Report on the singular value decomposition method for solving a system of linear equations

Dominik Brandstetter (1605547)
Department of Physics, Faculty of Natural Sciences, Graz University
(Dated: November 17, 2020)

As part of the exercise "Methods of Modeling and Simulation" (20W PHM.204UB), I present a report on the results of a singular value decomposition (SVD) method applied to a simple system of linear equations. This report will present a quick overview of the SVD method, the results of it applied to a simple example including the residual error as well as the variance-covariance matrix.

1. Introduction

The focus of this work lies on the proof of concept for the singular value decomposition (SVD). We will apply it to a simple system of linear equations to not only solve it but also calculate the variance-covariance matrix for the resulting least-square fit.
First we will take a quick look at the theory behind SVD and state the for this work important equations in section 2. After that we briefly discuss the computational details and state the problem in section 3 and the results will be documented in section 4. The code can be found in the appendix A.

2. Theory

2.1. Singular Value Decomposition SVD

For every matrix $A \in \mathbb{R}^{n\times m}$ there exist (at least one pair of) orthogonal matrices $U \in \mathbb{R}^{n\times n}$ and $V \in \mathbb{R}^{m\times m}$ such that

$$A = UwV^T \tag{1}$$

We call the elements in the diagonal matrix $w = \text{diag}(\{s_1, s_2, ..., s_r\})$, with r being the rank of A, the singular values of A. They are the squares of eigenvalues of A^TA.[2] Both U and V are orthogonal[2, 3] and thus $U^TU = UU^T = V^TV = VV^T = I$ with I being the identity matrix.[2][1]
We see that a matrix A of rank r has at most r unique non-zero singular values. Because U and V are orthogonal we can interpret them as a unitary transformation, i.e. a rotation or reflection, in the $\mathbb{R}^{n\times n}$ and $\mathbb{R}^{m\times m}$, respectively. The matrix w then is, because it is diagonal, a scaling.

[1] The lecture notes [3] explicitly state that U is orthogonal but $UU^T \neq I$, however, this is a property of all orthogonal matrices[2] and it certainly holds true for the matrices in our example eq.(8).

2.2. Pseudoinverse A^{-g}

We want to solve the problem of a inhomgenous system of linear equations written as matrix form as

$$\boldsymbol{y} = A\boldsymbol{a} \tag{2}$$

However, A is in general not invertible and therefore we define the pseudoinverse (Moore-Penrose pseudoinverse) A^{-g} of A as the matrix that solves[1]

$$A^{-g}\boldsymbol{y} = \boldsymbol{a} \tag{3}$$

It can be shown, if A has the SVD eq.(1), that[2, 3]

$$A^{-g} = Vw^{-1}U^T \tag{4}$$

This is justified as

$$A^{-g}A = V^{-1}w^{-1}U^TUwV^T = I$$

2.3. Variance-Covariance Matrix C

To get the variance-covariance matrix $C = \left(A^TA\right)^{-1}$ we use the SVD eq.(1)

$$\begin{aligned} C = \left(A^TA\right)^{-1} &= \left[\left(UwV^T\right)^T\left(UwV^T\right)\right]^{-1} \\ &= \left(Vw^TU^TUwV^T\right)^{-1} \\ &= \left(Vw^2V^T\right)^{-1} = Vw^{-2}V^T \end{aligned} \tag{5}$$

2.4. Residual Error r

As discussed previously the solution to eq.(7) is in general the least-square solution and therefore not exact. We define the residual r therefore as

$$r = \|y - Aa\| \tag{6}$$

3. Computational Details

The problem to be solved is the following system of linear equations

$$\begin{aligned} a_1 - 2a_2 + 3a_3 &= 4 \\ 2a_1 - a_2 + 4a_3 &= 3 \\ -a_1 - 4a_2 + a_3 &= 7 \end{aligned} \tag{7}$$

The SVD was performed by the "numpy.linalg.svd" routine of the NumPy (V1.19.1) package for the Python (V3.8.5) programming language. The pseudo-inverse was calculated with the "numpy.linalg.pinv" routine of said package. It was also utilised for standard vector and matrix operations.
For this work a wrapper method named "svd" was written (see appendix A) that calculates the solution vector a, residual error r as well as the variance-covariance matrix C.

4. Results

Reformulating our system of linear equations eq.(7) yields the matrix equation

$$\underbrace{\begin{pmatrix} 4 \\ 3 \\ 7 \end{pmatrix}}_{\boldsymbol{y}} = \underbrace{\begin{pmatrix} 1 & -2 & 3 \\ 2 & -1 & 4 \\ -1 & -4 & 1 \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix}}_{\boldsymbol{a}}$$

The system matrix A is square, however, not of full rank and thus, singular[2]. That A is of rank 2, i.e. rank deficient, can be seen by diagonalising A

$$\begin{pmatrix} 1 & -2 & 3 \\ 2 & -1 & 4 \\ -1 & -4 & 1 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & -2 & 3 \\ 0 & 3 & -2 \\ 0 & 0 & 0 \end{pmatrix}$$

This means our system is over- and under-determined or mixed-determined[3]. The result of the SVD are the three matrices

$$U = \begin{pmatrix} -0.594 & -0.069 & -0.802 \\ -0.661 & -0.527 & 0.534 \\ -0.459 & 0.847 & 0.272 \end{pmatrix}$$

$$\operatorname{diag}(w) = \begin{pmatrix} 6.289 \\ 3.668 \\ 3.063E-16 \end{pmatrix} \rightarrow \begin{pmatrix} 6.289 \\ 3.668 \\ 0 \end{pmatrix}$$

$$V = \begin{pmatrix} -0.231 & -0.537 & -0.811 \\ 0.586 & -0.742 & 0.324 \\ -0.776 & -0.400 & 0.487 \end{pmatrix} \tag{8}$$

Because the matrix A is of rank 2 we only expect two singular values $w_i \neq 0$, however, due to the computational limit of float numbers the third singular value w_{33} is only close to 0 and was set manually $w_{33} \rightarrow 0$.
Next we can solve the initial problem eq.(7) by using eq.(3) resulting in

$$\boldsymbol{a} = \begin{pmatrix} -0.318 \\ -1.530 \\ 0.491 \end{pmatrix}$$

with a residual error $r = 0.267\,(16\ \%)$. Last but not least the variance-covariance matrix

$$C = \begin{pmatrix} 0.022 & 0.026 & 0.021 \\ 0.026 & 0.050 & 0.011 \\ 0.021 & 0.011 & 0.027 \end{pmatrix}$$

[1] Numpy - svd. `https://numpy.org/doc/stable/reference/generated/numpy.linalg.svd.html`.
[2] T. Arens, F. Hettlich, C. Karpfinger, U. Kockelkorn, K. Lichtenegger, and H. Stachel. *Mathematik*. Springer Spektrum, 3 edition, 2015.
[3] A. K. Steiner. *Methods of Modeling and Simulation*, 2020.

Appendix A

```
import numpy as np
from copy import deepcopy

def svd(A, y=None, tol=1E-12):

    U, w, V_T = np.linalg.svd(A)
    V = V_T.T
    w = np.diag(w)

    if y is None:
        return U, w, V

    w_inv = deepcopy(w)
    if tol is not None:
        w_inv[abs(w_inv) <= tol] = 0

    w_inv = np.linalg.pinv(w_inv)

    a = V @ np.linalg.pinv(w) @ U.T @ y
    r = np.linalg.norm(y - A @ a)
    C = V @ w_inv @ w_inv @ V_T

    return [[U, w, V], a, r, C]
```