Linear Algebra

International Institute of Technology, Hyderabad

Project Final Report

Matrix decomposition

A study of various methods and algorithms

Team 33
Khooshi Asmi (2022114006)
Saravana Mitra Somayaji Tangirala (2022102016)
Vyakhya Gupta (2022101104)

Submission Date
June 21, 2023

Instructor
Chittaranjan Hens

Abstract

This project focuses on the study and analysis of matrix decomposition techniques, namely Singular Value Decomposition (SVD), Eigenvalue Decomposition, LU Decomposition, and QR Decomposition. These decomposition methods have significant applications in various fields of science and engineering, including linear algebra, signal processing, data analysis, and machine learning.

The project begins by introducing the concepts and principles behind each decomposition method, and exploring their mathematical formulations and properties. It further examines the state of the art literature related to each decomposition technique, highlighting their practical applications and advancements in computational algorithms.

The specific targets of the project include dimensionality reduction using SVD, accurate computation of eigenvalues and eigenvectors, efficiency analysis of LU decomposition, and the use of QR decomposition in solving linear systems and least square fitting problems.

Linear algebra plays a fundamental role in these decomposition methods, encompassing concepts such as orthogonal matrices, matrix factorization, eigenvalues and eigenvectors, orthogonality, and the Gram-Schmidt process. The project aims to provide a comprehensive understanding of these concepts and demonstrate their applications through practical examples and comparative analysis.

Further, this project explores the applications of matrix decomposition in recommender systems and digital signal processing. Singular Value Decomposition (SVD) is used in recommender systems to identify latent factors and generate personalized recommendations. Principal Component Analysis (PCA) is applied in digital signal processing for efficient computations and dimensionality reduction. The project demonstrates the versatility and effectiveness of matrix decomposition in these domains.

In conclusion, this project serves as a valuable resource for understanding and utilizing matrix decomposition techniques, providing insights into their theoretical foundations, computational efficiency, and practical implications in various domains.

Contents

1 Introduction

Matrix decomposition is a powerful tool in linear algebra that allows for the analysis and manipulation of matrices by breaking them down into their constituent parts. It provides valuable insights into the structure and properties of matrices, enabling efficient computation and solving of complex problems. This project focuses on four major matrix decomposition techniques: Singular Value Decomposition (SVD), Eigenvalue Decomposition, LU Decomposition, and QR Decomposition.

Singular Value Decomposition (SVD) is a widely-used matrix factorization technique that breaks down a matrix into three component matrices: a left singular matrix, a diagonal matrix of singular values, and a right singular matrix. SVD has numerous applications, including dimensionality reduction, image compression, data analysis, and recommendation systems.

Eigenvalue Decomposition decomposes a square matrix into a set of eigenvectors and eigenvalues. It is a fundamental concept in linear algebra and has important applications in solving systems of linear equations, analyzing dynamic systems, and understanding the behavior of linear transformations.

LU Decomposition factorizes a square matrix into the product of a lower triangular matrix and an upper triangular matrix. It is commonly used to solve systems of linear equations, compute matrix inverses, and perform numerical computations.

QR Decomposition decomposes a matrix into the product of an orthogonal matrix and an upper triangular matrix. It is widely used in solving least square problems, finding eigenvalues and eigenvectors, and performing numerical computations.

This project aims to provide a comprehensive understanding of these matrix decomposition techniques, exploring their mathematical foundations, practical applications, and computational efficiency. Through a review of the state of the art literature and comparative analysis, the project will showcase the strengths and limitations of each decomposition method, enabling a deeper insight into their use and potential advancements in the field of linear algebra.

2 Problem Statement

The main problem statement of this project is to study and analyze various matrix decomposition techniques, including Singular Value Decomposition (SVD), Eigenvalue Decomposition, LU Decomposition, and QR Decomposition. The project aims to understand the mathematical formulations, properties, and practical applications of these decomposition methods. The project also compares the efficiency, accuracy, and computational complexity of these techniques in different scenarios. Additionally, the project also aims to explore the applications of Matrix Decomposition in various fields.

3 Singular Value Decomposition (SVD)

3.1 Introduction to SVD and its applications

3.1.1 Introduction

Singular Value Decomposition (SVD) is a powerful matrix factorization technique in linear algebra that has found extensive applications across various fields. It provides a unique and efficient representation of matrices, allowing for dimensionality reduction, data compression, and extracting important features. SVD has become an essential tool in data analysis, machine learning, image processing, and collaborative filtering.

3.1.2 Applications of SVD

Singular Value Decomposition (SVD) has numerous applications across different fields. Here are some of the key applications of SVD:

1. Dimensionality Reduction: SVD is widely used for reducing the dimensionality of high-dimensional data. By selecting a subset of the most significant singular values and corresponding singular vectors, SVD allows for representing the data in a lower-dimensional space while preserving the important features. This is beneficial for data visualization, feature extraction, and efficient computation in machine learning and data analysis.

2. Image Compression: SVD is applied in image compression techniques such as JPEG. Images can be represented as matrices, and by applying SVD to the image matrix, one can identify the most significant singular values and vectors. By keeping only a subset of them, the image can be compressed while retaining visual quality to some extent. This is useful in reducing storage requirements and transmission bandwidth for images.

3. Principal Component Analysis (PCA): PCA is a statistical technique that uses SVD to extract the principal components of a dataset. It identifies the directions of maximum variance in the data and projects it onto a lower-dimensional space. PCA is used for feature extraction, data visualization, and noise reduction in various fields such as image processing, genetics, finance, and signal processing.

3.2 Theory and Mathematical Formulation of SVD

Singular value decomposition takes a rectangular matrix of gene expression data (defined as A, where A is an $n \times p$ matrix) in which the n rows represent the genes, and the p columns represent the experimental conditions. The SVD theorem states:

$$A_{n\times p} = U_{n\times n} \times \Sigma_{n\times p} \times V^T_{p\times p}$$

where $U^T U = I_{n\times n}$ and $V^T V = I_{p\times p}$ (i.e. U and V are orthogonal)

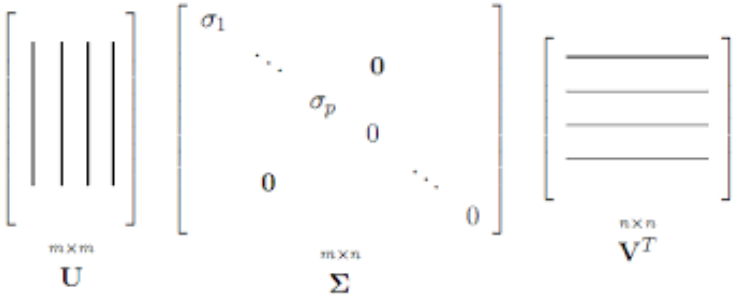
Where the columns of U are the left singular vectors (gene coefficient vectors); Σ (the same dimensions as A) has singular values and is diagonal (mode amplitudes); and V^T has rows that are the right singular vectors (expression level vectors). The SVD represents an expansion of the original data in a coordinate system where the covariance matrix is diagonal.

Calculating the SVD consists of finding the eigenvalues and eigenvectors of $A^T A$ and AA^T. The eigenvectors of $A^T A$make up the columns of V, the eigenvectors of AA^T make up the columns of U. Also, the singular values in Σ are square roots of eigenvalues from $A^T A$ or AA^T. The singular values are the diagonal entries of the Σ matrix ($\sigma_1, \sigma_2, \sigma_3, \ldots, \sigma_n$) and are arranged in descending order. The singular values are always real numbers. If the matrix A is a real matrix, then U and V are also real.

NOTE: For a given matrix A, the singular values are the square roots of the eigenvalues of the matrix AA^T (the non-zero eigenvalues are the singular values). They are denoted as $\sigma_1, \sigma_2, \ldots, \sigma_k, \ldots, \sigma_n$, where n is the dimension of the matrix.

The singular values are arranged in descending order, meaning that σ_1 is the largest singular value, σ_2 is the second largest, and so on. The number of non-zero singular values, known as the rank of the matrix, determines the dimensionality of the matrix's column and row spaces.

The goal of SVD is to factorize A into three separate matrices: U, Σ, and V^T.



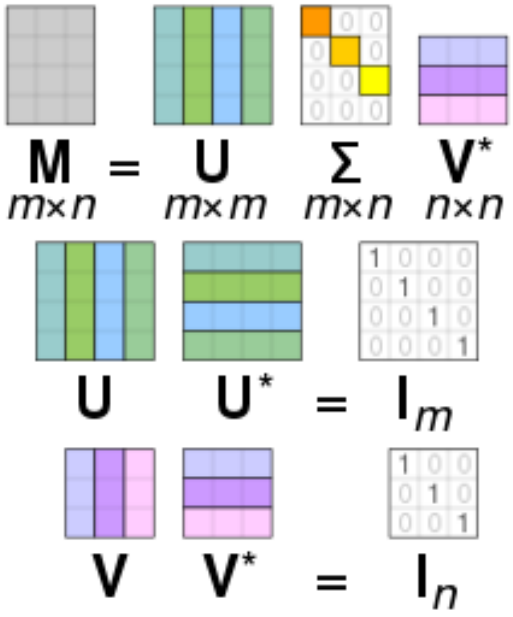


Figure 1: Visualization of the matrix multiplications in singular value decomposition

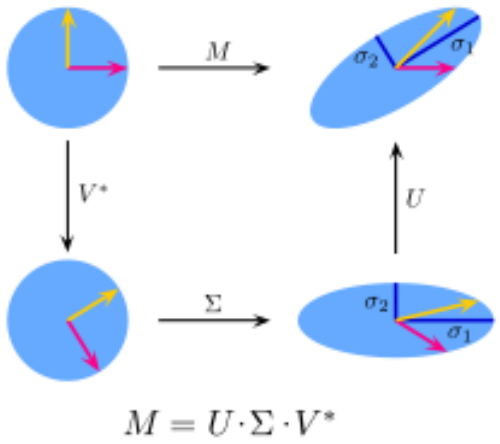


Figure 2: Illustration of the singular value decomposition $U\Sigma V^*$ of a real 2×2 matrix M

.

