Republic of Iraq

Ministry of Higher Education

and Scientific Research

University of Babylon

College of Education For Pure Sciences

Department of Mathematics

SINGULAR VALUE DECOMPOSITION

A graduation project submitted to the **Department of mathematics**, in partial fulfillment for the requirements for the award of the degree of Bachelor of **Mathematics**

BY

HUSSEIN ALAA HAMZAH

SUPERVISED BY

DR. LAMIS HAMMOUD

BABYLON, IRAQ

2023

بسم الله الرحمن الرحيم

((وَلَقَدْ آتَيْنَا دَاوُودَ وَسُلَيْمَانَ عِلْمًا ۖ وَقَالَا الْحَمْدُ لِلَّهِ الَّذِي فَضَّلَنَا عَلَىٰ كَثِيرٍ مِّنْ عِبَادِهِ

الْمُؤْمِنِينَ)) النمل اية(١٥)

صدق الله العلي العظيم

DEDICATION

To my parents and to our family who made this accomplishment possible.

ACKNOWLEDGEMENT

*Praise be to **ALLAH** who enabled us to complete this work under his benediction.*

*Cordial thanks and deepest gratitude to our supervisor (**DR.Lamis Hammoud**) for being assistance and for the continuous encouragement and suggestions from him which helped us to successfully complete this project.*

*In addition, we would like to express my thanks to the head of the Mathematics Department (**Dr. Azal Musa Jafar**) for his continuous support and encouragement.*

We would like to express our thanks to our University for their encouragements and anyone helped us.

Finally, sincere gratitude and appreciation and love to our families for their encouragements and supports and for all things they have given us.

ABSTRACT

Singular Value Decomposition (SVD) is a powerful matrix factorization technique that finds its applications in various domains such as linear algebra, signal processing, image compression, recommender systems, and data analysis. This comprehensive overview aims to provide a detailed understanding of SVD, including its mathematical formulation, computation methods, properties, and applications. Additionally, it discusses related concepts and variations of SVD, and explores the significance of SVD in modern data-driven technologies.

LIST OF FIGURES

.

Chapter One

Singular Value Decomposition

1-1 Introduction

Singular Value Decomposition (SVD) is a fundamental technique in linear algebra and numerical analysis. It plays a crucial role in various applications, including signal processing, image compression, data analysis, and machine learning. The history of SVD can be traced back to the early 19th century, with significant contributions from several mathematicians and researchers.

The concept of SVD emerged from the study of matrices and their properties. In the early 1800s, mathematicians were exploring the properties of quadratic forms and seeking ways to diagonalize them. Carl Friedrich Gauss made notable advancements in this field, and his work laid the foundation for later developments in matrix factorization techniques.

The modern formulation of SVD was introduced by Eugenio Beltrami in the late 19th century. Beltrami proposed a method for decomposing a matrix into three components: two orthogonal matrices and a diagonal matrix. However, the term "Singular Value Decomposition" was not explicitly used until much later.

In the early 20th century, significant contributions to SVD were made by mathematicians such as David Hilbert and Wilhelm Jordan. Hilbert studied the properties of integral equations and their associated matrices, which led to the formulation of what is now known as the singular value decomposition.

Further advancements in SVD came in the mid-20th century when several researchers independently worked on related problems. In 1936, Hans Weyl introduced the concept of the generalized singular value decomposition, which extended the theory of SVD to arbitrary matrices.

During the 1960s, Alan Turing and John W. Wilkinson made significant contributions to the practical implementation of SVD algorithms. They developed efficient numerical algorithms for computing the SVD of matrices, paving the way for its widespread use in scientific and engineering applications.

Since then, SVD has become a powerful tool in various fields, including image and signal processing, data compression, recommendation systems, and latent semantic analysis. Its ability to decompose a matrix into its constituent components provides valuable insights into the underlying structure of the data.

Today, SVD is an integral part of many numerical libraries and software packages, making it readily available for researchers, engineers, and data scientists. It continues to be an active area of research, with ongoing developments and refinements to improve its computational efficiency and extend its applications.

.

.

