Numerical Linear Algebra

master in fundamental principles of data science

Project 2

SVD Applications

Author
Vladislav Nikolov Vasilev

Faculty of Mathematics and Computer Science

Academic year 2021-2022

Contents

1 Introduction

The goal of this project is to discuss three common applications of the Singular Value Decomposition (SVD). First, let's briefly review what the SVD is.

Given a rectangular matrix $A \in \mathbb{R}^{m \times n}$ with $m \geq n$, we can express it as

$$A = U \Sigma V^T$$

where $U \in \mathbb{R}^{m \times m}$ and $V \in \mathbb{R}^{n \times n}$ are two orthogonal basis and $\Sigma \in \mathbb{R}^{m \times n}$ is a matrix that can be divided in the diagonal block $\Sigma\left[1:n, 1:n\right]$ with the singular values the singular values σ_i in the diagonal and the zero block $\Sigma\left[(m-n):m, 1:n\right]$. The singular values are ordered such that $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_n \geq 0$. Since U and V are orthogonal, we have that $U^{-1} = U^T$ and $V^{-1} = V^T$.

There are some cases in which we can also compute a reduced version of the SVD, which is faster and reduces the amount of memory needed to store the matrices. This can be particularly useful in scenarios where the matrix A is rank deficient.

Let $A \in \mathbb{R}^{m \times n}$ be a rectangular matrix with $rank(A) = r$, where $r < n$. For this case, the reduced SVD can be computed as

$$A = U_r \Sigma_r V_r^T \tag{1}$$

where $U_r \in \mathbb{R}^{m \times r}$ and $V_r^T \in \mathbb{R}^{r \times r}$ are the orthogonal basis and $\Sigma \in \mathbb{R}^{r \times r}$ is the diagonal matrix containing the nonzero singular values.

There are many applications of the SVD, but in this project we are going to focus on three of them: solving the Least Squares Problem, image compression and Principal Component Analysis.

2 Least Squares Problem

The first application that we are going to address is the Least Squares Problem (LSP). Recall that in this problem we have a matrix $A \in \mathbb{R}^{m \times n}$ with $m \geq n$ and a vector $b \in \mathbb{R}^m$. Our goal is to find a vector $x \in \mathbb{R}^n$ such that Ax is as close to b as possible. This can be expressed as a minimization problem, in which we have

$$\min ||Ax - b||_2 \tag{2}$$

There are many ways in which we can solve this problem: using iterative methods, normal equations, QR factorization or SVD, among many others. In this case, we are going to focus on the SVD method, which is also very appropriate for the rank deficient case.

Consider the reduced version of the SVD seen in expression (1). We can use it to compute the *Moore-Penrose inverse* (also called *pseudo-inverse*) of our matrix A as

$$A^+ = V_r \Sigma_r^{-1} U_r^T$$

with $A^+ \in \mathbb{R}^{n \times m}$. Thus, we can express the minimization problem (2) as follows:

$$x = A^+ b$$

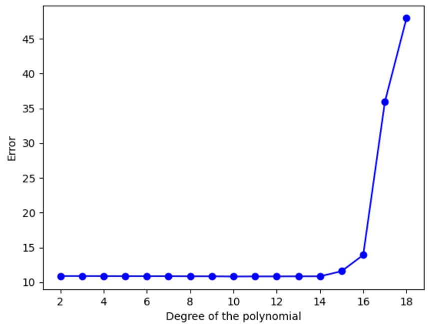
The solution obtained with this method is well-conditioned provided that the smallest nonzero singular value of A is not too small.

To solve the LSP problem, we have created a script called `lsp.py` which contains a function called `solve_lsp_svd`. This function computes the reduced SVD of an input matrix A, computes the pseudo-inverse and multiplies it by the b vector to get the value of x. For the computation of the SVD we have used the function `numpy.linalg.svd`. To get the rank of the matrix we have chosen the singular values greater than a certain tolerance, which can be fixed or is computed as `Numpy` does when computing the rank of a matrix (see implementation for further details and reference).

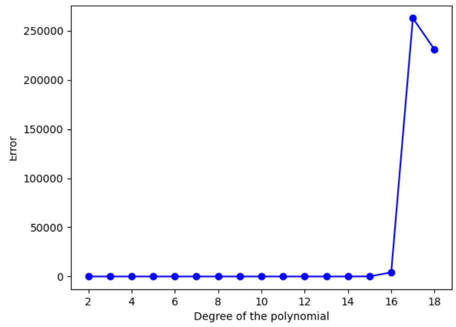
Let's now apply this method to some problems and see how it performs compared to another method: the QR factorization. The before mentioned script also contains a function called `solve_lsp_qr_full_rank`, which is used for the full rank problem, and another one called `solve_lsp_qr_rank_deficient`, which uses the QR factorization with pivoting to solve the rank deficient case.

