

# Singular Value Decomposition

An application to Big Data

---

Davide Sferrazza

January 13, 2023

Università degli Studi di Palermo

- 1 Singular Value Decomposition
  - What is it?
  - How can singular values be computed?
- 2 Finding eigenvalues and eigenvectors
  - QR Method
  - Improving QR Method using Givens rotation matrices
- 3 The Algorithm
  - Calculate  $U$  given  $V$
  - Code
- 4 Application to Big Data
  - Dimensionality reduction
- 5 References

# Singular Value Decomposition

---

# Definition of SVD

## Theorem

*Given a matrix  $A \in \mathbb{R}^{m \times n}$ , it can always be found a decomposition such that*

$$A = U \Sigma V^T \quad (1)$$

*where  $U \in \mathbb{R}^{m \times m}$ ,  $V \in \mathbb{R}^{n \times n}$  and  $\Sigma \in \mathbb{R}^{m \times n}$ .*

*$U$  and  $V$  are two orthogonal matrices and  $\Sigma$  is a diagonal matrix, namely:*

$$(\Sigma)_{ij} = \begin{cases} \sigma_i, & i = j \\ 0, & i \neq j \end{cases}$$

*where  $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_p \geq 0$ ,  $p = \min\{m, n\}$ .*

# Definition of SVD

The non-zero entries of  $\Sigma$ , denoted by  $\sigma_i$ , are called *singular values*.

They are arranged in a nonincreasing order by convention.

The column vectors  $\mathbf{u}_i$  of  $U$  are called *left singular vectors* and those  $\mathbf{v}_i$  of  $V$  are called *right singular vectors*.

Since in general  $m \neq n$ , we have:

$$A = \sum_{i=1}^p \mathbf{u}_i \sigma_i \mathbf{v}_i^T$$

# Definition of SVD

## Theorem

*If for some  $r$  such that  $1 \leq r < p$  we have*

$$\sigma_1 \geq \dots \geq \sigma_r > \sigma_{r+1} = \dots = \sigma_p = 0$$

*then*

- $\text{rank}(A) = r$
- $A = \sum_{i=1}^r \mathbf{u}_i \sigma_i \mathbf{v}_i^T$

It means that all other  $p - r$  dimensions of matrix  $A$  are linear combinations of the first  $r$ .

# Definition of SVD

## Lower rank approximation

Let  $A \in \mathbb{R}^{m \times n}$  be a matrix whose rank is  $\text{rank}(A) = r$ .

If for a fixed integer value  $k < r$  we define

$$A_k = \sum_{i=1}^k \sigma_i \mathbf{u}_i \mathbf{v}_i^T \quad (2)$$

and

$$\mathcal{B} = \{B \in \mathbb{R}^{m \times n} : \text{rank}(B) = k\}$$

then

$$\min_{B \in \mathcal{B}} \|A - B\|_2 = \|A - A_k\|_2 = \sigma_{k+1}$$

This result tell us that  $A_k$  represents the best approximation (considering the *spectral norm*) of rank  $k$  of matrix  $A$ .

# Singular values computation

To compute the singular values, consider the transpose of  $A$  given its decomposition:

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

The symmetric matrix  $A^T A$  is equal to:

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

Furthermore, this equation can be written as:

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

It means that the diagonal entries of the square matrix  $\Sigma^T \Sigma$ , which are the square of the singular values, are the eigenvalues of matrix  $A^T A$  and  $V$  is the matrix of eigenvectors.



# Singular values computation

Similarly, consider the product of  $AA^T$ . It is equal to:

$$AA^T = (U\Sigma V^T)(V\Sigma^T U^T) = U\Sigma\Sigma^T U^T$$

Which means that:

$$AA^T U = U\Sigma\Sigma^T$$

Hence  $U$  is the matrix of eigenvectors of  $AA^T$ .

Since  $\text{rank}(A) = r$ , only the first  $r$  eigenvalues of  $AA^T$  and  $A^T A$  are non-zero.