1.2 Mathematical Formulation

1.2.1 Matrix Representation

To understand SVD we need to first understand the *Eigenvalue Decomposition* of a matrix. We can think of a matrix **A** as a transformation that acts on a vector **x** by multiplication to produce a new vector **Ax**. We use $[\mathbf{A}]_{ij}$ or a_{ij} to denote the element of matrix **A** at row i and column j. If **A** is an $m\times p$ matrix and **B** is a $p\times n$ matrix, the matrix product **C**=**AB** (which is an $m\times n$ *matrix*) is defined as:

$$[\boldsymbol{c}]_{ij} = c_{ij} = \Sigma_{k=1}\ \alpha_{ik}b_{kj}$$

For example, the rotation matrix in a 2-d space can be defined as:

$$A = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

This matrix rotates a vector about the origin by the angle θ (with counterclockwise rotation for a positive θ). Another example is the stretching matrix **B** in a 2-d space which is defined as:

$$B = \begin{bmatrix} k & 0 \\ 0 & 1 \end{bmatrix}$$

This matrix stretches a vector along the *x*-axis by a constant factor *k* but does not affect it in the *y*-direction. Similarly, we can have a stretching matrix in *y*-direction:

$$c = \begin{bmatrix} 1 & 0 \\ 0 & k \end{bmatrix}$$

As an example, if we have a vector

$$x = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

then **y**=**Ax** is the vector which results after rotation of **x** by θ, and **Bx** is a vector which is the result of stretching **x** in the *x*-direction by a constant factor *k*.

Here the rotation matrix is calculated for $\theta=30^0$ and in the stretching matrix *k*=3. **Y** is the transformed vector of **x**.

By Python

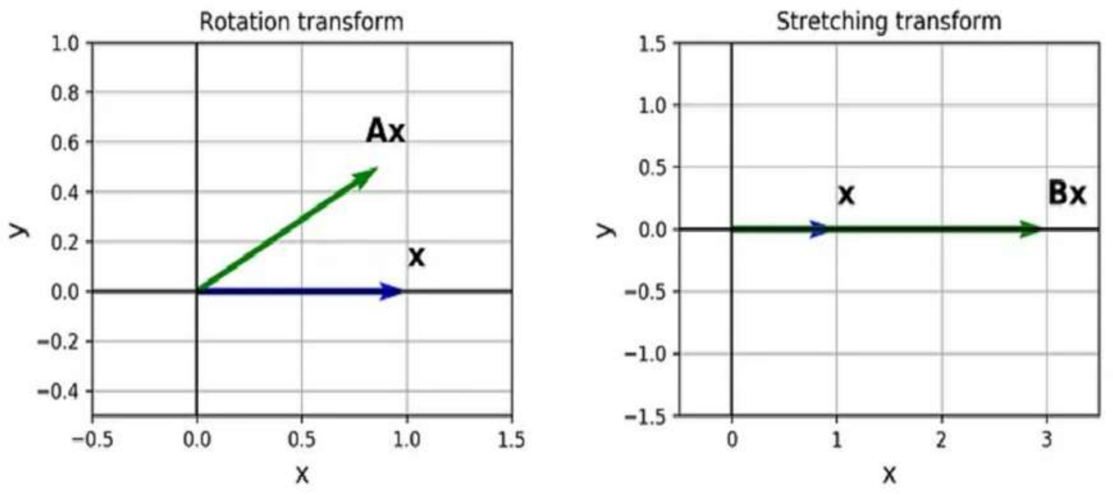

Figure 1

Now we are going to try a different transformation matrix. Suppose that

$$A = \begin{bmatrix} 3 & 2 \\ 0 & 2 \end{bmatrix}$$

However, we don't apply it to just one vector. Initially, we have a circle that contains all the vectors that are one unit away from the origin. These vectors have the general form of

$$x = \begin{bmatrix} xi \\ yi \end{bmatrix} \text{ where } x^2 i + y^2{}_i = 1$$

Now we calculate **t**=**Ax**. So **t** is the set of all the vectors in **x** which have been transformed by **A**.

Figure 2 shows the plots of **x** and **t** and the effect of transformation on two sample vectors **x1** and **x2** in **x**.