2.1 Polynomial fitting

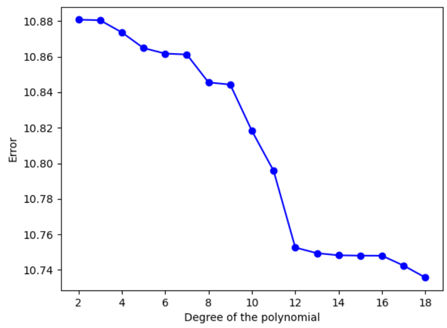
In this problem we are given a set of points x and their values b. We are going to try to approximate the values using polynomials of different degrees. To do that, we are going to use the SVD approach both with a fixed tolerance of 10^{-10} and an adaptive one based on `Numpy`'s tolerance for the computation of the rank of a matrix and the QR factorization for the full rank problem. Once we have the projection vector x, we are going to compute the error of the approximation using the 2-norm to see how good the it is and to compare the different methods.



(a) SVD with adaptive tolerance.



(b) SVD with fixed tolerance.



(c) QR factorization without pivoting.

Figure 1: Comparison of the approximation's error for different methods as the degree of the polynomial increases.

As we can see in figure 1, all of the methods yield a solution which has approximately the same error up until polynomials of degree 13. After that point, the matrix A starts to become close to rank deficient. This has a huge impact on the SVD with fixed tolerance because it doesn't take into account the magnitude of the singular values when discarding the less significant ones. Therefore, the problem is treated as a full rank one, and the problem becomes ill-conditioned because the smallest singular values are too small.

In the case of the SVD with adaptive tolerance, we can see that the error starts increasing even when adapting the tolerance to the magnitude of the singular values so that the problem is not treated as a full rank one. However, the error is not as large as in the previous case.

Finally, we can see that the QR factorization without pivoting outperforms the rest of the methods for large polynomials. It yields results with the lowest errors, and even the error seems to decrease as the degree of the polynomial increases.

2.2 The rank deficient LSP

For this case, we are going to study a problem in which we have a matrix $A \in \mathbb{R}^{15\times 11}$. However, we have that $rank(A) = 10$ since the first two columns are exactly the same. Therefore, since $r < n$, the matrix A is going to be rank deficient.

For this problem we have used two methods: the reduced SVD with adaptive tolerance and the QR factorization with pivoting since we cannot apply QR factorization directly to the matrix A.

Once again, we have solved the system and we have computed the error of the approximation using the 2-norm. In this case, both methods have produced approximately the same error, which is around 1.1496.

As we can see, both methods allow us to solve the problem with small error. However, the method based on the reduced SVD is much simpler and straightforward than the one using the QR factorization with pivoting. Therefore, for this problem we could have used the simplest one and still get a good result.

3 Image compression

Another problem where SVD can be applied is image compression, even though there are much better techniques out there. In order to compress images, we can use a **lower rank approximation** of them.

Fortunately, the SVD factorization has the property of giving the best low rank approximation matrix with respect to the Frobenius norm and the 2-norm. For a given value of $k = 1, \dots, r$, the matrix

$$A_k = \sum_{i=1}^{k} \sigma_i u_i v_i^T = U_k \Sigma_k V_k^T$$

is the best rank k approximation of A with respect to both the Frobenius norm and the 2-norm. For any of these two norms, we have that

$$\|A - A_k\| = \|\Sigma - \Sigma_k\| = \left\| \begin{bmatrix} 0 & & & & & \\ & \ddots & & & & \\ & & 0 & & & \\ & & & \sigma_{k+1} & & \\ & & & & \ddots & \\ & & & & & \sigma_r \\ & & & 0 & & \end{bmatrix} \right\| \tag{3}$$

The Frobenius norm for a given matrix A can be expressed in terms of its SVD factorization as follows:

$$\|A\|_F = \left\|U^T A V\right\|_F = \|\Sigma\|_F = \sqrt{\sum_{i=1}^{r} \sigma_i^2}$$

Following the results obtained in (3), we get that the Frobenius norm can be computed as

$$\|A - A_k\|_F = \sqrt{\sum_{i=k+1}^{r} \sigma_i^2}$$

The 2-norm can also be expressed in terms of the SVD:

$$\|A\|_2 = \left\|U^T A V\right\|_2 = \|\Sigma\|_2 = \sigma_1$$

Hence, using again the results from expression (3), we get the 2-norm can be computed as

$$\|A - A_k\|_2 = \sigma_{k+1}$$

4 Principal Component Analysis

4.1 Example problem

4.2 Genes problem

References

[1] Texto referencia
`https://url.referencia.com`