## Finding eigenvalues and eigenvectors

---

A possible method to find eigenvalues and eigenvectors of a matrix is based on  $QR$  decompositions and this theorem:

### Theorem

*Suppose  $A \in \mathbb{R}^{n \times n}$  is a matrix having eigenvalues  $\lambda_1, \lambda_2, \dots, \lambda_n$  satisfying*

$$|\lambda_1| > |\lambda_2| > \dots > |\lambda_n| \quad (3)$$

*then the following sequence for  $A_1 = A$  and  $k = 1, 2, \dots$*

$$\begin{cases} A_k = Q_k R_k \\ A_{k+1} = R_k Q_k \end{cases} \quad (4)$$

*converges to an upper triangular matrix where  $(A_k)_{ii} = \lambda_i, i = 1, 2, \dots, n$ .*

*In case (3) is not satisfied, this sequence converges to a triangular matrix with square blocks of order at most 2 along the diagonal.*

*If  $A$  is symmetric, then the sequence converges to a diagonal matrix.*

# QR Method

So a basic implementation would be like this:

```
1 while err > toll
2     [Q, R] = qr(A);
3     A = R * Q;
4
5     err = max( max( tril(A, -1) ) );
6 end
```

This method can be sped up by using a technique called *shifting*:

```
1 n = length( A );
2 while err > toll
3     % A(n, n) is a usual choice, it can be any real number
4     T = A(n, n) * eye(n);
5     [Q, R] = qr( A - T );
6     A = R * Q + T;
7
8     err = max( max( tril(A, -1) ) );
9 end
```

In our case, this method must be applied to  $AA^T$  and  $A^T A$ , so if (4) converges then we have a diagonal matrix.

Now, consider the diagonalization of  $B$ , where  $B = AA^T$  or  $B = A^T A$ :

$$B = P\Lambda P^{-1} = P\Lambda P^T$$

As  $B$  can be factored using (4), the matrix containing the eigenvectors must be equal to:

$$P = \prod_i Q_i = Q_1 Q_2 Q_3 \cdots$$

Hence, for every iteration  $B_k = Q_k R_k$  and  $B_{k+1} = R_k Q_k$ , requiring each step to have a computational cost equal to  $O(\frac{2n^3}{3})$ .

# Hessenberg Reduction

To achieve a lower computational cost, one solution consists in transforming  $B$  in a similar tridiagonal matrix using Householder matrices.

A triangular matrix can be obtained because  $B$  is symmetric.

In general, the process of transforming a matrix in a similar matrix using Householder matrices allows us to obtain a matrix in which  $B_{ij} = 0$ , for all  $i > j$ . Such a matrix is called a *Hessenberg matrix*.

Consequently, after  $B$  has been transformed, each step of the previous code can be realized using *Givens rotation matrices*.

# Givens rotation matrices

A *Givens rotation* can be represented with the following matrix:

$$G_{ij} = \begin{pmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & c & \cdots & -s & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & s & \cdots & c & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{pmatrix} \begin{matrix} \leftarrow i \\ \\ \leftarrow j \\ \\ \end{matrix}$$

$\begin{matrix} \uparrow & \uparrow \\ i & j \end{matrix}$

where  $c = \cos \theta$  and  $s = \sin \theta$ , for a particular value of  $\theta \in [0, 2\pi]$ .

A rotation occurs in the plane spanned by the two coordinates axes  $i$  and  $j$ .

## Givens rotation matrices

Using the following values:

$$\cos \theta = \frac{|b_{ii}|}{\sqrt{b_{ii}^2 + b_{ji}^2}}$$
$$\sin \theta = \operatorname{sign}\left(\frac{b_{ji}}{b_{ii}}\right) \frac{|b_{ji}|}{\sqrt{b_{ii}^2 + b_{ji}^2}}$$

when  $B$  gets left multiplied by  $G_{ij}$ , the final matrix will have the element in position  $(j, i)$ <sup>1</sup> equal to 0.

Therefore, instead of factoring  $B$  at each step by the  $QR$  method, if  $B$  is tridiagonal, we can simply iterate  $n - 1$  times by constructing  $G_{k\ k+1}$  and eliminating the elements in position  $(k + 1, k)$  and  $(k, k + 1)$ .

