datasets of images. This is not really necessary to complete the project as you will get these handed out, but if you are very interested in this there is a section in the appendix. The basis vectors that will be handed out are plotted in Figure 3.

[3] If provided data would not provide this, we could use our algorithm for the rank and Gram Schmidt to create this.

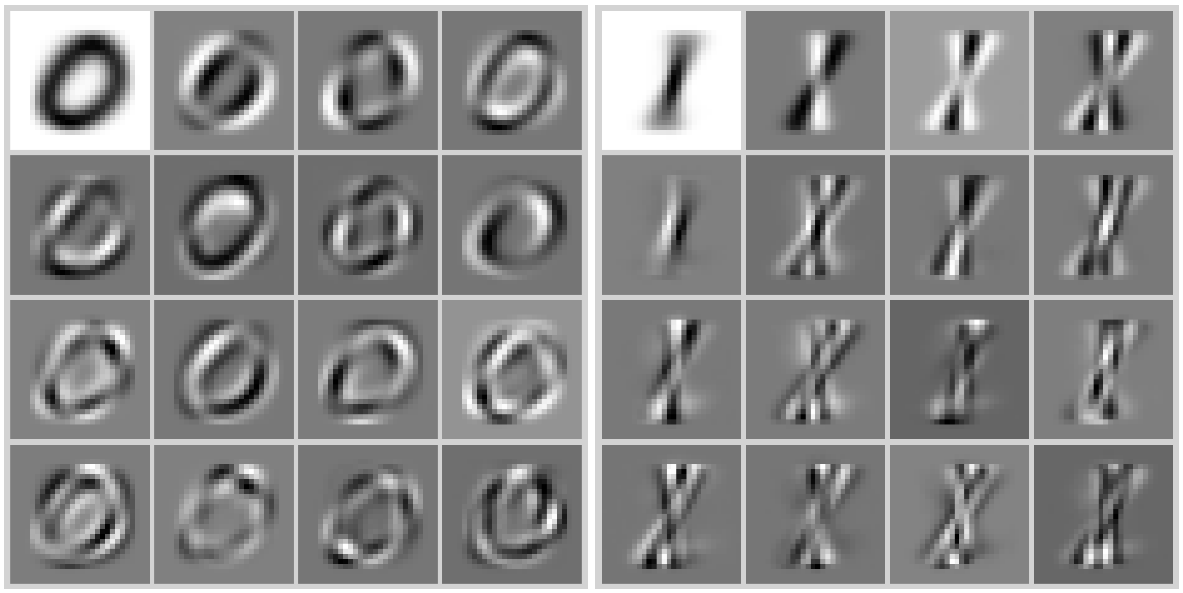
Figure 3: Basis vectors W_0 and W_1 respectively which we use to create the norm used for the rest of the project.

Assume now that W_0 and W_1 are given, we can then find the minimal norm and the Tikhonov regularisation solutions similar to Task 1 and 2, with a slight amount of extra work. For the minimal norm formulation, the problem reads (using $\boldsymbol{x} = \begin{pmatrix}\boldsymbol{u}\\ \boldsymbol{v}\end{pmatrix} \in \mathbb{R}^{2n}$)

$$\operatorname*{arg\,min}_{\boldsymbol{x}} \frac{1}{2}\boldsymbol{u}^\top B_0 \boldsymbol{u} + \frac{1}{2}\boldsymbol{v}^\top B_1 \boldsymbol{v}^\top, \quad \text{such that } \boldsymbol{u} + \boldsymbol{v} = \boldsymbol{y}$$

or equivalently

$$\operatorname*{arg\,min}_{\boldsymbol{x}} \frac{1}{2}\boldsymbol{x}^\top B\boldsymbol{x}, \quad \text{such that } A\boldsymbol{x} = \boldsymbol{y},$$

with

$$B = \begin{bmatrix} B_0 & 0 \\ 0 & B_1 \end{bmatrix}, \qquad A = \begin{bmatrix} I & I \end{bmatrix}.$$

The corresponding Lagrangian function reads

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) = \frac{1}{2}\boldsymbol{x}^\top B\boldsymbol{x} + \boldsymbol{\lambda}^\top (A\boldsymbol{x} - \boldsymbol{y}).$$

Similarly, the Tikhonov regularisation reads

$$\operatorname*{arg\,min}_{\boldsymbol{x}} \|A\boldsymbol{x} - \boldsymbol{y}\|^2 + \mu\|B\boldsymbol{x}\|^2.$$

For both the minimal norm approach and the Tikhonov regularisation approach the idea is the same: we want to (roughly) solve the system $A\boldsymbol{x} = \boldsymbol{u} + \boldsymbol{v} = \boldsymbol{y}$ while making sure that $\boldsymbol{u}$ is small in some norm, and $\boldsymbol{v}$ is small in some other norm.

In the following task you have to only implement one of the solvers, that is, *either* do 3.b) *or* 3.c). If you have the time, for sure, do both and compare.

