Singular Value Decomposition

Source:

* [7.4: Singular Value Decompositions - Mathematics LibreTexts](https://math.libretexts.org/Bookshelves/Linear_Algebra/Understanding_Linear_Algebra_(Austin)/07%3A_The_Spectral_Theorem_and_singular_value_decompositions/7.04%3A_Singular_Value_Decompositions)
* *Singular Value Decomposition* PPT Binus Session 13 Advanced Linear Algebra
* *Singular Value Decomposition* Docs Binus Forum Session 13 Advanced Linear Algebra
* The *Singular Value Decomposition* *Theorem*

The theorem suggest that an *m x n* matrix *A* may be written as *A = UΣVT*, where U is an orthogonal *m x m* matrix, V is an orthogonal *n x n*  matrix, and Σis an  *m x n* matrix whose entries / elements are zero except for the singular values of *A* which appear in decreasing order on the diagonal.

Since a singular value decomposition of A gives a singular value decomposition of AT, specifically if *A =* *UΣVT*, then



If *A* is an *m x n* matrix, then:

1. ***ATA* is orthogonally diagonalizable**
2. **The eigenvalues of *ATA* are non-negative real numbers**
3. ***ATA* and *AAT*  have the same set of positive eigenvalues**

By definition, if *A* is an *m x n* matrix and λ1, λ2, ..., λn are the eigenvalues of *ATA* then the numbers

**σ1 = , σ2 = , …, σn =** , are called the singular values of the matrix *A*.

* Based on a theorem from linear algebra, the following is said:

A rectangular matrix *A* can be broken down into the product of three matrices:

1. an orthogonal *n x n* matrix *U* of left singular vectors
2. a diagonal *m x* n matrix *Σ* of singular values (σ1 ≥ σ2 ≥ … ≥ σn**)** on the diagonal and zeros on the rest
3. the transpose of an orthogonal *m x m* matrix V of right singular vectors

* Applications

1. Find the Singular Value Decomposition of the matrix

A =

Solution:

* Compute *AT A*

*AT A =*

* Compute the eigenvalues of A

  λ1 = 3,   λ2 = 1,   λ3 = 0

* Compute the eigenvectors that corresponds to each of the eigenvalues of A

v1 = , v2 = , v3 =

* Compute the orthogonal matrix V and VT

*V* = [v1, v2, v3] =

*VT* =

* The singular values of matrix A are σ1 = , σ2 = 1, and σ3 = 0, so rank(*A)* = 2 where the amount of ranks corresponds to the amount of non-zero singular values

Σ = =

* Compute the orthogonal matrix U

*U* = [u1, u2, u3]