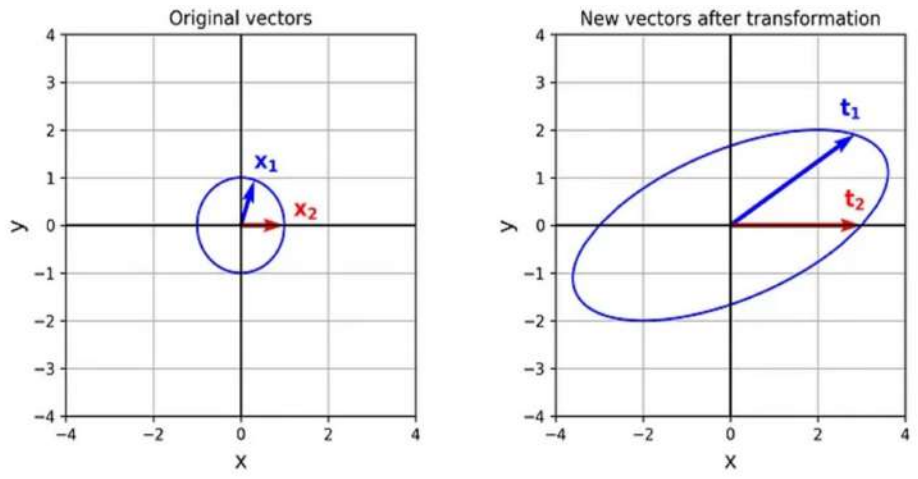

figure 2

The initial vectors (**x**) on the left side form a circle as mentioned before, but the transformation matrix somehow changes this circle and turns it into an ellipse.
The sample vectors **x1** and **x2** in the circle are transformed into **t1** and **t2** respectively. So:

$$t_1 = Ax_1 \qquad t_2 = Ax_2$$

1.2.2 Eigenvalues and Eigenvectors

A vector is a quantity which has both magnitude and direction. The general effect of matrix **A** on the vectors in **x** is a combination of rotation and stretching. For example, it changes both the direction and magnitude of the vector **x1** to give the transformed vector **t1**. However, for vector **x2** only the magnitude changes after transformation. In fact, **x2** and **t2** have the same direction. Matrix **A** only stretches **x2** in the same direction and gives the vector **t2** which has a bigger magnitude. The only way to change the magnitude of a vector without changing its direction is by multiplying it with a scalar. So if we have a vector **u**, and *λ* is a scalar quantity then *λ***u** has the same direction and a different magnitude. So for a vector like **x2** in figure 2, the effect of multiplying by **A** is like multiplying it with a scalar quantity like *λ*.

$$t_2 = Ax_2 = \lambda x_2$$

This is not true for all the vectors in **x**. In fact, for each matrix **A,** only some of the vectors have this property. These special vectors are called the eigenvectors of **A** and their corresponding scalar quantity λ is called an eigenvalue of **A** for that eigenvector. So the eigenvector of an ***n***×***n*** matrix **A** is defined as a nonzero vector **u** such that:

$$Au = \lambda u$$

where *λ* is a scalar and is called the eigenvalue of **A**, and **u** is the eigenvector corresponding to *λ*. In addition, if you have any other vectors in the form of *a***u** where *a* is a scalar, then by placing it in the previous equation we get:

$$\mathbf{A}(a\mathbf{u}) = a\mathbf{Au} = a\,\lambda\mathbf{u} = \lambda(a\mathbf{u})$$

which means that any vector which has the same direction as the eigenvector **u** (or the opposite direction if *a* is negative) is also an eigenvector with the same corresponding eigenvalue.

For example, the eigenvalues of

$$B = \begin{bmatrix} -1 & 1 \\ 0 & 2 \end{bmatrix}$$

are *λ1=-1* and *λ2=-2* and their corresponding eigenvectors are:

$$u_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad u_2 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

and we have:

$$Bu_1 = \lambda_1 u_1 \qquad Bu_2 = \lambda_2 u_2$$

This means that when we apply matrix **B** to all the possible vectors, it does not change the direction of these two vectors (or any vectors which have the same or opposite direction) and only stretches them. So for the eigenvectors, the matrix multiplication turns into a simple scalar multiplication.

1.2.3 Transpose

The transpose of the column vector **u (**which is shown by **u** superscript T) is the row vector of **u** (in this article sometimes I show it as $\mathbf{u}^T$). The transpose of an $m \times n$ matrix **A** is an $n \times m$ matrix whose columns are formed from the corresponding rows of **A**. For example if we have

$$C = \begin{bmatrix} 5 & 4 & 2 \\ 7 & 1 & 9 \end{bmatrix}$$

then the transpose of **C** is:

$$c^T = \begin{bmatrix} 5 & 7 \\ 4 & 1 \\ 2 & 9 \end{bmatrix}$$

So the transpose of a row vector becomes a column vector with the same elements and vice versa. In fact, the element in the *i*-th row and *j*-th column of the transposed matrix is equal to the element in the *j*-th row and *i*-th column of the original matrix. So

$$[A^T]_{ij} = [A]_{ji}$$

The transpose has some important properties. First, the transpose of the transpose of A is A. So:

$$(A^T) = A$$

In addition, the transpose of a product is the product of the transposes in the reverse order.