- **Top:** The action of M, indicated by its effect on the unit disc D and the two canonical unit vectors e_1 and e_2.
- **Left:** The action of V^*, a rotation, on D, e_1, and e_2.
- **Bottom:** The action of Σ, scaling by the singular values σ_1 horizontally and σ_2 vertically.
- **Right:** The action of U, another rotation.

3.2.1 Proof:

Let M be an $m \times n$ complex matrix. Since $M \times M$ is positive semi-definite and Hermitian, by the spectral theorem, there exists an $n \times n$ unitary matrix V such that $V \times M \times M \times V = D = \begin{bmatrix} D & 0 \\ D & 0 \end{bmatrix}$

Where D is diagonal and positive definite, of dimension $l \times l$, with l being the number of non-zero eigenvalues of $M \times M$ which can be shown to verify $l \leq \min(n, m)$. Note that V is here, by definition, a matrix whose i-th column is the i-th eigenvector of $M \times M$, corresponding to the eigenvalue $\overline{D_{ii}}$. Moreover, the j-th column of V, for $j > l$, is an eigenvector of M^*M with eigenvalue $\overline{D_{jj}} = 0$. This can be expressed by writing V as $V = [V_1 \quad V_2]$, where the columns of V_1 and V_2, therefore, contain the eigenvectors of M^*M corresponding to non-zero and zero eigenvalues, respectively. Using this rewriting of V, the equation becomes:

$$\begin{bmatrix} V_1^* \\ V_2^* \end{bmatrix} M * M \begin{bmatrix} V_1 & V_2 \end{bmatrix} = \begin{bmatrix} V_1^* M^* M V_1 & V_1^* M^* M V_2 \\ V_2^* M^* M V_1 & V_2^* M^* M V_2 \end{bmatrix} = \begin{bmatrix} D & 0 \\ D & 0 \end{bmatrix}$$

This implies that:

$$V_1 M^* M V_1 = D, \quad V_2 M^* M V_2 = 0.$$

Moreover, the second equation implies $MV_2 = 0$. Finally, the unitary-ness of V translates, in terms of V_1 and V_2, into the following conditions:

$$V_1^* V_1 = I_1, \quad V_2^* V_2 = I_2, \quad V_1^* V_1 + V_2^* V_2 = I_{12},$$

where the subscripts on the identity matrices are used to remark that they are of different dimensions.

Proof of existence based on the **spectral theorem.** Let us now define $U_1 = MV_1 D^{-1/2}$. Then,

$$U_1 D^{1/2} V_1^* = MV_1 D^{-1/2} D^{1/2} V_1^* = M(I - V_2 V_2^*) = M - (MV_2)V_2^* = M,$$

since $MV_2 = 0$. This can be also seen as an immediate consequence of the fact that $MV_1 V_1^* = M$. This is equivalent to the observation that if $\{v_i\}_{i=1}^l$ is the set of eigenvectors of M^*M corresponding to non-vanishing eigenvalues$\{\lambda_i\}_{i=1}^l$, then $\{Mv_i\}_{i=1}^l$ is a set of orthogonal vectors, and $\{\lambda_i^{-1/2} Mv_i\}_{i=1}^l$is a set of orthonormal vectors. This matches with the matrix formalism used above, denoting with V_1 the matrix whose columns are $\{\lambda_i\}_{i=1}^l$, with V_2 the matrix whose columns are the eigenvectors of $M * M$ with vanishing eigenvalue, and U_1 the matrix whose columns are the vectors $\{\lambda_i^{-1/2} Mv_i\}_{i=1}^l$

We see that this is almost the desired result, except that U_1 and V_1 are in general not unitary since they might not be square. However, we do know that the number of rows of U_1 is no smaller than the number of columns, since the dimensions of D is no greater tham m and n.
Also, since $U_1 * U_1 = D^{-1/2} V_1 * M * M * V_1 D^{-1/2} = D^{-1/2} D D^{-1/2} = I_1$, the columns in U_1 are orthonormal and can be extended to an orthonormal basis. This means that we can choose U_2 such that $U = [U_1\, U_2]$ is unitary. For V_1, we already have V_2 to make it unitary. Now, define

$$\Sigma = \begin{bmatrix} \begin{bmatrix} D^1/2 & 0 \\ 0 & 0 \end{bmatrix} \\ 0 \end{bmatrix}$$

where extra zero rows are added or removed to make the number of zero rows equal to the number of columns of U_2, and hence the overall dimensions of Σ equal to $m \times n$. Then

$$\begin{bmatrix} U_1 & U_2 \end{bmatrix} \begin{bmatrix} \begin{bmatrix} D^1/2 & 0 \\ 0 & 0 \end{bmatrix} \\ 0 \end{bmatrix} \begin{bmatrix} V_1 & V_2 \end{bmatrix}^* = \begin{bmatrix} U_1 & U_2 \end{bmatrix} \begin{bmatrix} D^{1/2} V_1^* \\ 0 \end{bmatrix} = U_1 D^{1/2} V_1^* = M$$

which is the desired result.

$$M = U \Sigma V^*$$

3.3 Netflix and SVD

Given the very large data matrix, it was only expected that competitors attempted to do dimensionality reduction, and as it turns out, this was the basis for the winning algorithm. Dimensionality reduction can be done via matrix factorization, which has the following advantage: when explicit feedback is not

available, we can infer user preferences using implicit feedback, which indirectly reflects opinion by observing user behavior, including purchase history, browsing history, search patterns, or even mouse movements.

We can write the SVD as:

$$A = U\sqrt{\Sigma}\sqrt{\Sigma}V^{\dagger}$$

and given the span of the subspaces above, we can now intuitively think what the terms $p_u = U\sqrt{\Sigma}$ and $q_i = V\sqrt{\Sigma}\dagger$ represent.

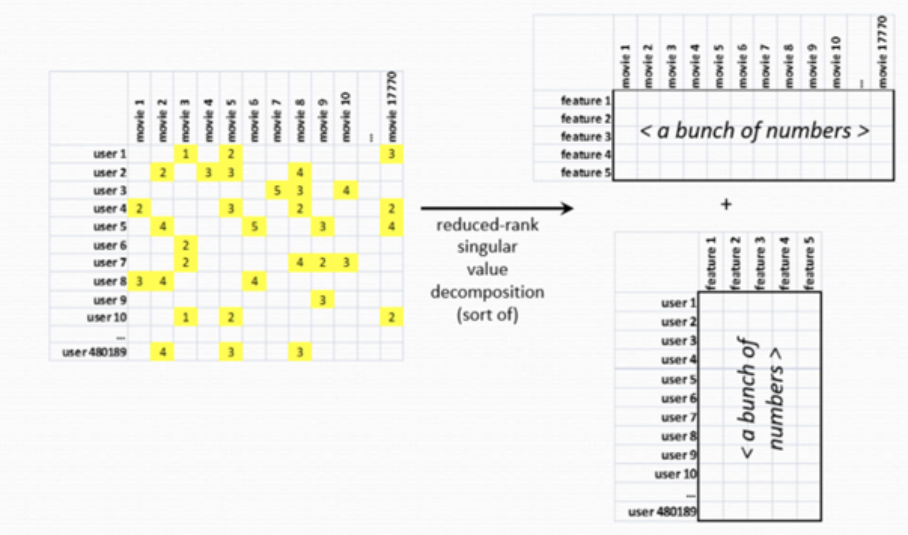


Figure 3: SVD decomposition reveals latent features weighted by the underlying singular values of the data matrix

.

The first term represents the column space and provides a representation of all $inter-user$ factors (also called latent or hidden features). The second term represents the row space and provides a representation of all inter-item factors. This brings us to the major point of what the $\sqrt{\Sigma}$ is doing to both terms. It represents the significance of those factors, and therefore we can very easily use the singular values it contains to "compress" our representation by selecting the f largest singular values and ignoring the rest.

Given these vectors of factors, we can now use them to predict the rating:

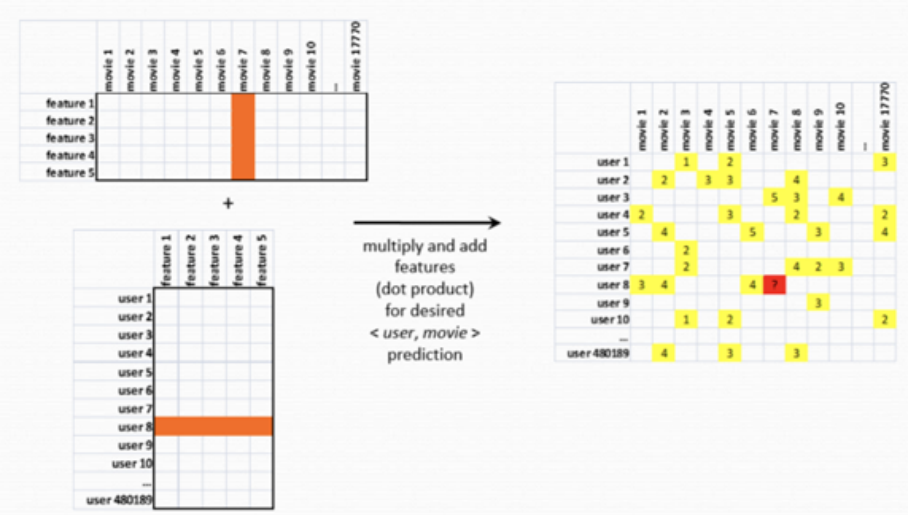


Figure 4: Prediction of a rating is the product between user latent features and movie latent features matrices

3.4 Applications

Code snippets for different operations on SVD:

3.4.1 Calculate Singular-Value Decomposition

The SVD can be calculated by calling the `svd()` function. The function takes a matrix and returns the U, Σ, and V^T elements. The Σ diagonal matrix is returned as a vector of singular values. The V matrix is returned in a transposed form, i.e., V^T. The code below defines a 3×2 matrix and calculates the Singular-value decomposition.

```
# Singular-value decomposition
from numpy import array
from scipy.linalg import svd

# define a matrix
A = array([[1, 2], [3, 4], [5, 6]])
print(A)

# SVD
U, s, VT = svd(A)
print(U)
print(s)
print(VT)
```

Output:

```
[[1 2]
 [3 4]
 [5 6]]
[[-0.2298477   0.88346102  0.40824829]
 [-0.52474482  0.24078249 -0.81649658]
 [-0.81964194 -0.40189603  0.40824829]]
[9.52551809  0.51430058]
[[-0.61962948 -0.78489445]
 [-0.78489445  0.61962948]]
```

3.4.2 SVD using NumPy and SciPy

In this code, we will try to calculate the Singular Value Decomposition using NumPy and SciPy. We will be calculating SVD and also performing the pseudo-inverse. In the end, we can apply SVD for compressing the image.

```
# import necessary libraries
from skimage.color import rgb2gray
from skimage import data
import matplotlib.pyplot as plt
import numpy as np
from scipy.linalg import svd

# define a matrix
X = np.array([[3, 3, 2], [2, 3, -2]])
print(X)

# perform SVD
U, singular, V_transpose = svd(X)

# print different components
print("U: ", U)
print("Singular array: ", singular)
print("V^T: ", V_transpose)

# inverse of singular matrix is just the reciprocal of each element
singular_inv = 1.0 / singular

# create m x n matrix of zeroes and put singular values in it
s_inv = np.zeros(X.shape)
s_inv[0][0] = singular_inv[0]
s_inv[1][1] = singular_inv[1]

# calculate pseudoinverse
M = np.dot(np.dot(V_transpose.T, s_inv.T), U.T)
print(M)

# plot the original image
cat = data.chelsea()
plt.imshow(cat)

# convert to grayscale
gray_cat = rgb2gray(cat)

# calculate the SVD and plot the image
U, S, V_T = svd(gray_cat, full_matrices=False)
S = np.diag(S)

fig, ax = plt.subplots(5, 2, figsize=(8, 20))
curr_fig = 0

for r in [5, 10, 70, 100, 200]:
    cat_approx = U[:, :r] @ S[0:r, :r] @ V_T[:r, :]
    ax[curr_fig][0].imshow(cat_approx, cmap='gray')
    ax[curr_fig][0].set_title("k = " + str(r))
    ax[curr_fig, 0].axis('off')
    ax[curr_fig][1].set_title("Original Image")
```

```
    ax[curr_fig][1].imshow(gray_cat, cmap='gray')
    ax[curr_fig, 1].axis('off')
    curr_fig += 1

plt.show()
```

Output:

```
[[ 3  3  2]
 [ 2  3 -2]]
U: [[-0.7815437 -0.6238505]
 [-0.6238505  0.7815437]]
Singular array: [5.54801894 2.86696457]
V^T: [[-0.64749817 -0.7599438  -0.05684667]
 [-0.10759258  0.16501062 -0.9804057 ]
 [-0.75443354  0.62869461  0.18860838]]

Inverse:
[0.11462451 & 0.04347826 ]
[0.07114625 & 0.13043478 ]
[0.22134387 & -0.26086957 ]
```

The changes in the image after making changes with SVD are shown below:

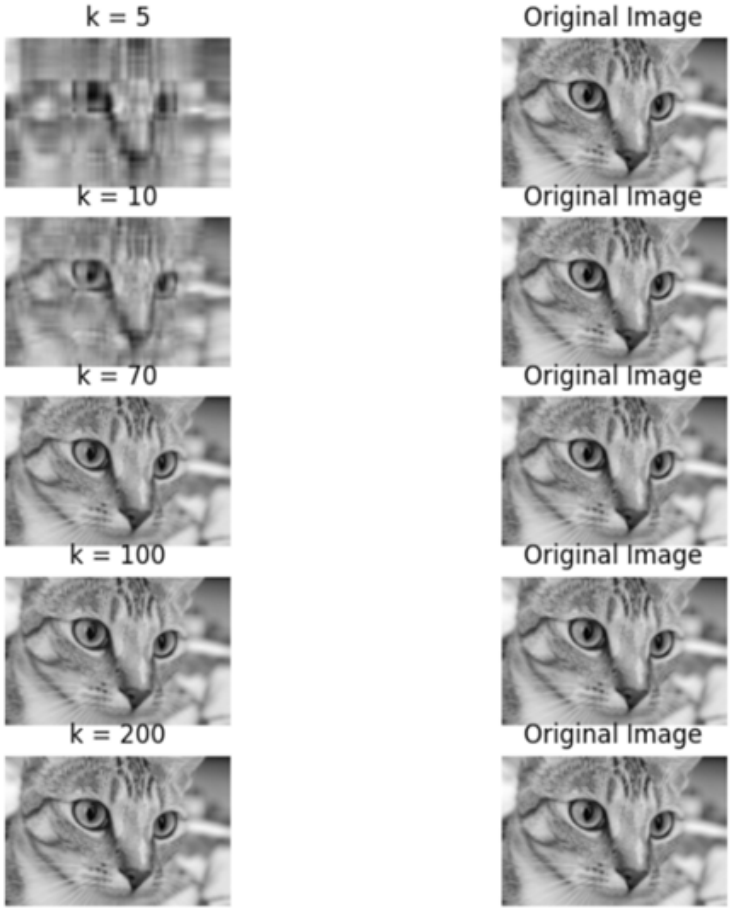


Figure 5

3.4.3 Noise filter

Singular value decomposition (SVD) is a technique commonly used in the analysis of spectroscopic data that both acts as a noise filter and reduces the dimensionality of subsequent least-squares fits. SVD allows for an unbiased differentiation between signal and noise; a small subset of singular values and vectors represents the signal well, reducing the random noise in the data. Due to this, phase information of the difference structure factors can be obtained. After identifying and fitting a kinetic mechanism, the time-independent structures of the intermediates could be recovered. This demonstrates that SVD will be a powerful tool in the analysis of experimental time-resolved crystallographic data.

4 Eigenvalue decomposition

In linear algebra, eigendecomposition is the factorization of a matrix into a canonical form, whereby the matrix is represented in terms of its eigenvalues and eigenvectors. Only diagonalizable matrices can be factorized in this way. When the matrix being factorized is a normal or real symmetric matrix, the decomposition is called "spectral decomposition," derived from the spectral theorem.

Let A be a square $n \times n$ matrix with n linearly independent eigenvectors q_i (where $i = 1, ..., n$). Then A can be factorized as

$$A = Q\Lambda Q^{-1}$$

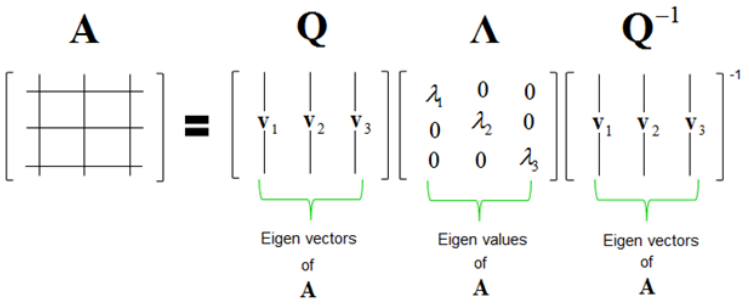
where Q is the square $n \times n$ matrix whose ith column is the eigenvector q_i of A, and Λ is the diagonal matrix whose diagonal elements are the corresponding eigenvalues, $\Lambda_{ii} = \lambda_i$. Note that only diagonalizable matrices can be factorized in this way. For example, the defective matrix (which is a shear matrix) cannot be diagonalized.

The decomposition can be derived from the fundamental property of eigenvectors:

$$Av = \lambda v$$

$$AQ = Q\Lambda$$

$$A = Q\Lambda Q^{-1}$$



Eigenvalue decomposition is applicable only to certain types of matrices, such as symmetric matrices or some classes of square matrices. Not all matrices can be decomposed using eigenvalue decomposition.

To perform eigenvalue decomposition, follow these steps: 1. Ensure that the matrix A is a square matrix.
2. Calculate the eigenvalues (λ) of the matrix A. These are the values that satisfy the equation $Av = \lambda v$, where v is the eigenvector corresponding to the eigenvalue λ.
3. Solve the equation $(A - \lambda I)v = 0$, where I is the identity matrix and v is the eigenvector. This equation helps find the eigenvectors corresponding to each eigenvalue.
4. Arrange the eigenvectors $v_1, v_2, ..., v_n$ in matrix V, where each eigenvector is a column vector.
5. Arrange the eigenvalues $\lambda_1, \lambda_2, ..., \lambda_n$ in a diagonal matrix Λ.
6. Calculate the inverse of matrix V, V^{-1}.
7. Finally, express the original matrix A as $A = V\Lambda V^{-1}$.
Eigenvalue decomposition has numerous applications, including solving systems of differential equations, diagonalizing matrices, and understanding the behavior of linear systems. It is also a crucial

step in many advanced techniques in linear algebra and data analysis, such as principal component analysis (PCA) and spectral clustering.

4.1 Classification:

There are different types of eigenvalue decompositions based on the properties of the matrix being decomposed. The three main types of eigenvalue decompositions are:

1. Symmetric Eigenvalue Decomposition: This type of decomposition is applicable to symmetric matrices. For a real symmetric matrix, the decomposition is given by $A = Q\Lambda Q^T$, where Q is an orthogonal matrix whose columns are the eigenvectors, and Λ is a diagonal matrix containing the eigenvalues.