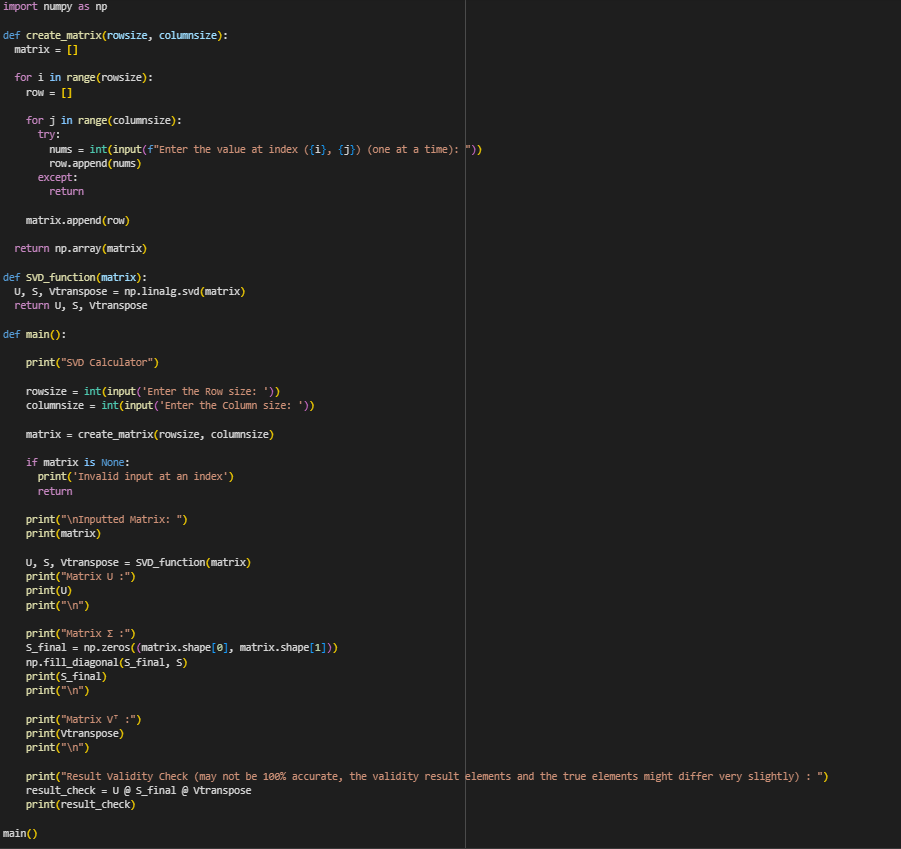
Where,

Normalize

Then,

*U =*

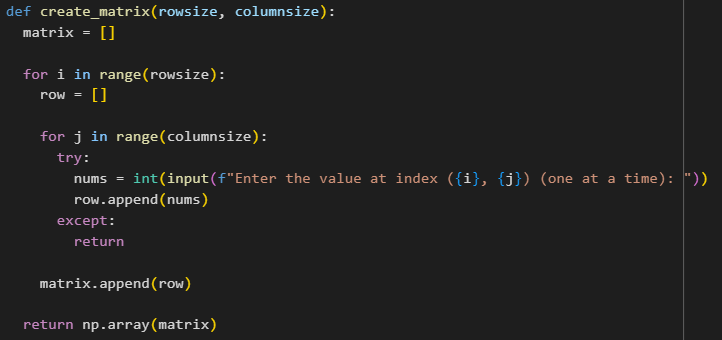
* The singular value decomposition for A is
* Python Code



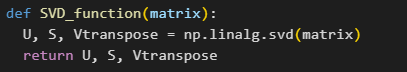
* Code Explanation:



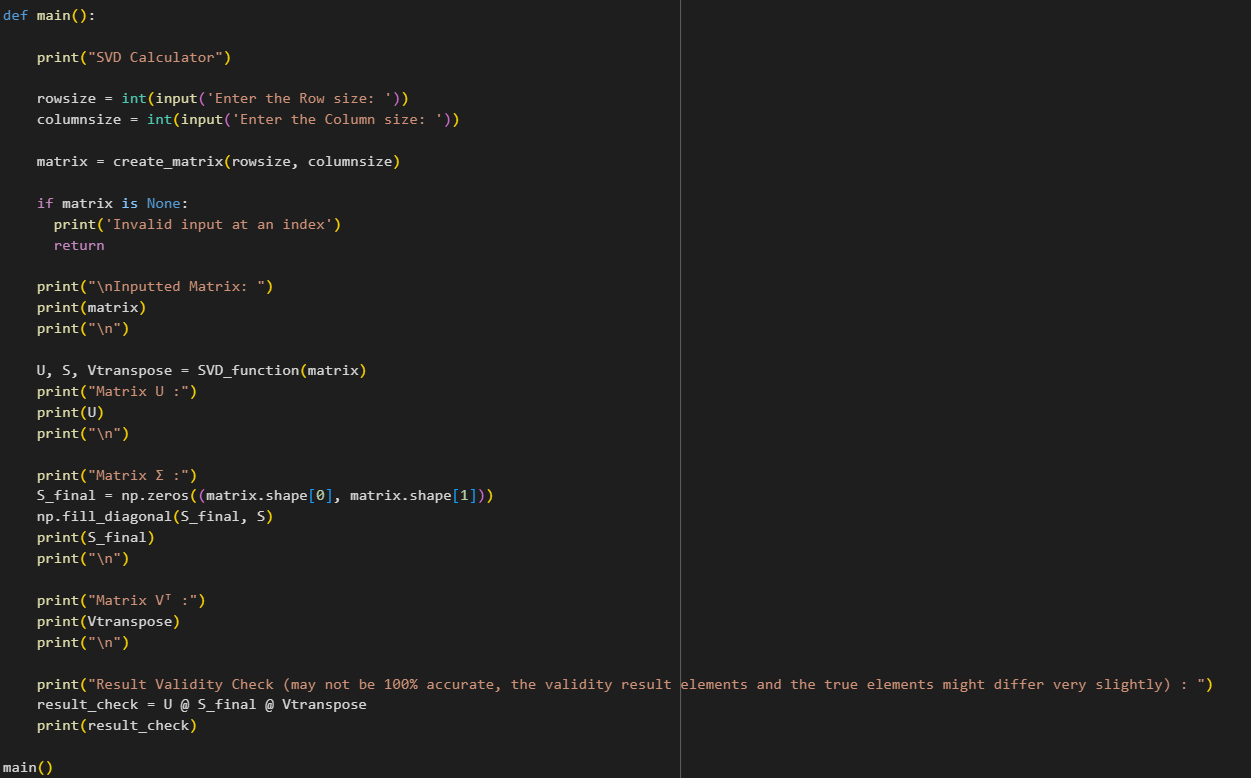
* Import the numpy library to make matrix operations easier, including the *Singular Value Decomposition* process