$$(AB)^T = B^T A^T$$

To prove it remember the matrix multiplication definition:

$$[AB]ij = \Sigma_{k=1}\ a_{ik}b_{kj}$$

and based on the definition of matrix transpose, the left side is:

$$[(AB)^T]_{ij} = [AB]_{ji} = \Sigma_{\ k=1}\ a_{jk}b_{ki}$$

and the right side is

$$\mathbf{[B^T A^T]_{ij} = \Sigma_{k=1}[B^T]_{ik}[A^T]_{jk} = \Sigma_{k=1}[B]_{ki}[A]_{kj} = \Sigma_{k=1}a_{jk}b_{ki}}$$

so both sides of the equation are equal.

1.2.4 Dot product

The dot product v • w of two vectors and the length $\|\, \boldsymbol{v} \| = \sqrt{\boldsymbol{v.u}}$

1.2.5 Partitioned matrix

When calculating the transpose of a matrix, it is usually useful to show it as a partitioned matrix.

1.2.6 A Basis for a Vector Space

A basis for a vector space is a sequence of vectors with two properties The basis vectors are linearly independent and they span the space.

Chapter Two

Singular Value Decomposition (SVD)

2.1 *SVD*

Let A be an $m \times n$ matrix and rank A = r. So the number of non-zero singular values of A is r. Since they are positive and labeled in decreasing order, we can write them as

$$\sigma_1 \geq \sigma_2 \geq \dots \sigma_n$$

$$v_1 \qquad v_2 \quad \dots v_n$$

where

$$\sigma_{r+1} = \sigma_{r+2} = \sigma_n = 0$$

We know that each singular value $\boldsymbol{\sigma_i}$ is the square root of the $\boldsymbol{\lambda_i}$ *(*eigenvalue of $A^T A$), and corresponds to an eigenvector vi with the same order. Now we can write the *singular value decomposition* of A as:

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

where V is an $n \times n$ matrix that its columns are vi. So:

$$V = [V_1 \;\; V_2 \quad \dots \quad v_n]$$

We call a set of orthogonal and normalized vectors an *orthonormal* set. So the set {vi} is an orthonormal set. A matrix whose columns are an orthonormal set is called an ***orthogonal matrix***, and V is an orthogonal matrix.

Σ is an $m \times n$ diagonal matrix of the form:

$$\boldsymbol{\Sigma} = \underbrace{\left.\begin{bmatrix} \sigma_1 & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & \sigma_2 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \dots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & \sigma_r & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 \end{bmatrix}\right\}}_{\text{n columns}} \text{m rows}$$

So we first make an $r \times r$ diagonal matrix with diagonal entries of $\sigma 1, \sigma 2, \dots, \sigma r$. Then we pad it with zero to make it an $m \times n$ matrix.

We also know that the set {Av1, Av2, …, Avr} is an orthogonal basis for *Col* A, and σi = ||Avi||. So we can normalize the Avi vectors by dividing them by their length:

$$\mathbf{u}_\mathbf{i} = \frac{\boldsymbol{A\,vi}}{||\boldsymbol{AVi}||} = \frac{\boldsymbol{A\,vi}}{||\boldsymbol{\sigma i}||}\ \mathbf{1<i<r}$$

Now we have a set {**u1**, **u2**, …, $\mathbf{u}_\mathbf{r}$} which is an orthonormal basis for Ax which is r dimensional. We know that A is an $m \times n$ matrix, and the rank of **A** can be m at most (when all the columns of **A** are linearly independent). Since we need an $m \times m$ matrix for **U**, we add *(m-r)* vectors to the set of **ui** to make it a normalized basis for an mdimensional space $\mathbf{R}^m$ (There are several methods that can be used for this purpose. For example we can use the ***Gram-Schmidt Process***. However, explaining it is beyond the scope of this article). So now we have an orthonormal basis {**u1**, **u2**, … ,$\mathbf{u}_\mathbf{m}$}. These vectors will be the columns of **U** which is an orthogonal $m \times m$ matrix

$$\mathbf{U} = [\mathbf{u}_1 \quad \mathbf{u}_2 \quad \dots \quad \mathbf{u}_\mathbf{m}]$$

