The Chinese University of Hong Kong, Shenzhen
SDS · School of Data Science

Andre Milzarek · Fall Semester 2023-24

DDA 3005 — Numerical Methods

Course Project

Singular Value Decomposition

The goal of this course project is to investigate and study different algorithms to compute singular value decompositions and to utilize SVDs in several interesting imaging applications.

Project Tasks.

1. Singular Value Decomposition. Implement the two-phase procedure discussed in the lecture to compute a singular value decomposition

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$$

of a general matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ in `MATLAB` or `Python`.

   - *Phase I.* Implement the Golub-Kahan bidiagonalization to reduce the matrix $\boldsymbol{A}$ to bidiagonal form. In each step of the bidiagonalization, the matrix $\boldsymbol{A}$ is multiplied by two orthogonal matrices from the left and right in an alternating fashion. First, a left Householder transformation is applied to introduce a column of zeros below the diagonal (this step is identical to the reduction in the QR factorization). Afterwards a right Householder transformation is used to eliminate all entries after the first superdiagonal element in the current row. This procedure is repeated until the updated matrix has bidiagonal form.

   - *Phase II–A.* Suppose the matrix $\boldsymbol{A}$ has been reduced to bidiagonal form $\boldsymbol{B} \in \mathbb{R}^{n\times n}$ (or $\mathbb{R}^{m\times m}$; here we discard the additional zero rows or columns in the matrix returned by phase I). Implement the QR iteration with Wilkinson shift and deflation to calculate an eigendecomposition of the symmetric tridiagonal matrix $\boldsymbol{B}^\top\boldsymbol{B}$. Use the obtained eigenvalues and eigenvectors to construct or compute a singular value decomposition of the initial matrix $\boldsymbol{B}$.

     Combine the results from phase I and II to obtain a full singular value decomposition $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$ of $\boldsymbol{A}$, i.e., return the singular vectors $\boldsymbol{U}$, $\boldsymbol{V}$, and the singular values $\boldsymbol{\Sigma}$.

   - *Phase II–B.* Consider the following iterative procedure:

     Initialize $\boldsymbol{X}^0 = \boldsymbol{B}$ *and for* $k = 0, 1, 2, \ldots$ *do:*

$$\boldsymbol{Q}_k\boldsymbol{R}_k = (\boldsymbol{X}^k)^\top, \quad \boldsymbol{L}_k\boldsymbol{L}_k^\top = \boldsymbol{R}_k\boldsymbol{R}_k^\top, \quad \boldsymbol{X}^{k+1} = \boldsymbol{L}_k^\top.$$

     This method first computes a QR factorization of $(\boldsymbol{X}^k)^\top$; then a Cholesky decomposition of the tridiagonal matrix $\boldsymbol{R}_k\boldsymbol{R}_k^\top$ is computed and $\boldsymbol{X}^{k+1}$ is set as the (upper) bidiagonal matrix $\boldsymbol{L}_k^\top$. (Notice that $\boldsymbol{R}_k$ is an upper bidiagonal matrix whereas $\boldsymbol{L}_k$ is a lower bidiagonal matrix).

     Verify that this algorithm coincides with the QR iteration for the tridiagonal matrix $\boldsymbol{B}^\top\boldsymbol{B}$ with zero shift. Implement this procedure as an alternative to the traditional QR algorithm used in the previous part (phase II–A). You can perform a deflation step as soon as the elements $\boldsymbol{X}^k(\texttt{1},\texttt{2})$ or $\boldsymbol{X}^k(\texttt{n-1},\texttt{n})$ of the bidiagonal iterate $\boldsymbol{X}^k$ have sufficiently small absolute value (say $< 10^{-12}$ or $< 10^{-14}$).