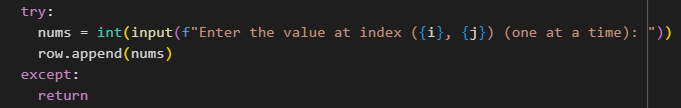
* This function allows the user to the input their desired matrix. It will create an empty array called “matrix” and then enter the for loop and iterate through it based on the number of rows and columns inputted. For every row, an empty array called “row” will be created and the user will be asked to input the necessary amount of elements based on the number of columns for each row. The try and except line is used to ensure that for each column, there will only be a maximum of 1 inputted value and not more. After an element is inputted, it will be added to the current row’s array and if the current row has already finished iterating through the columns, it will be added to the matrix. After the for loop is finished and each row and column has an element, the function will return the matrix as an np.array



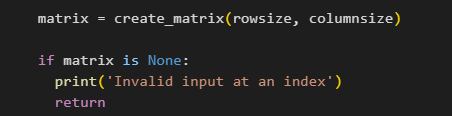
* This function will be the main point of the code since it will be the function that computes the *Singular Value Decomposition* of the matrixitself using np.linalg.svd. After that, it will return the corresponding matrices of U, Σ, and VT.



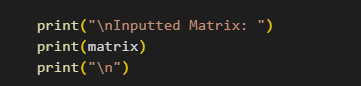
* This function will be the interface in which the user will see. First, the function will ask the user to input the number of rows and columns of the matrix. After that, it will call the previous function “create\_matrix” to create the matrix. To prevent an invalid input, like 1 column has more than one element, the try and except part from the “create\_matrix” function will validate the inputted elements



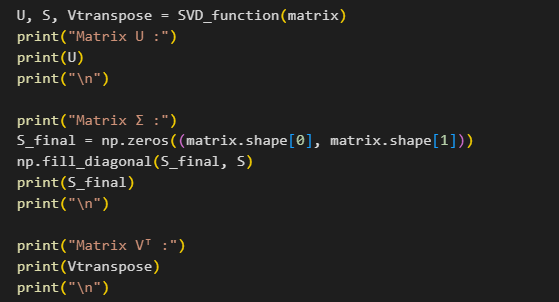
* If the inputted elements is more than 1 in a single column, it will cause an error and go to the “except” part and return nothing.