2. Generalized Eigenvalue Decomposition: This decomposition is used when dealing with two related matrices. Given two matrices A and B, where A is not necessarily symmetric, the decomposition is represented as $Av = \lambda Bv$, where v is the eigenvector and λ is the eigenvalue. The eigenvectors and eigenvalues are then used to form the matrices V and Λ, respectively.

3. Jordan Canonical Form: This decomposition is used when dealing with matrices that are not diagonalizable, meaning they cannot be decomposed into eigenvectors and eigenvalues directly. The Jordan Canonical Form decomposes a matrix into a block diagonal matrix, where each block represents Jordan blocks associated with eigenvalues and generalized eigenvectors.

4.2 Python Codes for Eigen Decomposition

Here is an example of Python code for performing eigen decomposition using the NumPy library:

The following Python code demonstrates the eigen decomposition of a matrix:

```
import numpy as np

# Create a sample matrix
A = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

# Perform eigendecomposition
eigenvalues, eigenvectors = np.linalg.eig(A)

# Print the eigenvalues
print("Eigenvalues:")
print(eigenvalues)

# Print the eigenvectors
print("\nEigenvectors:")
print(eigenvectors)
```

The output of the code is as follows:

```
Eigenvalues:
[ 1.61168440e+01 -1.11684397e+00 -9.75918483e-16]

Eigenvectors:
[[-0.23197069 -0.78583024  0.40824829]
 [-0.52532209 -0.08675134 -0.81649658]
 [-0.81867349  0.61232756  0.40824829]]
```

4.3 Applications

Eigenvalue decomposition (eigendecomposition) has various real-time applications across different fields. Here are a few examples:

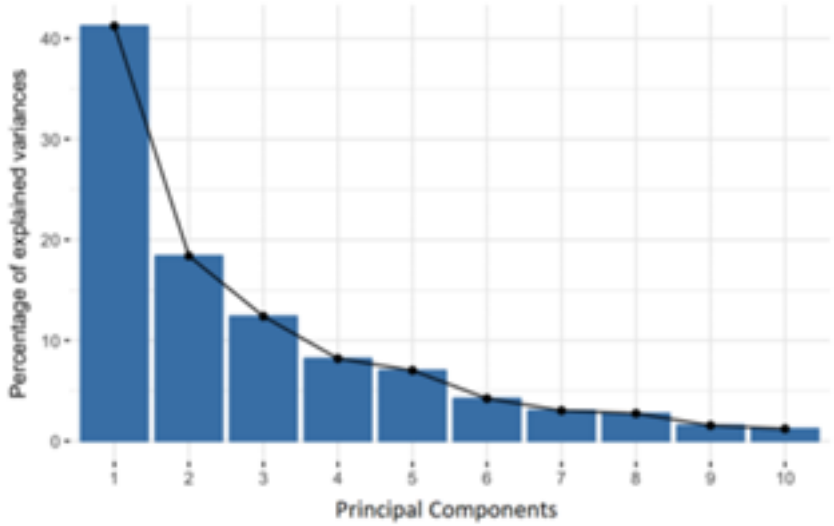


Figure 6: Percentage of variance information for each PC

- **Principal Component Analysis (PCA):** PCA is a widely used technique in data analysis and dimensionality reduction. It utilizes eigendecomposition to find the principal components of a dataset, which are linear combinations of the original variables. PCA is employed in fields such as image processing, signal processing, and machine learning.

Here is a detailed explanation of how PCA uses eigendecomposition:

1. **Data Preparation:** Assume we have a dataset consisting of n data points, where each data point has d features or variables. Before applying PCA, it is common practice to standardize the data by subtracting the mean and dividing by the standard deviation of each feature. This step ensures that the features are on similar scales, which is important for PCA to work effectively.

$$z = \frac{value - mean}{standarddeviation}$$

2. **Covariance Matrix:** The next step in PCA is to calculate the covariance matrix of the standardized data. The covariance matrix provides information about the relationships between different features in the dataset. It is a symmetric matrix with dimensions $d \times d$, where each element represents the covariance between two features.

   Covariance matrix for 3D data:

$$\begin{bmatrix} Cov(x,x) & Cov(x,y) & Cov(x,z) \\ Cov(x,x) & Cov(x,y) & Cov(x,z) \\ Cov(x,x) & Cov(x,y) & Cov(x,z) \end{bmatrix}$$

3. **Eigen Decomposition:** The eigen decomposition of the covariance matrix is performed to obtain the eigenvectors and eigenvalues. The covariance matrix can be written as $C = V\Lambda V^T$, where V is a matrix whose columns are the eigenvectors, and Λ is a diagonal matrix containing the eigenvalues. The eigenvectors represent the principal components of the dataset, and the corresponding eigenvalues represent the amount of variance explained by each principal component.

4. **Selection of Principal Components:** The eigenvectors are sorted in descending order based on their corresponding eigenvalues. This ordering ensures that the most important information is captured by the first few principal components. Typically, the principal components are selected based on a threshold, such as retaining a certain percentage of the total variance (e.g., 95%). This determines the number of principal components to keep for dimensionality reduction.

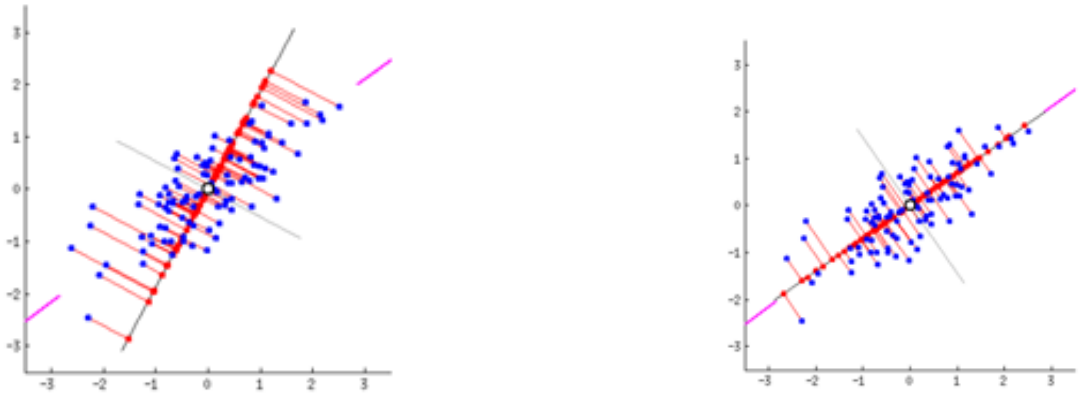


Figure 7: Selection of Principal compeonents

The line that matches the purple marks because it goes through the origin and it's the line in which the projection of the points (red dots) is the most spread out. Or mathematically speaking, it's the line that maximizes the variance (the average of the squared distances from the projected points (red dots) to the origin). The second principal component is calculated in the same way, with the condition that it is uncorrelated with (i.e., perpendicular to) the first principal component and that it accounts for the next highest variance.This continues until a total of p principal components have been calculated, equal to the original number of variables.

5. **Dimensionality Reduction:** Finally, the dataset is transformed into the lower-dimensional space defined by the selected principal components. This is achieved by multiplying the standardized data by the matrix of selected eigenvectors. The resulting transformed dataset will have reduced dimensionality, with each data point represented by the values along the principal components.

$$FinalDataSet = FeatureVector^T * StandardizedOriginalDataset^T$$

PCA is beneficial in various ways. It helps in visualizing and exploring high-dimensional data, identifying the most informative features, reducing noise, and removing redundant information. By reducing the dimensionality of the dataset, it also simplifies subsequent analysis and can improve computational efficiency.

- **Image Compression:** Eigendecomposition, specifically using methods like Singular Value Decomposition (SVD), can be applied to image compression techniques such as JPEG. By representing an image in terms of its eigenvectors and eigenvalues, it becomes possible to discard less significant components and achieve compression while maintaining important visual information.

5 LU Decomposition

The LU factorization was introduced in 1948 by the great English mathematician Alan M. Turing (1912–1954) in a paper entitled "Rounding-off Errors in Matrix Processes".

5.1 Introduction to LU decomposition and its applications

LU decomposition, also known as LU factorization, is a matrix factorization technique that decomposes a square matrix A into the product of two matrices: a lower triangular matrix L and an upper triangular matrix U. The LU decomposition can be represented as $A = LU$.

- **Solving Systems of Linear Equations** Given a system of equations, $AX = B$; A is the coefficient matrix, X the vector of unknowns, and B the constant vector, LU efficiently solves for X. By decomposing A into LU, we rewrite the system as $L(UX) = B$. We first solve $Lc = B$ for c by forward substitution, and then $Ux = c$ for x by back substitution.
- **Computing Matrix Inverses** Once a matrix A is factorized into LU form, the inverse is found by solving the equations $L(UX) = I$, we can find the inverse matrix $X = U^{-1}(L^{-1})$.
- **Sparse Matrix Computations** Modified LU decompositions, such as the SciPy library in python using SuperLU algorithm can exploit the sparsity pattern and provide storage and computational advantages in solving sparse linear systems.

5.2 Theory and mathematical formulation

The mathematical formulation of LU decomposition involves the following steps: **Step 1: Gaussian Elimination**

To obtain the LU decomposition, we start by performing Gaussian elimination with partial pivoting on the matrix A. The goal is to transform A into an upper triangular matrix U, while simultaneously applying the same row operations to another matrix L, which keeps track of the multipliers used in the elimination steps. In the operation $R_i - kR_j$, we will refer to the scalar k as the multiplier. The multipliers are precisely the entries of L that are below its diagonal.

Step 2: Lower Triangular Matrix

The lower triangular matrix L is constructed based on the multipliers used during the elimination process. It has ones on its main diagonal and the multipliers below the diagonal. The form of L can be represented as:

$$L = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ l_{21} & 1 & \cdots & 0 \\ l_{31} & l_{32} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \cdots & 1 \end{bmatrix}$$

where l_{ij} represents the multipliers used to eliminate the elements below the main diagonal.