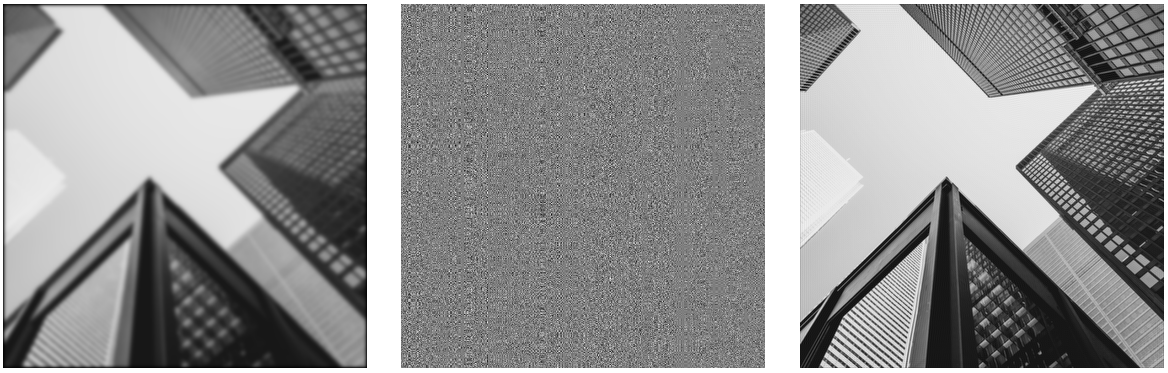

Figure 1: The blurry image is generated via $\boldsymbol{B} = \boldsymbol{T}^{66}\boldsymbol{X}\boldsymbol{T}^{66}$ where $\boldsymbol{T}$ is a tridiagonal matrix with the value 2.1/4.1 on the diagonal and 1/4.1 on the two off-diagonals. The picture in the middle shows the reconstruction $\boldsymbol{T}^{-66}\boldsymbol{B}\boldsymbol{T}^{-66}$ using `MATLAB`'s backslash operator. The image on the right uses truncated SVDs with $\ell_{\texttt{trunc}} = r_{\texttt{trunc}} = 333$.

Hints and Further Guidelines:

- You can use `MATLAB` or `Python` inbuilt code to compute QR or Cholesky factorizations. Notice that it might be possible to improve the performance of your code by using specialized Cholesky factorizations that allow to exploit the structure of $\boldsymbol{R}_k\boldsymbol{R}_k^\top$.
- You may assume that the matrix $\boldsymbol{R}_k\boldsymbol{R}_k^\top$ is positive definite to ensure applicability of the Cholesky factorization. Hence, this assumes that $\boldsymbol{A}$ has full column or row rank. You can return an error message if the Cholesky factorization fails.
- As a first step, you can concentrate on the computation of the singular values. For debugging, you can compare your results with `MATLAB` or `Python` code on small scale examples.
- *Optional:* The stopping criterion in phase II–B can be further improved by checking if *any* element on the superdiagonal of $\boldsymbol{X}^k$ has sufficiently small absolute value. In this case, the matrix $\boldsymbol{X}^k$ can be decomposed into two smaller bidiagonal matrices. Investigate whether such a strategy can enhance the performance of your algorithm.

2. Deblurring Revisited. In this part of the project, we want to explore how the SVD can be utilized in the context of deblurring problems. The general deblurring problem can be modeled as follows. Given a blurry image $\boldsymbol{B} \in \mathbb{R}^{n\times n}$ and two blurring kernels $\boldsymbol{A}_\ell \in \mathbb{R}^{n\times n}$ and $\boldsymbol{A}_r \in \mathbb{R}^{n\times n}$, we consider linear systems of the form

$$\boldsymbol{A}_\ell\boldsymbol{X}\boldsymbol{A}_r = \boldsymbol{B}.$$

We seek to recover the original image $\boldsymbol{X} \in \mathbb{R}^{n\times n}$, i.e., we can set $\boldsymbol{X} = \boldsymbol{A}_\ell^{-1}\boldsymbol{B}\boldsymbol{A}_r^{-1}$ if the matrices $\boldsymbol{A}_\ell$ and $\boldsymbol{A}_r$ are invertible. In one of our exercises, we have already discussed such an application. Here, we want to study and apply truncation techniques based on the SVD that allow to resolve bad conditioning or singularity issues when computing $\boldsymbol{X} = \boldsymbol{A}_\ell^{-1}\boldsymbol{B}\boldsymbol{A}_r^{-1}$ via conventional LU or QR factorizations.