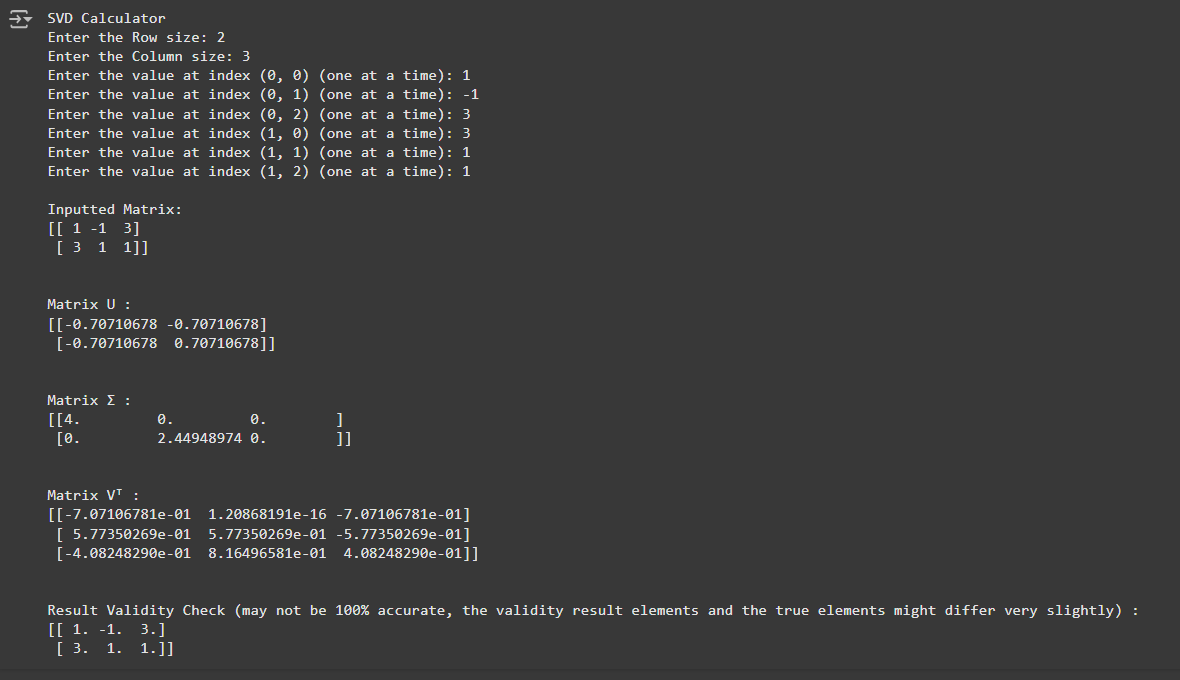


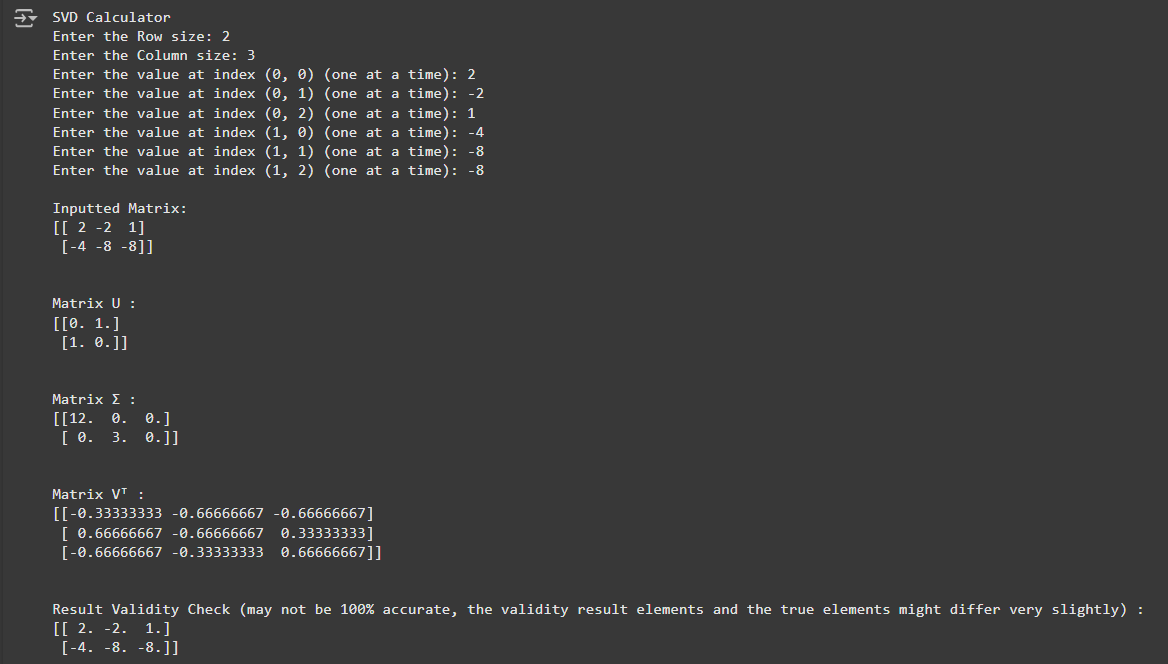
* This part will validate if the matrix is valid or not. If it’s not valid, it will print(‘Invalid input at an index) and terminate the program