Step 3: Upper Triangular Matrix

The resulting matrix U is the upper triangular matrix obtained after performing Gaussian elimination. It has zeros below the main diagonal, and its form can be represented as:

$$U = \begin{bmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\ 0 & u_{22} & u_{23} & \cdots & u_{2n} \\ 0 & 0 & u_{33} & \cdots & u_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & u_{nn} \end{bmatrix}$$

where u_{ij} represents the modified values of matrix A after the elimination steps.

5.3 Algorithms

5.3.1 LU Decomposition

Doolittle's method provides a way to factor A into an LU decomposition without going through the hassle of Gaussian Elimination. For a general n×n matrix A, we assume that an LU decomposition exists, and write the form of L and U explicitly. We then systematically solve for the entries in L and U from the equations that result from the multiplications necessary for A=LU.

Require: Square matrix A of size $n \times n$
Ensure: Lower triangular matrix L and upper triangular matrix U such that $A = LU$
Initialize empty lower triangular matrix L of size $n \times n$
Initialize empty upper triangular matrix U of size $n \times n$
for $i = 1$ to n **do**
    Set $L[i, i] = 1$ (diagonal elements of L are 1)
    for $j = i$ to n **do**
        Compute the dot product of $L[i, 1 : i-1]$ and $U[1 : i-1, j]$:
            sum $= \sum_{k=1}^{i-1} L[i, k] \cdot U[k, j]$
        Compute the value of $U[i, j]$: $U[i, j] = A[i, j] - \text{sum}$
    end for
    for $k = i + 1$ to n **do**
        Compute the dot product of $L[k, 1 : i-1]$ and $U[1 : i-1, i]$:
            sum $= \sum_{j=1}^{i-1} L[k, j] \cdot U[j, i]$
        Compute the value of $L[k, i]$: $L[k, i] = \frac{A[k,i] - \text{sum}}{U[i,i]}$
    end for
end for

5.3.2 $P^T LU$ decomposition

In general, P will be the product P = P_k ... $P_2 P_1$ of all the row interchange matrices $P_1, P_2, ..., P_k$ (where P_1 is performed first, and so on.) Such a matrix P is called a permutation matrix.

Let A be a square matrix. A factorization of A as $A = P^T LU$, where P is a permutation matrix, L is unit lower triangular, and U is upper triangular, is called a $P^T LU$ factorization of A.
Example: To find $P^T LU$ factorization of

$$A = \begin{bmatrix} 0 & 0 & 6 \\ 1 & 2 & 3 \\ 2 & 1 & 4 \end{bmatrix}$$

First we reduce A to row echelon form. Clearly, we need at least one row interchange.

$$A = \begin{bmatrix} 0 & 0 & 6 \\ 1 & 2 & 3 \\ 2 & 1 & 4 \end{bmatrix} -> \begin{bmatrix} 1 & 2 & 3 \\ 0 & 0 & 6 \\ 2 & 1 & 4 \end{bmatrix} -> \begin{bmatrix} 1 & 2 & 3 \\ 0 & 0 & 6 \\ 0 & -3 & -2 \end{bmatrix}$$
$$= \begin{bmatrix} 1 & 2 & 3 \\ 0 & -3 & -2 \\ 0 & 0 & 6 \end{bmatrix}$$

We have used two row interchanges (R1 4 R2 and then R2 4 R3), so the required permutation matrix is

$$P = P_1 P_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

So the LU factorisation of PA is

$$PA = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 1 & 4 \\ 0 & 0 & 6 \end{bmatrix} -> \begin{bmatrix} 1 & 2 & 3 \\ 0 & 3 & -2 \\ 0 & 0 & 6 \end{bmatrix} = U$$

So,

$$A = P^T LU = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 0 & 3 & -2 \\ 0 & 0 & 6 \end{bmatrix}$$

5.3.3 Algorithm to solve Ax=B

This involves two steps:

The original equation is to solve $Ax - b = 0$.
At the end of the Gaussian elimination, the resulting equations were:
$a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1$ $a'_{22}x_2 + a'_{23}x_3 + \ldots + a'_{2n}x_n = b'_2$ $a''_{33}x_3 + a''_{34}x_4 + \ldots + a''_{3n}x_n = b''_3$
$\ldots$ $a^{(n-1)}_{nn}x_n = b^{(n-1)}_n$

which can be written as: $Ux - d = 0$ (1)

Premultiplying (1) by another matrix L, which results in: $L(Ux - d) = 0$
That is: $LUx - Ld = 0$ (2)

Comparing (1) and (2), it is clear that:
$LU = A$ $Ld = b$ (3)

To reduce computational load, L is taken as a lower triangular matrix with 1's along the diagonal.

1. Forward substitution: Solve $Ld = b$ to find d. The values of d_i are given by $d_1 = b_1$ $d_i = b_i - \sum_{j=1}^{i-1} l_{ij}d_j \quad i = 2, 3, ..., n$

2. Back substitution: Solve $Ux = d$ to find x. The values of x_i are given by $x_n = \frac{d_n}{u_{nn}}$ $x_i = \frac{d_i - \sum_{j=i+1}^{n} u_{ij}x_j}{u_{ii}} \quad i = n-1, n-2, ..., 1$

5.3.4 Algorithm to compute matrix inverse

Here's the two-step procedure to find the inverse of a matrix A using LU decomposition

- Find the LU decomposition $A = LU$
- Forward Substitution: Solve the equation $Lc = b$, where c is a column vector and b is a column vector consisting of the columns of the identity matrix. This step involves forward substitution to obtain c.
- Back Substitution: Solve the equation $Ux = c$, where x is a column vector. This step involves back substitution to obtain x.
- The vector x obtained in the previous step is the column vector of the inverse matrix A^{-1}. Repeat steps 2 and 3 for each column of the identity matrix to obtain the complete inverse matrix.

5.3.5 Cholesky Decomposition

The Cholesky factorization, also known as Cholesky decomposition, is a process of breaking down of a Hermitian, positive-definite matrix into the product of a lower triangular matrix and its conjugate transpose.

If A is a real matrix (and thus symmetric positive-definite), the factorization can be stated as follows:

$$A = LL^T$$

Given: Positive definite Hermitian matrix A of size $n \times n$

Initialize an $n \times n$ lower triangular matrix L with zeros
for $i = 1$ to n **do**
    for $j = 1$ to $i - 1$ **do**
        Compute the sum term: sum $= \sum_{k=1}^{j-1} L[i,k] \cdot L[j,k]$
    end for
    Compute the diagonal term: $L[i,i] = \sqrt{A[i,i] - \text{sum}}$
    for $j = i + 1$ to n **do**
        Compute the off-diagonal terms: $L[j,i] = \frac{A[j,i] - \sum_{k=1}^{j-1} L[j,k] \cdot L[i,k]}{L[i,i]}$
    end for
end for

The resulting matrix L is the lower triangular matrix in the Cholesky decomposition $A = LL^*$.

5.4 Examples and Numerical Experiments

5.4.1 Computation Time

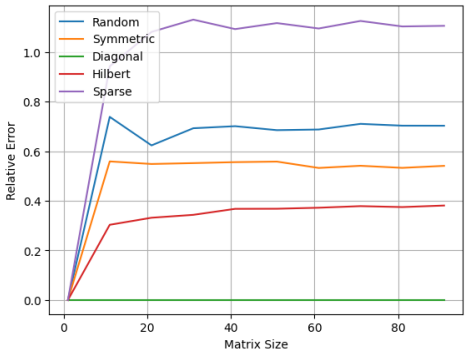
Figure 8: Error with increase in size

LU decomposition is less expensive time-wise compared to decompositions such as QR and SVD which involve sophisticated calculations and orthogonalisations. The increase in size of matrix leads to a quick increase in time required.

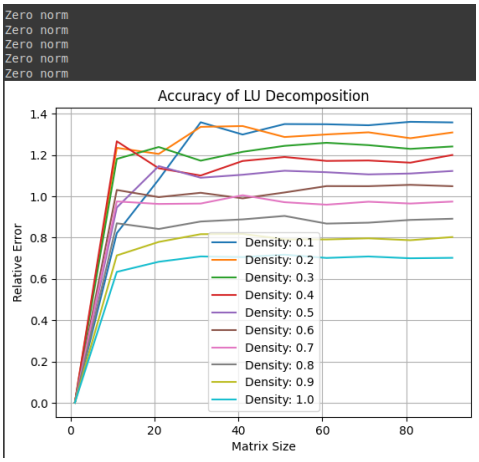
5.4.2 Accuracy and Scalability

Here we have calculated relative errors in LU decomposition for different kinds of matrices:

- Random : Increases linearly till matrix size=10, then stagnates at 0.7.
- Diagonal : Least error. Stays close to 0.
- Symmetric :linearly till matrix size=10, then stagnates at 0.55.
- Sparse : Most error comparatively, linearly till matrix size=10, then slope reduces till matrix size=30 and further stays near 1.5.



(a) Errors according to types of matrices



(b) Errors according to density of sparse matrices

Figure 9: Accuracy for LU

Sparse matrices: Relative error increases with decrease in density, i.e. increase in number of zeroes. It also gives error when the norm is zero.

5.5 Computational aspects and efficiency analysis

If A is $n \times n$, then the total number of operations (multiplications and divisions) required to solve a linear system $Ax = b$ using an LU factorization of A is $T = \frac{1}{3}n^3$, the same as is required for Gaussian elimination. This is since the forward elimination phase produces the LU factorization in $\frac{1}{3}n^3$ steps, whereas both forward and backward substitution require $\frac{1}{2}n^2$ steps. Therefore, for large values of n, the n^3 term is dominant. From this point of view, then, Gaussian elimination and the LU factorization are equivalent.