---

<sup>1</sup>Note that the indices are reversed



# Pseudocode for finding eigenvalues and eigenvectors

A pseudocode of what to do should clarify what we have done so far (excluding shifting for brevity):

```
 $B, P \leftarrow \text{hessemberg}(B)$   
while error > toll do  
  for  $k = 1, 2, \dots, n$  do  
    construct  $G_{k\ k+1}$   
     $B \leftarrow G_{k\ k+1} B$   
  end for  
  for  $k = 1, 2, \dots, n$  do  
     $B \leftarrow B G_{k\ k+1}^T$   
     $P \leftarrow P G_{k\ k+1}^T$   
  end for  
  update error  
end while
```

The function implementing the Hessemberg reduction must also return the matrix used for the transformation, meaning that  $B = P H P^T$ .

Note that  $P$  is updated in this way because  $Q_i = G_{12}^i G_{23}^i \cdots G_{n-1\ n}^i$ , where  $i$  denotes the generic iteration for error minimization.

# The Algorithm

---

## Calculation of left singular vectors

Suppose we start computing  $V$  by the method explained in the previous slides.

We apply the method to  $A^T A$  and get  $P = V$  and  $B = \Sigma^2$ .

Once we have taken the square root of the singular values, by explicitly writing  $\Sigma$ , the matrix containing the left singular vectors can be calculated as:

$$A = U\Sigma V^T \Rightarrow U = AV\Sigma^{-1}$$

Unlike (1),  $U$  is a rectangular matrix, and  $\Sigma$  and  $V$  are square matrices. The algorithm is now complete. The next slides will illustrate the code.

# Function to calculate Givens rotation matrices

```
1 function [c, s] = givens(A, i, j)
2 %GIVENS Function to compute cos and sin of Gij
3 %   Given a matrix A, this function returns a vector containing the values
4 %   of givens rotation matrix Gij such that Gij * A is equal to A except
5 %   for the element (j, i), which will be zero.
6     c = abs( A(i, i) ) / sqrt( A(i, i)^2 + A(j, i)^2 );
7     s = - sign( A(j, i) / A(i, i) ) * abs( A(j, i) ) / sqrt( A(i, i)^2 + A
      (j, i)^2 );
8 end
```

# Function to calculate the Hessemberg reduction i

```
1 function [A, P] = hessemberg(A)
2 %HESSEMBERG Fuction to compute the Hessember reduction of a matrix
3 % Given a matrix A, this function trasform A in a similar hessemberg
4 % matrix. It will be produced as output the hessember matrix and the
5 % matrix P used for the transformation.
6 n = size(A, 2);
7 P = eye(n);
8 for k=1:n-2
9     sigma = sign(A(k+1, k)) * norm(A(k+1:n, k));
10    v = [sigma + A(k+1, k); A(k+2:n, k)];
11    beta = 1 / (sigma * (sigma + A(k+1, k)));
12    for j=k:n
13        tau = beta * (v' * A(k+1:n, j));
14        A(k+1:n, j) = A(k+1:n, j) - tau * v;
15    end
16    for j=1:n
17        tau = beta * (A(j, k+1:n) * v);
18        A(j, k+1:n) = A(j, k+1:n) - tau * v';
```

## Function to calculate the Hessemberg reduction ii

```
19         tau = beta * (P(j, k+1:n) * v);  
20         P(j, k+1:n) = P(j, k+1:n) - tau * v';  
21     end  
22 end  
23 end
```

# Function to calculate singular values and singular vectors i

```
1 function [M, H] = singular_vectors(A, toll, left)
2 %SINGULAR_VECTORS Function to compute the left or right singular vectors
3 % Given a matrix A and a tollerance for the stopping criterion, this
4 % function computes the left or right singular vectors of A and its
5 % corrsponding singular values. If left is true then U is calculated, V
6 % otherwise.
7 if left
8     [H, P] = hesseberg(A * A. ');
9 else
10    [H, P] = hesseberg(A.' * A);
11 end
12
13 n = length(H);
14
15 G = zeros(n - 1, 2);
16 G_aux = zeros(2);
17
18 M = P;
```