So in the end, we can decompose A as

$$\mathbf{A} = \begin{bmatrix}\mathbf{u_1} & \mathbf{u_2} & \dots & \mathbf{u_m}\end{bmatrix} \begin{bmatrix} \sigma_1 & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & \sigma_2 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \dots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & \sigma_r & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 \end{bmatrix} \begin{bmatrix} \mathbf{v_1}^T \\ \mathbf{v_2}^T \\ \vdots \\ \mathbf{v_n}^T \end{bmatrix}$$

To better understand this equation, we need to simplify it:

$$\mathbf{A} = \begin{bmatrix}\mathbf{u_1} & \mathbf{u_2} & \dots & \mathbf{u_m}\end{bmatrix} \left.\begin{bmatrix} \sigma_1\mathbf{v_1}^T \\ \sigma_2\mathbf{v_2}^T \\ \vdots \\ \sigma_r\mathbf{v_r}^T \\ 0 \\ \vdots \\ 0 \end{bmatrix}\right\} \text{m rows}$$
$$= \sigma_1\mathbf{u_1}\mathbf{v_1}^T + \sigma_2\mathbf{u_2}\mathbf{v_2}^T + \dots + \sigma_r\mathbf{u_r}\mathbf{v_r}^T$$

We know that σ_i is a scalar; $\mathbf{u_i}$ is an m-dimensional column vector, and **vi** is an **n-** ndimensional column vector. So each $\sigma_i\mathbf{u_i}\ \mathbf{v_i}^T$ is an $m\times n$ matrix, and the **SVD** equation decomposes the matrix **A** into *r* matrices with the same shape ($m\times n$).

First, let me show why this equation is valid. If we multiply both sides of the SVD equation by x we get:

$$\mathbf{Ax} = \sigma\mathbf{u1}\mathbf{v}^T{}_1\mathbf{x} + \sigma_2\mathbf{u}_1\mathbf{v}^T{}_2\mathbf{x} + \dots + \sigma_r\mathbf{u}_r\mathbf{v}^T{}_r\mathbf{x}$$

We know that the set {**u1, u2, …,** $\mathbf{u_r}$} is an orthonormal basis for **Ax**. So the vector **Ax** can be written as a linear combination of them.

$$\mathbf{Ax} = a_1\mathbf{u}_1 + a_2\mathbf{u}_2 \dots a_r\mathbf{u}_r$$

and since **ui** vectors are orthogonal, each term *ai* is equal to the dot product of **Ax** and **ui** (scalar projection of Ax onto **ui**):

$$a_i = \mathbf{Ax}.\mathbf{u_i} = (\mathbf{Ax})^T\mathbf{u_i} = \mathbf{x}^T\mathbf{A}^T\mathbf{u_i}$$

but we also know that

$$\mathbf{u_i} = \frac{\mathbf{Av_i}}{\sigma_i}$$

So by replacing that into the previous equation, we have:

$$a_i = \mathbf{x}^T\mathbf{A}^T\frac{\mathbf{Av_i}}{\sigma_i} = \frac{1}{\sigma_i}\mathbf{x}^T\mathbf{A}^T\mathbf{Av_i}$$

We also know that **vi** is the eigenvector of **A^T A** and its corresponding eigenvalue *λi* is the square of the singular value *σi*

$$a_i = \frac{1}{\sigma_i}\mathbf{x}^T\mathbf{A}^T\mathbf{Av_i} = \frac{1}{\sigma_i}\mathbf{x}^T\sigma_i^2\mathbf{v_i} = \sigma_i\mathbf{x}^T\mathbf{v_i}$$

But dot product is commutative, so

$$a_i = \sigma_i\mathbf{x}^T\mathbf{v_i} = \sigma_i\mathbf{v_i}^T\mathbf{x}$$

Notice that **vi^Tx** gives the scalar projection of **x** onto **vi**, and the length is scaled by the singular value. Now if we replace the *ai* value into the equation for **Ax**, we get the SVD equation:

$$\mathbf{Ax} = \sigma_1\mathbf{u_1}\mathbf{v_1}^T\mathbf{x} + \sigma_2\mathbf{u_2}\mathbf{v_2}^T\mathbf{x} + \ldots + \sigma_r\mathbf{u_r}\mathbf{v_r}^T\mathbf{x}$$

So each $ai = \sigma i v^T_i$ x is the scalar projection of **Ax** onto ui, and if it is multiplied by **ui**, the result is a vector which is the orthogonal projection of **Ax** onto **ui**. The singular value *σi* scales the length of this vector along **ui**. Remember that in the eigendecomposition equation, each **ui** $\mathbf{u}^T_\mathbf{i}$ was a projection matrix that would give the orthogonal projection of x onto **ui**. Here *σi*vi ^T can be thought as a projection matrix that takes x, but projects **Ax** onto **ui**. Since it projects all the vectors on **ui**, its rank is 1. Figure 3 summarizes all the steps required for SVD. We start by picking a random 2-d vector x1 from all the vectors that have a length of 1 in x (Figure 3–1). Then we try to calculate **Ax1** using the SVD method.