However, the LU factorization has other advantages:

- For an invertible matrix A, an LU factorization of A can be used to find A^{-1}, if necessary. Moreover, this can be done in such a way that it simultaneously yields a factorization of A^{-1}.
- Numerical stability: LU decomposition is numerically stable for well-conditioned matrices, meaning it preserves the accuracy of the original matrix under small perturbations.
- Reusability: Once the LU decomposition is computed, it can be reused for solving multiple linear systems with the same coefficient matrix, leading to computational savings.

Limitations:

- Singularity: LU decomposition cannot be directly applied to singular matrices or matrices that become singular during the decomposition process. It requires additional techniques, such as pivoting, to handle these cases.
- It requires (like most) pivoting to ensure numerical stability.

5.6 Parallel LU Decomposition Method and It's Application in Circle Transportation

5.6.1 Introduction

This paper presents a method that utilizes the Parallel LU Decomposition Algorithm for solving large-scale dense linear equations. The approach is based on the divide and conquer strategy, and it focuses on analyzing the speedup and efficiency of the algorithm. Furthermore, the parallel LU decomposition algorithm is applied to solve a circling transportation problem, showcasing its practical application.

5.6.2 Parallelism

Parallelizing the LU decomposition process allows for the distribution of computational tasks across multiple processors or computing units, enabling faster execution and improved scalability. The parallelization is typically achieved by partitioning the matrix and assigning subtasks to different processing elements.

If there are P processors, matrix L were divided into P processors in row, then each processor have n/p row, matrix U were divided into P processors in column.

For a large dense linear equations, the use of parallel computing time of algorithm significantly less than the serial algorithm is used for time.Pairs of large linear equations, application of parallel thinking, through the divide and conquer strategy designed to parallel LU decomposition algorithm reduces the size of the problem-solving and shorten the calculation time and improve the computational efficiency.

5.6.3 Transportation problem

How to reasonably dispatch vehicles, minimize the transport process,to improve the run of empty transportation departments of economic significance. Under the background of the transportation problem with practical application,use of linear programming theory, developed a cycle of transport routes, combined cycle parallel LU decomposition algorithm for solving linear equations, the general theory of the transport plan.

A transport department,accept a manufacturer of missions. According to the contract,shall be four different goods $H1, H2, H3, H4$ respectively from different storage warehouse to designated location, a specific transport task.

Table 1 goods transportation task

Name of goods	Delivery Point	Receiving Point	The quantity of goods (T)
H_1	A_1	B_1	55
H_2	A_2	B_2	30
H_3	A_3	B_3	80
H_4	A_4	B_4	35

In order to efficiently complete the transportation task, the vehicle dispatching department has chosen to use single measures for 5-ton load trucks. With a total requirement of 40 trucks, the available eight trucks must each make five round-trips. By leveraging local stability distribution and ensuring that trucks return empty after unloading, the dispatchers can optimize the transportation planning process. It is important to consider the specific transport routes to maximize efficiency and overall operational benefit.

Dispatching officers need to consider the location where each truck should return for reloading after unloading goods, in order to maximize the overall transportation benefit obtained.

Delivery /Receiving Point	A_1	A_2	A_3	A_4	The number of start
B_1	3	8	9	6	11
B_2	10	7	12	4	6
B_3	5	3	6	8	16
B_4	7	9	4	1	7
The number of receive	11	6	16	7	

Table 3 Backhaul traffic flow program

	A_1	A_2	A_3	A_4	
B_1			11		11
B_2	1		5		6
B_3	9			7	16
B_4	1	6			7
	11	6	16	7	

According to the Freight traffic flow plan and return programs, organizations loop transport, can get four cycles cyclic route, namely:

Suppose four lines' cycles were x_1, x_2, x_3, x_4, is to be determined according to cycle transport program Transportation scheme, it has the linear equations :

1. $A_1 \Rightarrow B_1 \rightarrow A_3 \Rightarrow B_3 \rightarrow A_1$ (A_1 front line has already appeared,this circuit has been closed, Under similar)
2. $A_1 \Rightarrow B_1 \rightarrow A_3 \Rightarrow B_3 \rightarrow A_4 \Rightarrow B_4 \rightarrow A_1$
3. $A_1 \Rightarrow B_1 \rightarrow A_3 \Rightarrow B_3 \rightarrow A_4 \Rightarrow B_4 \rightarrow A_2 \Rightarrow B_2 \rightarrow A_1$
4. $A_2 \Rightarrow B_2 \rightarrow A_3 \Rightarrow B_3 \rightarrow A_4 \Rightarrow B_4 \rightarrow A_1$

$$\begin{cases} x_1 + x_2 + x_3 = 11 \\ x_3 + x_4 = 6 \\ x_1 + x_2 + x_3 + x_4 = 16 \\ x_2 + x_3 + x_4 = 7 \end{cases}$$

By the parallel LU decomposition algorithm can get

$$(x_1 \quad x_2 \quad x_3 \quad x_4)^T = (9 \quad 1 \quad 1 \quad 5)$$

In order to fulfill the transportation requirements (11, 6, 16, 7), a specific plan can be implemented. This plan involves four line cycles, with the following frequencies: nine times, one time, one time, and five times respectively. The concrete implementation of this plan takes into consideration the circular nature of the routes, including the starting and ending points of each cycle and the interfaces between the different circular routes. The distances between the garage and the transportation locations must be determined in order to execute the plan effectively.

This study focuses on addressing the circular transportation problem by utilizing the parallel LU decomposition algorithm to solve linear equations. The main objective is to improve the efficiency of solving linear equations by enhancing the speed of computation. The aim is to expedite the resolution of circular transportation problems and enhance overall work efficiency.

6 QR Decomposition

6.1 Introduction to QR decomposition and its applications

Let A be an $m \times n$ matrix with linearly independent columns. Then A can be factored as $A = QR$, where Q is an $m \times n$ matrix with orthonormal columns and R is an invertible upper triangular matrix.

- Solving Linear Systems: By decomposing the coefficient matrix A into QR form, solving the system Ax = b reduces to solving the triangular system Rx = $Q^T b$, which is computationally more efficient.
- Least Squares Fitting: QR decomposition is used in least squares regression. Given a set of data points, (x, y), QR decomposition allows for the efficient computation of the best-fit line or curve by minimizing the sum of squared residuals.
- Eigenvalue Computation: QR iteration is an iterative method for computing eigenvalues and eigenvectors of a matrix. It involves iteratively applying QR decomposition to a matrix to obtain its eigenvalues and eigenvectors. This method is particularly useful for computing eigenvalues of large matrices.

6.2 Theory and mathematical formulation

Consider the *Gram-Schmidt Process*, with the vectors to be considered in the process as columns of the matrix A. That is,

$$A = \begin{bmatrix} a_1 & | & a_2 & | & \cdots & | & a_n \end{bmatrix}$$

$$u_1 = a_1, \quad e_1 = \frac{u_1}{\|u_1\|}$$

$$u_2 = a_2 - (a_2 \cdot e_1)e_1, \quad e_2 = \frac{u_2}{\|u_2\|}$$

$$u_{k+1} = a_{k+1} - (a_{k+1} \cdot e_1)e_1 - \cdots - (a_{k+1} \cdot e_k)e_k, \quad e_{k+1} = \frac{u_{k+1}}{\|u_{k+1}\|}$$

where $\|v\|$ is the norm.

The resulting QR factorization is

$$A = \begin{bmatrix} a_1 & | & a_2 & | & \cdots & | & a_n \end{bmatrix} = \begin{bmatrix} e_1 & | & e_2 & | & \cdots & | & e_n \end{bmatrix} \begin{bmatrix} a_1 \cdot e_1 & a_2 \cdot e_1 & \cdots & a_n \cdot e_1 \\ 0 & a_2 \cdot e_2 & \cdots & a_n \cdot e_2 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_n \cdot e_n \end{bmatrix} = QR$$

where each column $\mathbf{q}_i$ $(1 \leq i \leq n)$ is a unit vector and $\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_n$ are orthogonal to each other. In other words, $Q^T Q = I$, where Q^T represents the transpose of Q, and I is the identity matrix.

6.3 Algorithms

6.3.1 Gram Schmidt Process

```python
import numpy as np
def diag_sign(A):
    "Compute the signs of the diagonal of matrix A"
    D = np.diag(np.sign(np.diag(A)))
    return D

def adjust_sign(Q, R):
    """ Adjust the signs of the columns in Q and rows in R to
    impose positive diagonal of Q """
    D = diag_sign(Q)
    Q[:, :] = Q @ D
    R[:, :] = D @ R
    return Q, R

def QR_Decomposition(A):
    n, m = A.shape # get the shape of A
    Q = np.empty((n, n)) # initialize matrix Q
    u = np.empty((n, n)) # initialize matrix u
    u[:, 0] = A[:, 0]
    Q[:, 0] = u[:, 0] / np.linalg.norm(u[:, 0])

    for i in range(1, n):
        u[:, i] = A[:, i]
        for j in range(i):
            u[:, i] -= (A[:, i] @ Q[:, j]) * Q[:, j] # get each u vector
        Q[:, i] = u[:, i] / np.linalg.norm(u[:, i]) # compute each e vetor
    R = np.zeros((n, m))
    for i in range(n):
        for j in range(i, m):
            R[i, j] = A[:, j] @ Q[:, i]
    return Q, R
```

Figure 10: QR Decomposition

The given code performs QR decomposition of a matrix using the Gram-Schmidt process with column pivoting. QR decomposition decomposes a matrix A into an orthogonal matrix Q and an upper triangular matrix RR such that A=QR.

Here's a step-by-step explanation of the code:

1. The diagsign function computes the signs of the diagonal elements of a matrix AA and returns a diagonal matrix DD with these signs.