As discussed, a typical format for the blurring kernels $\boldsymbol{A}_\ell$ and $\boldsymbol{A}_r$ is

$$\boldsymbol{A} = \begin{bmatrix} a_n & a_{n-1} & & a_2 & a_1 \\ a_{n+1} & a_n & a_{n-1} & & a_2 \\ & \ddots & \ddots & \ddots & \\ a_{2n-2} & & \ddots & \ddots & a_{n-1} \\ a_{2n-1} & a_{2n-2} & & a_{n+1} & a_n \end{bmatrix} \in \mathbb{R}^{n\times n}, \quad a_i \in \mathbb{R}, \quad \forall\, i.$$

For instance, if $\boldsymbol{A}$ is symmetric with $a_i \geq 0$ and $a_1 + \cdots + a_n = 1$, then the corresponding blurring kernel computes weighted averages of pixels in the matrix $\boldsymbol{X}$ which results in the blurry image $\boldsymbol{B}$.

- On Blackboard, we have provided different test images $\boldsymbol{X}$ that you can use for the deblurring tasks. Download the test image package from BB.

  Select several images and construct suitable blurring kernels $\boldsymbol{A}_\ell$ and $\boldsymbol{A}_r$ as well as the corresponding blurry image $\boldsymbol{B}$. For instance, you can choose $\boldsymbol{A}_\ell$ and $\boldsymbol{A}_r$ as in the deblurring task in exercise sheet 04.

- Compute the SVDs $\boldsymbol{A}_\ell = \boldsymbol{U}^\ell \boldsymbol{\Sigma}^\ell (\boldsymbol{V}^\ell)^\top$ and $\boldsymbol{A}_r = \boldsymbol{U}^r \boldsymbol{\Sigma}^r (\boldsymbol{V}^r)^\top$ using your code from the first part of the project. Plot the singular values of $\boldsymbol{A}_\ell$ and $\boldsymbol{A}_r$ and generate the truncated reconstruction

$$\boldsymbol{A}^+_{\ell,\texttt{trunc}} = \sum_{i=1}^{\ell_{\texttt{trunc}}} \frac{\boldsymbol{v}_i^\ell (\boldsymbol{u}_i^\ell)^\top}{\sigma_i^\ell}, \quad \boldsymbol{A}^+_{r,\texttt{trunc}} = \sum_{i=1}^{r_{\texttt{trunc}}} \frac{\boldsymbol{v}_i^r (\boldsymbol{u}_i^r)^\top}{\sigma_i^r}, \quad \boldsymbol{X}_{\texttt{trunc}} = \boldsymbol{A}^+_{\ell,\texttt{trunc}} \boldsymbol{B} \boldsymbol{A}^+_{r,\texttt{trunc}}.$$

  Here, we assume $\sigma_1^\ell \geq \sigma_2^\ell \geq \cdots \geq \sigma^\ell_{\ell_{\texttt{trunc}}}$ and $\sigma_1^r \geq \sigma_2^r \geq \cdots \geq \sigma^r_{r_{\texttt{trunc}}}$, i.e., we truncate the pseudoinverses $\boldsymbol{A}_\ell^+$ and $\boldsymbol{A}_r^+$ according to the leading singular values. (The idea is to exclude terms corresponding to very small singular values σ_i^ℓ or σ_i^r in the expansions of $\boldsymbol{A}_\ell^+$ and $\boldsymbol{A}_r^+$). Test different suitable choices of $\ell_{\texttt{trunc}}$ and $r_{\texttt{trunc}}$ and report the quality of the obtained solution $\boldsymbol{X}_{\texttt{trunc}}$. You can use the peak-signal-to-noise ratio

$$\texttt{PSNR} = 10 \log_{10}(n^2 / \|\boldsymbol{X}_{\texttt{trunc}} - \boldsymbol{X}\|_F^2)$$

  to measure the quality of your reconstruction.