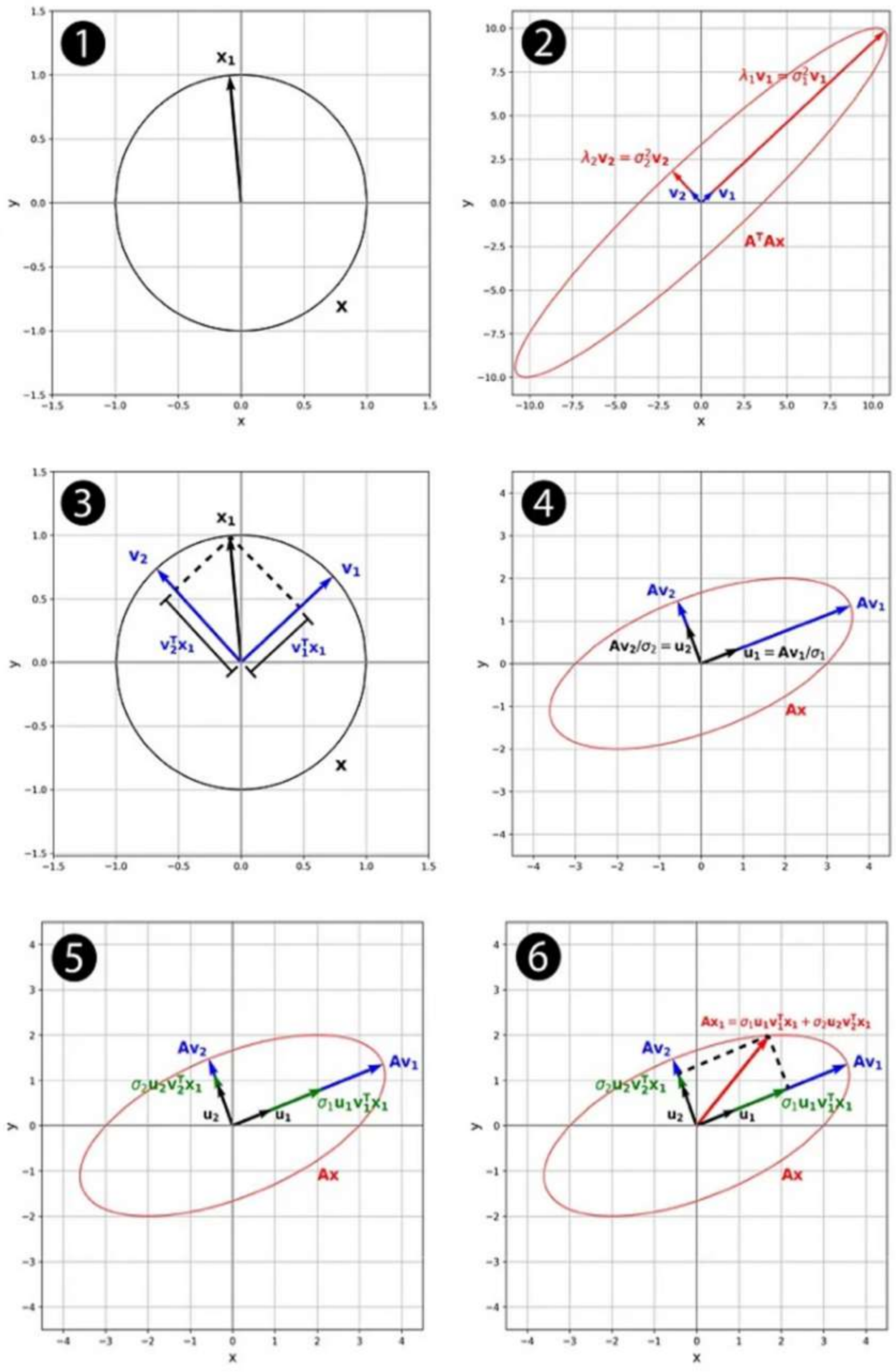
1
x_1
x
2
$\lambda_1 v_1 = \sigma_1^2 v_1$
$\lambda_2 v_2 = \sigma_2^2 v_2$
v_2
v_1
A^TAx
3
x_1
v_2
v_1
$v_2^T x_1$
$v_1^T x_1$
x
4
Av_2
Av_1
$Av_2/\sigma_2 = u_2$
$u_1 = Av_1/\sigma_1$
Ax
5
Av_2
Av_1
$\sigma_2 u_2 v_2^T x_1$
$\sigma_1 u_1 v_1^T x_1$
u_2
u_1
Ax
6
$Ax_1 = \sigma_1 u_1 v_1^T x_1 + \sigma_2 u_2 v_2^T x_1$
Av_2
Av_1
$\sigma_2 u_2 v_2^T x_1$
$\sigma_1 u_1 v_1^T x_1$
u_2
u_1
Ax