# Function to calculate singular values and singular vectors ii

```
19     err = toll + 1;
20
21     while err > toll
22         H1 = H;
23         T = H(n, n) * eye(n);
24
25         H = H - T;
26
27         for k = 1:n-1
28             [G(k, 1), G(k, 2)] = givens(H, k, k+1);
29             G_aux(1, 1) = G(k, 1);
30             G_aux(1, 2) = - G(k, 2);
31             G_aux(2, 1) = G(k, 2);
32             G_aux(2, 2) = G(k, 1);
33
34             H(k:k+1, k:n) = G_aux * H(k:k+1, k:n);
35         end
36
37         for k = 1:n-1
```



## Function to calculate singular values and singular vectors iii

```
38     G_aux(1, 1) = G(k, 1);
39     G_aux(1, 2) = G(k, 2);
40     G_aux(2, 1) = - G(k, 2);
41     G_aux(2, 2) = G(k, 1);
42
43     H(1:k+1, k:k+1) = H(1:k+1, k:k+1) * G_aux;
44     M(1:n, k:k+1) = M(1:n, k:k+1) * G_aux;
45 end
46
47 H = H + T;
48
49 err = norm(diag(H - H1), 1);
50 end
51
52 H = sqrt( diag( diag( H ) ) );
53
54 discard_imag = 10^-5;
55
56 if all( imag( diag(H) ) < discard_imag )
```

# Function to calculate singular values and singular vectors iv

```
57     H = real(H);  
58 end  
59 if all( imag( diag(M) ) < discard_imag )  
60     M = real(M);  
61 end  
62 end
```

# Function to calculate SVD i

```
1 function [U, S, V] = custom_svd(A, toll)
2 %CUSTOM_SVD Function to compute the SVD decomposition of a matrix
3 %   Given a matrix A as input and a tollerance for the stopping criterion,
4 %   this function computes the SVD decomposition of A.
5     [m, n] = size(A);
6     [V, S1] = singular_vectors(A, toll, false);
7
8     [New_Diag, s_order] = sort(diag(S1), 'descend');
9
10    S = diag(New_Diag);
11
12    new_order = 1:n;
13    for h = 1:length(s_order)
14        if h ~= s_order(h)
15            new_order(h) = s_order(h);
16        end
17    end
18
```

## Function to calculate SVD ii

```
19     V = V(:, new_order);
20
21     arr = New_Diag;
22     arr(arr > 0) = NaN;
23     [zero, index] = max(arr, [], 'omitnan');
24     if index ~= 1
25         arr = New_Diag(1 : index - 1);
26     else
27         arr = New_Diag;
28     end
29
30     S1 = diag( 1./ arr );
31     S1(n, 1) = 0;
32     S1(1, n) = 0;
33     U = A*V*S1;
34 end
```

# Application to Big Data

---

# Dimensionality reduction

The SVD decomposition can be applied to Big Data in order to reduce the dimensionality of data sets.

As an example, consider the data set [5] containing 33 different attributes for 145 students, which includes student ID, personal information and data, and higher education performance ratings<sup>2</sup>.

Dimensionality can be reduced as follows:

1. truncate the SVD retaining top- $k$  singular values,  $A \approx U_k \Sigma_k V_k^T$ ;
2. compute the modified data set as  $D' = DV_k = U_k \Sigma_k$ .

---

<sup>2</sup><https://archive.ics.uci.edu/ml/datasets/Higher+Education+Students+Performance+Evaluation+Dataset>

# Dimensionality reduction

$D'$  represents a new data set that corresponds to the projection of the data set  $D$  on a  $k$ -dimensional basis system of right singular vectors.

## Number of singular values

As indicated in [3], a usual choice of  $k$  is dictated by maintaining enough singular values to make up 90% of the energy in  $\Sigma$ .

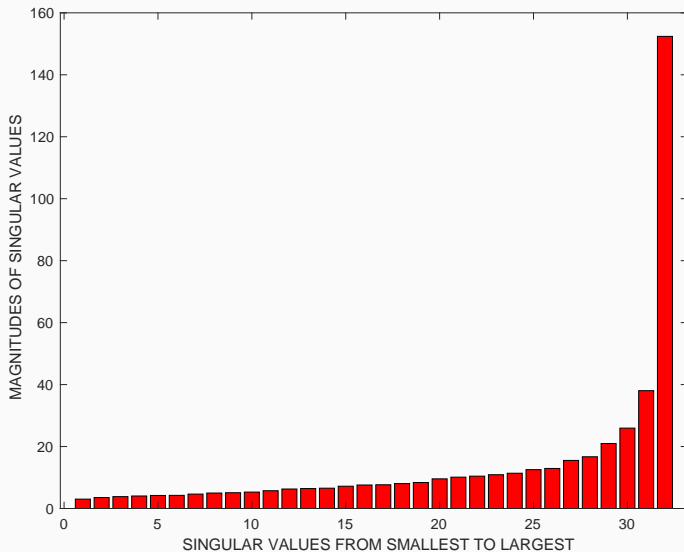
The total energy of a data set is:

$$E(\Sigma) = \sum_{i=1}^r \sigma_i^2$$

where  $r = \text{rank}(D)$ .

By plotting the singular values we can get an insight of how they are distributed and how many of them we can retain.

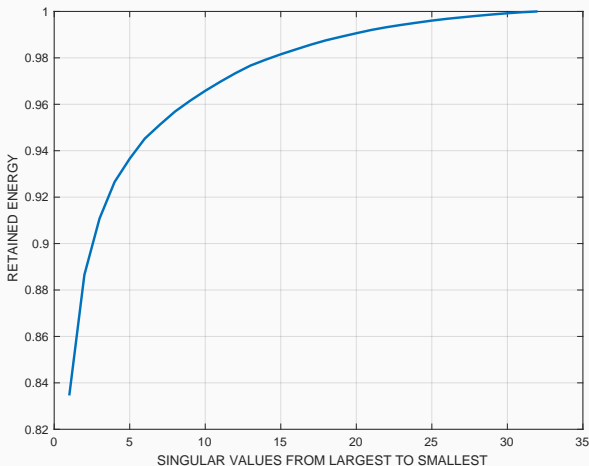
# Magnitudes of singular values





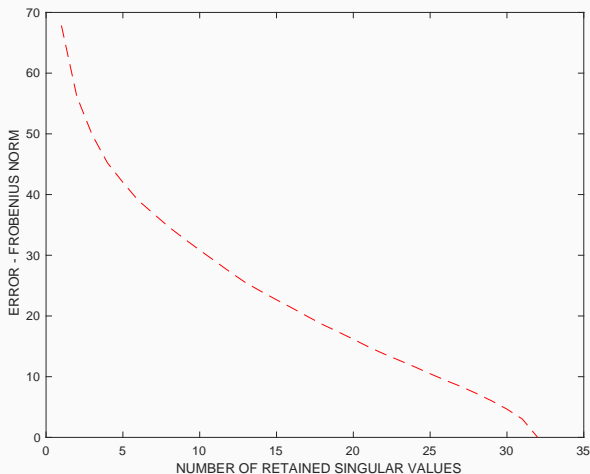
## Retained energy

From what we can see, with 3 singular values we can get about 25,346 of energy, which is enough to exceed 90% of  $E(\Sigma) = 27,830$ .



## Decreasing error as singular values increase

By increasing the number of retained singular values, the following graph is obtained:



## References

---

## References

---

- [1] G. Monegato, *Metodi e algoritmi per il calcolo numerico*. Clut, 2008.
- [2] C. C. Aggarwal *et al.*, *Data mining: the textbook*. Springer, 2015, vol. 1.
- [3] J. Leskovec, A. Rajaraman, and J. D. Ullman, *Mining of massive data sets*. Cambridge university press, 2020.
- [4] C. F. Van Loan and G. Golub, “Matrix computations (johns hopkins studies in mathematical sciences),” *Matrix Computations*, 1996.

- [5] N. Yılmaz and B. Sekeroglu, “Student performance classification using artificial intelligence techniques,” in *10th International Conference on Theory and Application of Soft Computing, Computing with Words and Perceptions - ICSCCW-2019*, R. A. Aliev, J. Kacprzyk, W. Pedrycz, M. Jamshidi, M. B. Babanli, and F. M. Sadikoglu, Eds., Cham: Springer International Publishing, 2020, pp. 596–603, ISBN: 978-3-030-35249-3.
- [6] D. Dua and C. Graff, *UCI machine learning repository*, 2017. [Online]. Available: <http://archive.ics.uci.edu/ml>.