Numerical Algorithms for Computer Vision and Machine Learning
F. Bernard, L. Sang, F. Hofherr, Technische Universität München
Winter Semester 20/21

Matlab Submission 2

Due on 18th of January, 18:00

Exercise 1 (Simple SVD Solver)**.** In this exercise you will implement most of the components for a simple SVD solver from scratch.

a) (25 Points) We first implement the QR Algorithm from the lecture to compute the eigenvalues and eigenvectors for **symmetric** matrices. Complete the function `[V, D] = eigViaQR(A, thres)` that returns the eigenvalues of the symmetric matrix `A` on the diagonal of the (diagonal) matrix `D` as well as the respective eigenvectors as the columns of `V`.

Also implement a convergence control that stops the iterations when the eigenvalue equations are fulfilled up to the scalar `thres`, i.e. as soon as

$$\|Av_i - \lambda_i v_i\| \leq \texttt{thres}$$

holds for all eigenvalue/eigenvector pairs.

Note: Apart from Matlabs QR decomposition (`qr`) you are not allowed to use any other "complex" Matlab functions. The standard operations like `+, -, *, eye, size, mod, diag, sqrt, ^2,`... are allowed, of course.

b) (25 Points) We now use the eigenvalue solver from the last exercise to implement the SVD solver. Complete the function `[U, S, V] = svdViaEig(A)` that returns the matrices of the **partial** singular value decomposition for the matrix $A \in \mathbb{R}^{m\times n}$ with $m \geq n$ (as usual `S` $= \Sigma$).

To find out to which precision the eigenvalue problem must be solved, compare your results to Matlabs `svd`. For the submission you can use the standard value `thres` $= 1 \cdot 10^{-6}$.

Note: Again, only standard Matlab operations are allowed, as well as `sort`. Additionally, you may also use `eig` as a "complex" function (in case you did not implement the eigenvalue solver from the previous exercise).

Exercise 2 (Rigid Procrustes Problem)**.** (20 Points) In this exercise we use your SVD solver to implement a solver for the (slightly simplified) rigid Procrustes problem. As introduced in the lecture, we are given two d-dimensional point clouds $Y, Z \in \mathbb{R}^{n\times d}$ and we want to find the affine transformation that transforms the points in Z as close to Y as possible. This leads to the following optimization problem

$$(R^*, t^*) = \underset{R^\top R = I, t\in\mathbb{R}^3}{\arg\min} \left\| Y - \left(ZR + \mathbf{1}t^\top \right) \right\|_F^2 . \tag{1}$$

Note that we have relaxed the problem slightly compared to the lecture by dropping the constraint that $\det(R) = 1$, which allows reflections besides the rotations. In practise this is often no problem, however, one could also solve the more restricted problem as well, if necessary.

For the solution of the problem we introduce $\bar{y}$ and $\bar{z}$ which are the column means of Y and Z as well as the mean-corrected point clouds $\bar{Y} = Y - \bar{y}$ and $\bar{Z} = Z - \bar{z}$. It can be shown that the optimal transformation parameters can be found as

$$\begin{aligned} R^* &= UV^\top \\ t^* &= \bar{y} - \bar{z}R^*, \end{aligned}$$

where U and V are the matrices from the SVD of $\bar{Z}^\top \bar{Y} = U\Sigma V^\top$.

Complete the function `[R, t] = procrustesAlignment(Y, Z)` that takes the point clouds Y and Z as input and returns the orthogonal matrix R and the translation vector t that minimizes Equation (1). To test your code you can download the two point clouds contained in `PointcloudsEx2.mat`.

Note: You are allowed to use Matlabs `svd` (in case you did not succeed with the previous exercises).

Exercise 3 (The average S)**.** (30 Points) The Procrustes alignment can be used to compute the *Procrustes average* of a collection $\{Z_l\}_1^L$ of L shapes, which is defined by the following problem

$$\min_{\{R_l\}_1^L, \{t_l\}_1^L, M} \sum_{l=1}^{L} \left\| M - \left(Z_l R_l + \mathbf{1} t_l^\top \right) \right\|_F^2 . \tag{2}$$

That is, we want to find the shape M that minimizes the sum of the Procrustes distances to all the shapes well as the respective transformations that transform each individual shape as close to M as possible[1]. The problem can be solved by the following iterative scheme

```
$M \leftarrow \bar{Z}_1$
**while** $\|M_{new} - M_{old}\| > \varepsilon$ **do**
    1. For each shape solve the Procrustes alignment problem for fixed $M$ to
       obtain $Z_l' \leftarrow Z_l R_l + \mathbf{1} t_l^\top$
    2. Update $M \leftarrow \frac{1}{L} \sum_{l=1}^{L} Z_l'$
**end while**
```

We initialize the shape as the mean-corrected fist shape to have an initial guess as well as to position the average shape at the origin.

Complete the function

[1]Note that, since we are not regularizing R and t, the solution has a rotation and translation freedom, which we will ignore, however. Just keep in mind that M could drift away

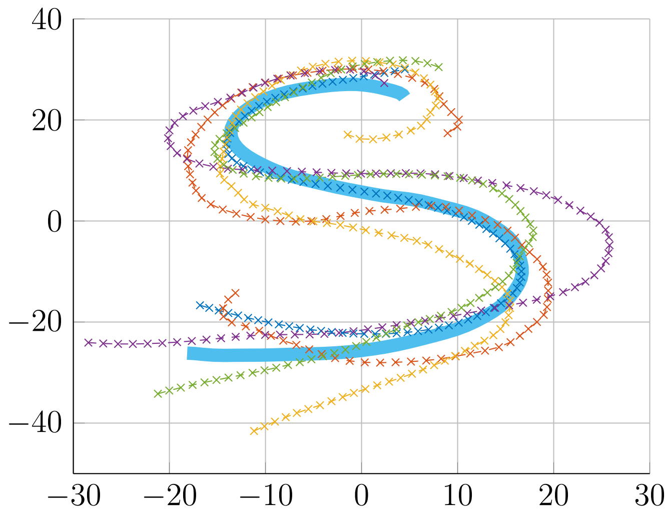

Figure 1: Possible result of Exercise 3. The average S is plotted in bold, while the optimally aligned individual shapes are plottet in thin lines with crosses in the background.

`[M, rotations, translations] = generalizedProcrustes(shapes, eps)` that takes as input a cell array, containing the point clouds $Z_l \in \mathbb{R}^{n\times d}$ and returns the average shape M as well as cell arrays `rotations` and `translations` containing the matrices R_l and t_l to transform the individual shapes optimally close to the average shape M.

To test your code download `HandwrittenS.mat` which contains 2D point-clouds corresponding to polygonal chains of handwritten letters S (they could have been obtained e.g. by tracking the movement of a stylus on a tablet). Using your code you can compute the "average S" from the given shapes. The result might look like in Figure 1.