Figure 3

First, we calculate the eigenvalues (*λ1, λ2*) and eigenvectors (**v1, v2**) of **A^TA.** We know that the singular values are the square root of the eigenvalues ($\sigma i^2 = \lambda i$) as shown in (Figure 17–2). **Av1** and **Av2** show the directions of stretching of **Ax,** and **u1** and **u2** are the unit vectors of **Av1** and **Av2** (Figure 17–4). The orthogonal projection of **Ax1** onto **u1** and **u2** are

$$\sigma_1 \mathbf{u_1} \mathbf{v_1^T} \mathbf{x_1} \quad , \quad \sigma_2 \mathbf{u_2} \mathbf{v_2^T} \mathbf{x_1}$$

respectively (Figure 17–5), and by simply adding them together we get **Ax1**

$$\mathbf{A}\mathbf{x_1} = \sigma_1 \mathbf{u_1} \mathbf{v_1^T} \mathbf{x_1} + \sigma_2 \mathbf{u_2} \mathbf{v_2^T} \mathbf{x_1}$$

2.2 Computation Methods

2.2.1 Direct Methods Algorithm

The direct method for computing the SVD involves the following steps:

1. Compute the eigenvalues and eigenvectors of the matrix A^TA.
2. Sort the eigenvalues in descending order and arrange the corresponding eigenvectors accordingly.

3. Form the singular values by taking the square root of the eigenvalues (after sorting) from step 2.
4. Normalize the eigenvectors obtained in step 2 to obtain the columns of the matrix **U**.

Compute the matrix V by using the formula V = A^TUΣ^(-1).

It's important to note that the direct method for SVD can be computationally expensive for large matrices since it involves computing the eigenvalues and eigenvectors. In practice, iterative methods such as the Golub-Reinsch algorithm (also known as the bidiagonalization method) or the Lanczos algorithm are often preferred for efficient computation of the SVD.

2.2.2 Iterative Methods.

The iterative method of singular value decomposition (SVD) is an algorithmic approach to compute the SVD of a matrix by iteratively improving an initial estimate of the solution. **The iterative method for SVD is typically faster and more memory-efficient than the direct method, especially for large matrices**.

One commonly used iterative method for SVD is the Golub-Reinsch algorithm, also known as the bidiagonalization method. The basic idea of this algorithm is to transform the input matrix **A** into bidiagonal form, which is a matrix with non-zero entries only on the diagonal and one subdiagonal (or superdiagonal), using a sequence of orthogonal transformations. The bidiagonal matrix is then reduced to a diagonal matrix through a sequence of Givens rotations, which are plane rotations that eliminate off-diagonal elements. The singular values of the input matrix can be obtained from the diagonal elements of the resulting diagonal matrix.

The basic steps of the Golub-Reinsch algorithm are as follows:

1. Compute the matrix A^TA and its eigendecomposition $A^TA = V\Lambda V^T$.
2. Choose an initial orthogonal matrix Q and set $B = Q^TA$.
3. Repeat the following steps until convergence:

Other iterative methods for SVD include the Lanczos algorithm, which is a variant of the power iteration method, and the implicitly restarted Arnoldi method, which is a variant of the Arnoldi iteration. These methods are typically more efficient than the Golub-Reinsch algorithm for large sparse matrices, but may be less stable or accurate for matrices with small singular values.

2.2.3 Numerical Stability

To enhance the numerical stability of SVD, it is advisable to preprocess the input data by scaling or normalizing it appropriately. This can help mitigate ill-conditioning and minimize the impact of round-off errors during the computation of the SVD.

Overall, while iterative methods for SVD offer better numerical stability compared to the direct method, it's always a good practice to be aware of the properties of the input matrix, its condition number, and the potential effects of numerical errors to ensure accurate and reliable results.