2. The adjustsign function adjusts the signs of the columns in matrix Q and rows in matrix RR to ensure a positive diagonal of Q. It uses the diagsign function to obtain the diagonal matrix DD and then performs matrix multiplication to adjust the signs of Q and R accordingly.

3. The QRDecomposition function performs the QR decomposition of the input matrix A. It initializes matrices Q and u with appropriate dimensions. The first column of Q and u are assigned values based on the first column of A. Then, for each subsequent column of AA, the function iteratively computes the corresponding u vector by subtracting the projections of AA onto the

previously computed Q vectors. The u vector is then normalized to obtain the corresponding column of Q.

4. The matrix R is initialized with zeros and is filled by computing the inner products between the columns of A and the columns of Q.

5. Finally, the adjustsign function is called to adjust the signs of Q and R based on the diagonal matrix D.

6.3.2 Solving Ax=B

$$A = QR, Q^T = Q^-1, Ax = B$$
$$QRx = B$$
$$x = Q^T B$$
$$x = R^-1 Q^T B$$

6.3.3 Least Square Fitting

$$Y = X\beta$$

The solutions to the previous least squares problem are given by the nxn matrix equation, also known as the normal equation:

$$(X^T X)\beta = X^T Y$$

where X^T is the transpose of X.
($(X^T X)^{-1} X^T$ is called the Moore-Penrose pseudo-inverse of X)

Python Code:

```
Q, R = np.linalg.qr(X_train)
Rinv_Qt = np.dot(np.linalg.inv(R),Q.T)
beta = np.dot(Rinv_Qt,y_train)

y_pred = np.dot(X_test, beta)             # predicted y=XB

mean_sq_error = mean_squared_error(y_pred,y_test)

print("Mean␣sq.␣error␣is",mean_sq_error)
```

6.3.4 Eigenvalues

Here's the step-by-step procedure to compute eigenvalues using QR decomposition:
Input: Square matrix A of size $n \times n$

Step 1: Perform QR decomposition on matrix A to obtain matrices Q and R.
Initialize $Q = A$ and $R =$ None
for $i = 1$ to n **do**
    Set $v = Q[:, i]$
    for $j = 1$ to $i - 1$ **do**
        Compute $R[j, i] = Q[:, j]^T \cdot Q[:, i]$
        Update $v = v - R[j, i] \cdot Q[:, j]$
    end for
    Compute $R[i, i] = \|v\|$, the norm of v
    Set $Q[:, i] = v/R[i, i]$
end for
Step 2: Compute the product $B = R \cdot Q$
Step 3: Repeat until convergence or a specified number of iterations:
for each iteration **do**
    Perform QR decomposition on matrix B to obtain matrices Q and R
    Update $B = R \cdot Q$
end for
Step 4: The eigenvalues of the original matrix A are given by the diagonal elements of the converged upper triangular matrix B

6.4 Examples and Numerical Experiments

6.4.1 Find QR decomposition using Gram Schmidt Process

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

The orthonormal basis for col(A) produced by the Gram-Schmidt Process is

$$q_! = \begin{bmatrix} 0.5 \\ -0.5 \\ -0.5 \\ 0.5 \end{bmatrix}, q_2 = \begin{bmatrix} 0.67 \\ 0.67 \\ 0.22 \\ 0.22 \end{bmatrix}, q_3 = \begin{bmatrix} -0.41 \\ 0 \\ 0.41 \\ 0.82 \end{bmatrix}$$

$$Q = \begin{bmatrix} 0.5 & 0.67 & -0.41 \\ -0.5 & 0.67 & 0 \\ -0.5 & 0.22 & 0.41 \\ 0.5 & 0.22 & 0.82 \end{bmatrix}$$

To find R in the QR decomposition, we can use the fact that Q has orthonormal columns, which implies that $Q^T Q = I$. Therefore, we can write:

$$Q^T A = Q^T Q R = IR = R$$

Hence, R $= \begin{bmatrix} 2 & 1 & 0.5 \\ 0 & 2.23 & 3.35 \\ 0 & 0 & 1.22 \end{bmatrix}$

6.4.2 Computation Time

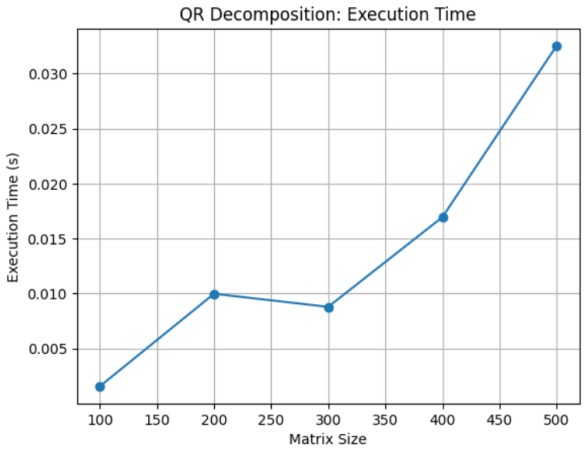


Figure 11: Error with increase in size

QR decomposition tends to be computationally more expensive than LU decomposition, especially for large matrices. QR decomposition involves orthogonalization and triangularization of the matrix, which can be computationally intensive operations. On the other hand, LU decomposition involves the factorization of the matrix into lower and upper triangular matrices.

6.4.3 Eigenvalues

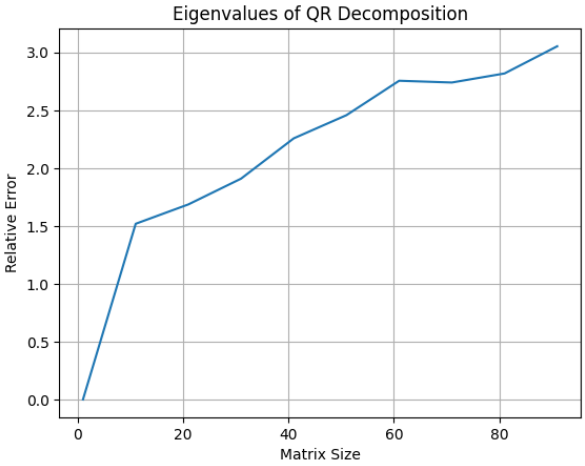


Figure 12: Error with increase in size

It is visible that the absolute error in eigenvalues computation using QR decomposition is close to 3 even for very large matrices of the order 100. Hence this method is accurate.

6.5 Computational aspects and efficiency analysis

The choice of the QR decomposition algorithm depends on the specific requirements of the problem and the properties of the matrix.

- Computational Complexity: The computational complexity of QR decomposition depends on the size of the matrix being decomposed. For an n x n matrix, the complexity is typically $O(n^3)$. This is because the algorithm involves performing several matrix multiplications and matrix factorizations.

- QR decomposition may require additional memory to store intermediate matrices, such as the Q and R matrices.
- QR decomposition can be sensitive to numerical stability issues, especially when dealing with ill-conditioned or nearly singular matrices. Care should be taken to use appropriate techniques, such as pivoting, to enhance stability and accuracy.

6.6 Principal component analysis using QR decomposition

QR-based PCA refers to Principal Component Analysis (PCA) algorithms that utilize the QR decomposition (also known as QR factorization) as part of the computation. PCA is a widely used technique in statistics and machine learning for dimensionality reduction and data analysis. Similar to the singular value decomposition (SVD) based PCA method this method is numerically stable.

6.6.1 Process

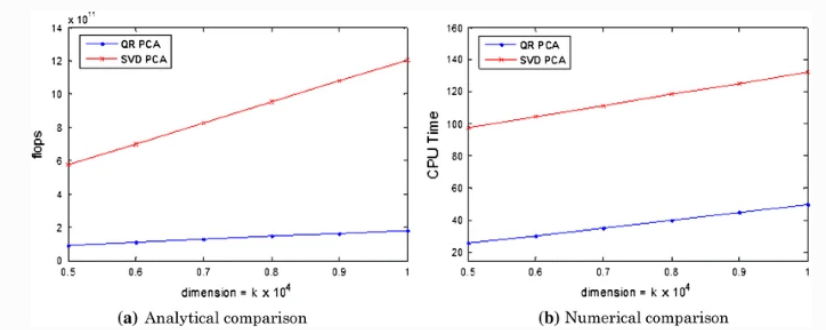
1. Start with a rectangular matrix $HR^{d\times n}$, where d is much larger than n.
2. Use the economic QR decomposition to factorize H into two matrices: $Q_1R^{d\times t}$ and $R_1R^{t\times n}$, where t is the rank of Σ_x.
3. Substitute the QR decomposition into the expression $\Sigma_x = HH^T$, resulting in $\Sigma_x = Q_1R_1R_1^TQ_1^T$.
4. Perform singular value decomposition (SVD) on the matrix R_1 to obtain $R_1 = U_1D_1V^T$, where $U_1R^{n\times t}$, $VR^{t\times t}$, and $D_1R^{t\times t}$.
5. Substitute the SVD result into the expression $\Sigma_x = Q_1VD_1U_1^TU_1D_1V^TQ_1^T$, simplifying it to $\Sigma_x = Q_1V\Lambda V^TQ_1^T$, where $\Lambda = D_1^2$.
6. The matrix Q_1V is an orthogonal matrix that diagonalizes Σ_x, with Λ being the eigenvalue matrix.
7. Select the h largest diagonal entries of D_1 and extract the corresponding column vectors from V to form the matrix $V_hR^{t\times h}$.
8. The PCA transform is given by $\Phi = Q_1V_hR^{d\times h}$, where Φ represents the eigenvector matrix.

By following these steps, the QR-based PCA method allows for the projection of data into a lower-dimensional space while retaining the most significant variations. This approach is numerically stable and particularly useful for working with rectangular matrices.

6.6.2 Analysis

The paper analyses the difference between computational complexities of QR and LU.