Task 3

3.a) Let $B = I - WW^\mathsf{T}$ and W be a matrix with (pairwise) orthogonal columns. Show that B is a projection matrix (i. e. $B^2 = B$) and that it is symmetric positive semi-definite.

Hint. Use the definition of eigenvalues and eigenvectors $A\boldsymbol{v} = \lambda\boldsymbol{v}$ to show that a projection matrix can only have two different eigenvalues.

3.b) Minimal norm

1) Show that this approach in general leads to the system

$$\begin{bmatrix} \frac{1}{2}B^\mathsf{T} + \frac{1}{2}B & A^\mathsf{T} \\ A & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} 0 \\ \boldsymbol{y} \end{bmatrix},$$

where 0 are suitably sized matrices containing only 0. For our problem, show that this system becomes

$$\begin{bmatrix} I - W_0W_0^\mathsf{T} & 0 & I \\ 0 & I - W_1W_1^\mathsf{T} & I \\ I & I & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{v} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \boldsymbol{y} \end{bmatrix},$$

where I are suitably sized identity matrices.

2) Eliminate $\boldsymbol{\lambda}$ from the equations and find that you can solve the system as $C\boldsymbol{u} = \boldsymbol{d}$ for some matrix C and vector $\boldsymbol{d}$, and then obtain $\boldsymbol{v} = \boldsymbol{y} - \boldsymbol{u}$.

3) Implement a suitable method for calculating $\boldsymbol{u}$ and $\boldsymbol{v}$ using the minimal norm approach. You can use any method from the lecture or this project, including any NumPy or SciPy functions. Make sure to *concisely* explain your approach and potential advantages/disadvantages.

3.c) Tikhonov regularisation

1) Show that this approach in general leads to the system

$$(A^\mathsf{T} A + \mu B^\mathsf{T} B)\boldsymbol{x} = A^\mathsf{T}\boldsymbol{y},$$

and in our case the system

$$\begin{bmatrix} I + \mu(I - W_0 W_0^\mathsf{T}) & I \\ I & I + \mu(I - W_1 W_1^\mathsf{T}) \end{bmatrix} \begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{v} \end{bmatrix} = \begin{bmatrix} \boldsymbol{y} \\ \boldsymbol{y} \end{bmatrix},$$

2) Show that you can solve this by first solving a system $C_0\boldsymbol{u} = \boldsymbol{d}_0$ for some matrix C_0 and vector $\boldsymbol{d}_0$, and then another similar system $C_1\boldsymbol{v} = d_1$.

3) Implement a suitable method for calculating $\boldsymbol{u}$ and $\boldsymbol{v}$ using the Tikhonov regularisation approach. You can use any method from the lecture or this project, including any NumPy or SciPy functions. Make sure to *concisely* explain your method and potential advantages/disadvantages. Select for example $\mu = 1.0$, but feel free to try other parameters.

3.d) Test the solver you developed in 3.b) or 3.c) by first loading the basis vectors stored in `W0.npy` and `W1.npy`. See the handed out code for how to do this. Test your method on the data in `mixed.npy`, which contains 100 different test examples. Time how long it takes to separate all 100 images, and plot at least one example and the resulting separated images. Make a quick qualitative assessment of the separated image(s). Does this approach work for single channel source separation? Does the approach have some drawbacks?

Appendix - Finding the bases W_0 and W_1 in Task 3

For actual problems, handcrafting W_0 and W_1 is nearly impossible so we will need to rely on some learning techniques. For those of you who took TMA4320 VitBer last semester, this should remind you of the last project, as we use the same ideas as we did there.

We assume that we have a dataset of "clean" signals of both types $\boldsymbol{u}$ and $\boldsymbol{v}$, and we can then use the SVD to find the d "best" orthognal basis vectors for this basis. Remember the first d singular vectors of a matrix A create the "best" rank d approximation to the matrix A.

Assume we have N different images of class 0, $\boldsymbol{u}^{(j)}$, where the superscript denotes the j-th data, and similarly N different images of class 1, $\boldsymbol{v}^{(j)}$. We can then create two matrices $\mathcal{U}$ and $\mathcal{V}$ where each column is one of the data points. We can then calculate the SVDs

$$\mathcal{U} = U_0\Sigma_0V_0^{\mathsf{T}}, \quad \mathcal{V} = U_1\Sigma_1V_1^{\mathsf{T}}.$$

Here U_0 and U_1 contain an orthogonal basis for the spaces spanned by our data. This approach is also called Principal Component Analysis (PCA), which is something you will learn more about in TMA4267 Linear statistical models and TMA4268 Statistical Learning. Now, we don't want to use the entire basis, both because this is computationally wasteful and because it yields worse results in the end, so we will extract the first d columns of U_0 and U_1, and use these as the matrices W_0 and W_1. The results of doing this with $d = 16$ and $N = 5000$ is shown in Figure 3.