2.3 Properties of SVD

2.3.1 Uniqueness and Existence

Uniqueness and existence of svd

The existence and uniqueness of singular value decomposition (SVD) are fundamental properties of this matrix factorization.

Existence: For any real or complex matrix A with dimensions m×n, an SVD always exists. This means that every matrix can be decomposed into the product of three matrices: U, Σ, and V^T, as follows:

A = UΣV^T

In this factorization, U is an m×m orthogonal matrix, Σ is an m×n diagonal matrix with non-negative real numbers on the diagonal (the singular values), and V^T is the transpose of an n×n orthogonal matrix V. The singular values are typically arranged in non-increasing order along the diagonal of Σ.

Uniqueness: The uniqueness of the SVD lies in the orthogonality properties of the matrices U and V. While the singular values in Σ may differ, the singular vectors in U and V are uniquely determined up to a sign. This means that although there may be multiple valid SVDs for a given matrix, the singular vectors remain the same, apart from a possible sign change.

To be precise, if (U, Σ, V^T) and (U', Σ', V'^T) are two SVDs of the same matrix A, then U and U' are both orthogonal, V and V' are both orthogonal, and Σ and Σ' have the same dimensions. Additionally, the corresponding singular values in Σ and Σ' are the same, except for a possible rearrangement in order.

It's worth noting that for matrices with repeated singular values, the singular vectors associated with those singular values may not be unique. In such cases, there may be multiple valid SVDs that differ in the choice of singular vectors corresponding to the repeated singular values.

Overall, while the singular values and their ordering may differ, the existence of SVD ensures that every matrix can be decomposed in the form A = UΣV^T, and the uniqueness property guarantees that the singular vectors, up to a sign, remain the same across different valid SVDs of the same matrix.

2.3.2 The orthogonality property

The orthogonality property of the SVD arises from the properties of the matrices U and V.

The matrix U is unitary, which means that $U^TU=I$, where I is the identity matrix. This property implies that the columns of U form an orthonormal set, meaning that they are orthogonal to each other and have unit length.

The matrix V is also unitary, $V^TV=I$. Similarly to U, the columns of V are orthogonal to each other and have unit length.

The orthogonality of the columns in U and V is crucial in the SVD because it allows for the representation of the original matrix A as a product of three matrices. These orthogonal matrices U and V enable the transformation of the original matrix A into a diagonal matrix ΣΣ. The columns of U and V define new coordinate systems in which the original matrix A can be decomposed into a scaling transformation represented by ΣΣ.

In summary, the singular value decomposition provides an orthogonal decomposition of a matrix by expressing it as a product of three matrices: a matrix of left singular vectors (U), a diagonal matrix of singular values (ΣΣ), and a matrix of right singular vectors (V). The orthogonality of U and V ensures that the SVD retains the linear independence and orthogonality properties of the original matrix A.

Chapter Three

Applications of svd

3.1 Dimensionality reduction

We can store an image in a matrix. Every image consists of a set of pixels which are the building blocks of that image. Each pixel represents the color or the intensity of light in a specific location in the image. In a grayscale image with PNG format, each pixel has a value between 0 and 1, where zero corresponds to black and 1 corresponds to white. So a grayscale image with $m \times n$ pixels can be stored in an $m \times n$ matrix

3.2 Eigenfaces

Eigenfaces is a method that is useful for face recognition and detection by determining the variance of faces in a collection of face images and use those variances to encode and decode a face in a machine learning way without the full information reducing computation and space complexity.

3.3 Reducing noise

SVD can be used to reduce the noise in the images.

References:

[1]GILBERT STRANG, (2009), INTRODUCTION TO LINEAR ALGEBRA, WELLESLEY - CAMBRIDGE PRESS

[2] Stewart, G. W. (1998). Matrix algorithms, volume II: Eigensystems. SIAM.

[3] Trefethen, L. N., & Bau III, D. (1997). Numerical linear algebra. SIAM.

[4] Van Loan, C. F. (2000). Computational frameworks for the fast Fourier transform. Society for Industrial and Applied Mathematics

[5] Golub, G. H., & Van Loan, C. F. (2012). Matrix computations. JHU Press.

[6] Brand, M. (2006). Fast low-rank modifications of the thin singular value decomposition. Linear algebra and its applications, 415(1), 20-30.

[7] Halko, N., Martinsson, P. G., & Tropp, J. A. (2011). Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. SIAM review, 53(2), 217-288