- Test at least three different scenarios / images. Compare the performance and runtime of the phase II–A and II–B approaches. Which method returns more accurate singular values? Which algorithm generally converges faster?

Hints and Guidelines:

- You can resize/rescale the images to test your implementation first on simpler and low-dimensional examples. This part of the project can be implemented independently from part 1. You can first use `MATLAB` or `Python` inbuilt functions to test your model; code from part 1 can be incorporated at a later stage.

- You can construct $\boldsymbol{X}_{\texttt{trunc}}$ using `MATLAB` or `Python` inbuilt code if the algorithms from part 1 do not return correct or highly different results. Provide and include a short discussion in your final report in such a case.

3. Power Iterations and Video Background Extraction. In this part of the project, we want to utilize the SVD and power iterations in order to extract the background information of some given video data. In general, a video $\boldsymbol{M} \in \mathbb{R}^{m \times n \times 3 \times s}$ can be modeled as a $m \times n \times 3 \times s$-dimensional tensor, i.e., $\boldsymbol{M}$ corresponds to a sequence of frames

$$\boldsymbol{M}_1 = \boldsymbol{M}(\texttt{1:m}, \texttt{1:n}, 1), \quad \boldsymbol{M}_2 = \boldsymbol{M}(\texttt{1:m}, \texttt{1:n}, 2), \quad \ldots \quad \boldsymbol{M}_s = \boldsymbol{M}(\texttt{1:m}, \texttt{1:n}, s)$$

and each $\boldsymbol{M}_i \in \mathbb{R}^{m \times n \times 3}$ represents a single colored frame or image of the video data $\boldsymbol{M}$. Our task is to detect the common background of the frames $\boldsymbol{M}_1, \ldots, \boldsymbol{M}_s$ and to extract it in a single image $\boldsymbol{B} \in \mathbb{R}^{m \times n \times 3}$ or $\boldsymbol{B} \in \mathbb{R}^{m \times n}$. An example of different video frames and a corresponding recovered video background is given in Figure 2.

In the following, we assume that each frame $\boldsymbol{M}_i$ is a grayscale image, i.e., we have $\boldsymbol{M}_i \in \mathbb{R}^{m \times n}$ for all i. The image $\boldsymbol{B}$ can be obtained as follows:

Figure 2: The first five images correspond to the frames $\boldsymbol{M}_{50}$, $\boldsymbol{M}_{100}$, $\boldsymbol{M}_{150}$, $\boldsymbol{M}_{200}$, and $\boldsymbol{M}_{250}$. The last image depicts the extracted background information $\boldsymbol{B}$ from the video $\boldsymbol{M}$.

1. We first reshape each frame $\boldsymbol{M}_i$ as a long vector $\boldsymbol{m}_i = \mathrm{vec}(\boldsymbol{M}_i)$ by stacking all of the columns of $\boldsymbol{M}_i$. We then build the video matrix

$$\boldsymbol{A} := \begin{bmatrix} \boldsymbol{m}_1 & \boldsymbol{m}_2 & \dots & \boldsymbol{m}_s \end{bmatrix} \in \mathbb{R}^{mn \times s}.$$

2. Let $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$ be a singular value decomposition of $\boldsymbol{A}$ and let σ_1, $\boldsymbol{u}_1$, and $\boldsymbol{v}_1$ denote the largest singular value and the associated singular vectors of $\boldsymbol{A}$. Then, $\boldsymbol{B}$ is given via:

$$\mathrm{vec}(\boldsymbol{B}) = \sigma_1(\boldsymbol{v}_1^\top \boldsymbol{e}_1) \cdot \boldsymbol{u}_1,$$

   i.e., $\boldsymbol{B}$ coincides with the first column of the rank-1 matrix $\sigma_1 \boldsymbol{u}_1 \boldsymbol{v}_1^\top$.