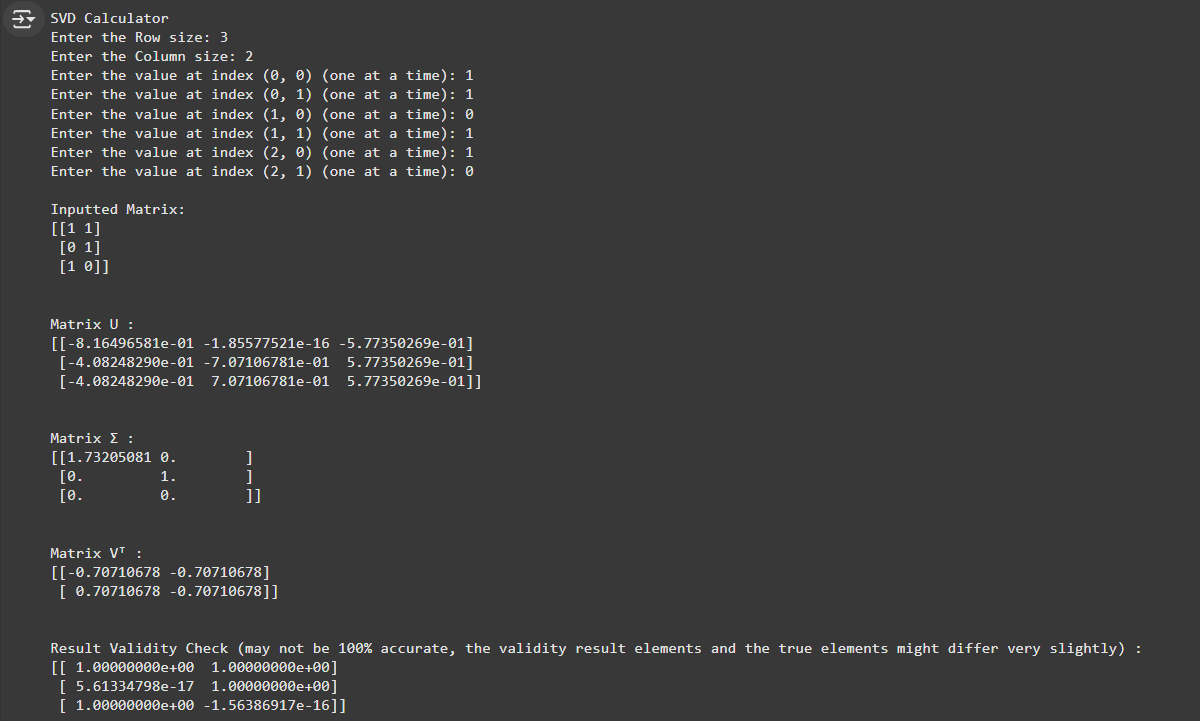


* This part ensure’s that the inputted elements is formatted correctly into the matrix form, as what the user’s want



* Variabel U, S, and Vtranspose will hold each of the matrices that corresponds to their respective symbol, that the function “SVD\_function” will return
* “S\_final” is used to store the elements of the S matrix into an *m x n* matrix, since the “SVD\_function” will return the matrix where the elements only contains the singular values so the dimensions of said matrix doesn’t match the original matrix
* The “np.zeros((matrix.shape[0], matrix.shape[1]))” is used to create a matrix, in this case called “S\_final” with the dimensions of the original matrix and the elements only consists of zeros
* “np.fill\_diagonal(S\_final, S)” changes the diagonal elements of “S\_final” to match the diagonal elements from the matrix S
* The validity check is made to make sure that the SVD result when multiplied back together, gives back the original matrix
* The “@” symbol is a special symbol that is used to do matrix multiplications
* Code Examples



* In some cases / matrices, the Validity check might not be 100% accurate, as in the validity matrix elements will differ very very slightly from the original matrix elements
* Edge cases (matrices with zero singular values)