The goal of this part of the project is to develop an efficient SVD- or power iteration-based algorithm that can return and calculate the background image $\boldsymbol{B}$ for a given video input data $\boldsymbol{M} \in \mathbb{R}^{m \times n \times s}$ (or $\mathbb{R}^{m \times n \times 3 \times s}$).

– On Blackboard, we have provided twenty different real-world video datasets. The videos are all taken from the website `https://pixabay.com` and are stored as `mp4`-files. Download the video datasets (from BB or visit the mentioned website) and implement a video reader that can build the matrix $\boldsymbol{A}$.

  The following commands can be helpful:

  `MATLAB:` `VideoReader`, `readFrame`, `hasFrame`, `rgb2gray`, `VideoWriter`.
  `Python:` `imageio`, `moviepy.editor` (`VideoFileClip`, `get_frame`), `scipy` (`misc.imresize`).

  The setup in `Python` can require some work. On BB, we have provided a few converted videos as `mat`-files that follow the construction of $\boldsymbol{A}$. You can use those files at the beginning or if there are issues with the setup.

– Implement a SVD- or power iteration-based method to compute the largest singular value σ_1 and the associated singular vectors $\boldsymbol{u}_1$ and $\boldsymbol{v}_1$ of $\boldsymbol{A}$. Construct the background image $\boldsymbol{B}$ as specified in the project description. Test your code on at least three different videos and report the required runtime and recovered backgrounds.

– Compare the performance and runtime of your algorithm with standard software packages. In `MATLAB`, the command $[\boldsymbol{U}, \boldsymbol{S}, \boldsymbol{V}] = \texttt{svds}(\boldsymbol{A}, k)$ can be used to compute the k-largest singular values and the associated singular vectors. In `Python`, you can call `scipy.sparse.linalg.svds` (this also works for dense inputs).

Hints and Guidelines:

– This is a (very) large-scale application. Consider resizing/rescaling the video resolution or frames if you encounter memory issues and convert the video frames to grayscale to save storage. Try to implement your SVD- or power iteration-based algorithm as efficiently as possible to improve performance.

Project Report. This project is designed for groups of *four or five students*. Please send the following information to `lijiang@link.cuhk.edu.cn` before **December, 12th, 12:12 pm**:

- Name and student ID number of the participating students in your group, group name.

Please contact the instructor in case your group is smaller to allow adjustments of the project outline and requirements.

A report should be written to summarize the project and to collect and present your different results. The report itself should take no more than 10–13 typed pages plus a possible additional appendix. It should cover the following topics and items:

- What is the project about?
- What have you done in the project? Which algorithmic components have you chosen to implement? What are your main contributions?
- Summarize your main results and observations.
- Describe your conclusions about the different problems and methods that you have studied.

You can organize your report following the outlined structure in this project description. As the different parts in this project only depend very loosely on each other, you can choose to distribute the different tasks and parts among the group members. Please clearly indicate the responsibilities and contributions of each student and mention if you have received help from other groups, the teaching assistant, or the instructor.

Try to be brief, precise and fact-based. Main results should be presented by highly condensed and organized data and not just piles of raw data. To support your summary and conclusions, you can put more selected and organized data into an appendix which is not counted in the page limit. Please use a cover page that shows the names and student ID numbers of your group members.

The deadline for the report submission is **December, 24th, 11:59pm (night)**. Please submit your report (as pdf-document) and all supporting materials and code online using Blackboard.