6.6.3 Computation Complexity



(a) Analytical comparison (b) Numerical comparison

A comparison of cpu time between QR based PCA and SVD based PCA methods

Figure 13: Error with increase in size

Figure 1a shows the analytical comparison of flops between QR based PCA and SVD based PCA methods. Figure 1b shows the numerical comparison of their cpu times when implemented on Matlab software. It can be seen from this figure that the proposed method is computationally more efficient than the SVD based PCA method both analytically and Matlab-wise.

The proposed method offers a computational advantage by utilizing the QR decomposition of the covariance matrix. By decomposing the rectangular matrix H into an orthogonal matrix and an upper triangular matrix, the SVD procedure can be applied to the upper triangular matrix to compute the leading eigenvectors. This approach is more computationally efficient compared to applying SVD directly to the rectangular matrix H. The QR-based PCA method is particularly beneficial when dealing with low-rank rectangular matrices.

7 Applications of Matrix decomposition

Matrix decomposition, also known as matrix factorization or matrix factor analysis, is a powerful technique that breaks down a matrix into simpler and more interpretable components. This mathematical tool finds widespread applications in various fields, including recommender systems, digital signal processing, image processing, and data analysis.

Matrix decomposition techniques, such as Singular Value Decomposition (SVD) and Principal Component Analysis (PCA), enable us to extract valuable insights from complex data structures. By decomposing a matrix into its constituent parts, we can uncover latent factors, reduce dimensionality, and enhance computational efficiency.

In this section, we will explore two notable applications of matrix decomposition: recommender systems using SVD and digital signal processing using PCA. These applications demonstrate the versatility and effectiveness of matrix decomposition in solving real-world problems.

7.1 Recommender Systems using SVD

Recommender systems are widely used in various online platforms to provide personalized recommendations to users. One popular approach to building recommender systems is using Singular Value Decomposition (SVD), a matrix factorization technique. This subsection explores the application of SVD in recommender systems and its underlying mathematical principles.

7.1.1 Data Representation

The user-item interaction data can be represented as a matrix R, where $R(i, j)$ represents the rating or preference of user i for item j. The matrix R is typically sparse since users only rate or interact with a subset of items.

7.1.2 Matrix Factorization with SVD

SVD is applied to decompose the user-item matrix R into three separate matrices: U, Σ, and V^T. Mathematically, it can be represented as

$$R \approx U\Sigma V^T$$

where U is an $m \times k$ matrix representing users, Σ is a $k \times k$ diagonal matrix containing singular values, and V^T is a $k \times n$ matrix representing items.

7.1.3 Dimensionality Reduction

To reduce the dimensionality of the user-item matrix, we retain only the top k singular values and their corresponding singular vectors. This results in truncated or approximated matrices: U' is an $m \times k$ matrix, Σ' is a $k \times k$ diagonal matrix containing the top k singular values, and V'^T is a $k \times n$ matrix.

7.1.4 Latent Factor Calculation

The reduced matrices U', Σ', and V'^T obtained from SVD represent the latent factors in the user-item matrix. The latent factors capture the underlying characteristics and preferences of users and items. The latent representation of a user i can be calculated as $U'(i,:)$, and the latent representation of an item j can be calculated as $V'^T(:,j)$.

7.1.5 Recommendation Generation

Recommendations for a user i can be generated by calculating the similarity between their latent representation $U'(i,:)$ and the latent representations of items, represented as V'^T. This can be done using cosine similarity or Euclidean distance. For example, the similarity between user i and item j can be calculated as:

$$\text{similarity}(i,j) = \frac{U'(i,:) \cdot V'^T(:,j)}{\|U'(i,:)\| \|V'^T(:,j)\|}$$

Items that have the highest similarity to the user's latent representation are recommended as top suggestions.

7.1.6 Rating Prediction

SVD-based recommender systems can predict the ratings that users would give to items they have not interacted with. This is done by reconstructing the original user-item matrix using the reduced matrices U', Σ', and V'^T. The predicted rating of user i for item j can be calculated as:

$$\text{rating}(i,j) = U'(i,:) \cdot \Sigma' \cdot V'^T(:,j)$$

7.1.7 Evaluation and Iteration

The performance of the recommender system can be evaluated using metrics such as mean average precision, precision-recall curve, or root mean square error. Based on the evaluation results, the recommender system can be refined by adjusting parameters, incorporating additional data, or using regularization techniques to avoid overfitting.

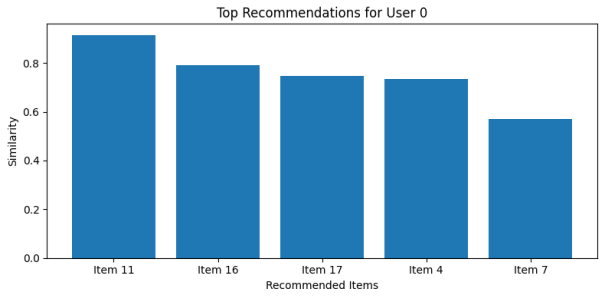


Figure 14: Recommendations of a user based on randomly generated data

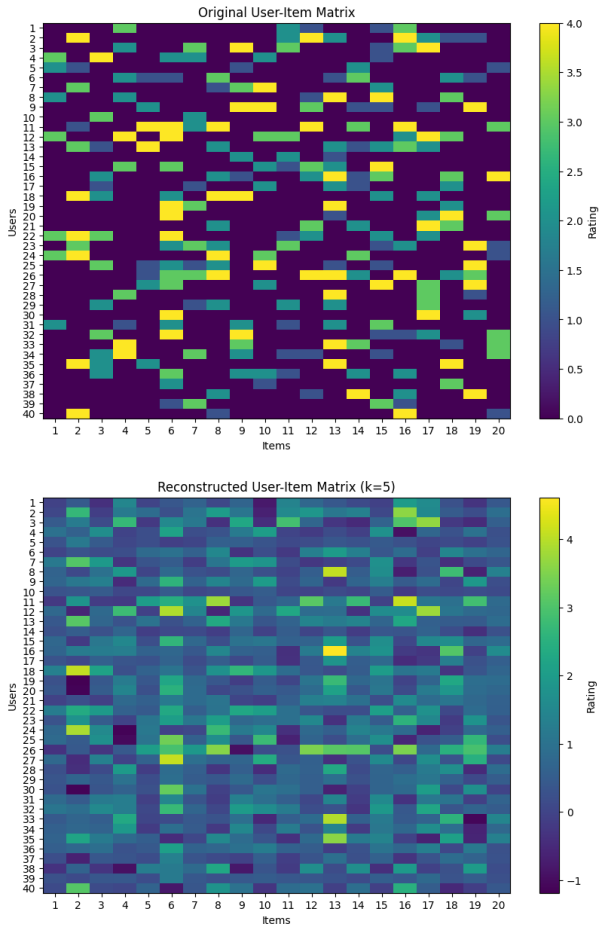


Figure 15: User-item matrix based on randomly generated data

In conclusion, SVD is a powerful tool that can be used to build effective recommender systems. By decomposing the user-item interaction matrix into three separate matrices, SVD can be used to identify latent factors that capture the underlying characteristics and preferences of users and items. These latent factors can then be used to generate recommendations for users. SVD-based recommender systems have been shown to be effective in a variety of domains, including music, movies, and books.

7.2 Digital Signal Processing

Digital signal processing (DSP) is a field of study that involves manipulating and analyzing digital signals using mathematical techniques. One of the fundamental concepts in DSP is matrix decomposition, which plays a crucial role in various signal processing algorithms. Matrix decomposition involves breaking down a matrix into simpler and more interpretable components, enabling efficient computations and analysis. Principal Component Analysis (PCA) is a statistical procedure that transforms a set of correlated variables (or features) into a new set of uncorrelated variables called principal components. These principal components capture the maximum amount of variance in the original data.

Here's an updated step-by-step explanation of how matrix decomposition using PCA can be applied in digital signal processing:

7.2.1 Signal Representation

A digital signal can be represented as a matrix, where each column or row represents a different aspect of the signal. For example, a time-domain signal can be represented as a matrix with one row or

column corresponding to the sample values at different time instances.

7.2.2 Matrix Formation

The digital signal matrix is constructed by arranging the samples appropriately. For instance, if a time-domain signal has N samples, the matrix would be of size $M \times N$, where M represents the number of channels or dimensions of the signal.

7.2.3 Data Centering

The signal matrix is centered by subtracting the mean value of each column (or row) from the corresponding elements. Centering is important to remove any bias or offset in the data.

7.2.4 Principal Component Analysis

PCA is applied to the centered signal matrix. The procedure involves calculating the covariance matrix of the centered data, given by

$$C = \frac{1}{N} \cdot X \cdot X^T$$

where X is the centered signal matrix and N is the number of samples.

7.2.5 Eigendecomposition

Eigendecomposition is performed on the covariance matrix C to obtain the eigenvectors and eigenvalues. The eigenvectors represent the directions in which the data varies the most, while the eigenvalues represent the amount of variance captured by each eigenvector. The eigenvectors can be obtained by solving the equation

$$C \cdot V = \lambda \cdot V$$

where V is the matrix of eigenvectors and λ is the diagonal matrix of eigenvalues.

7.2.6 Eigenvalue and Eigenvector Selection

The eigenvalues are sorted in descending order. By selecting the top K eigenvalues and their corresponding eigenvectors, the most significant principal components can be retained. These selected eigenvectors form a matrix V_k.

7.2.7 Dimensionality Reduction

The original signal matrix X can be transformed into a lower-dimensional space by multiplying it with the selected eigenvectors. The reduced-dimension representation of the signal can be obtained by

$$Y = V_k^T \cdot X$$

.

7.2.8 Reconstruction

To reconstruct the original signal from the reduced-dimension representation Y, the matrix multiplication of the reduced signal matrix with the transposed selected eigenvectors is performed. The reconstructed signal can be obtained as

$$X_{\text{reconstructed}} = V_k \cdot Y$$

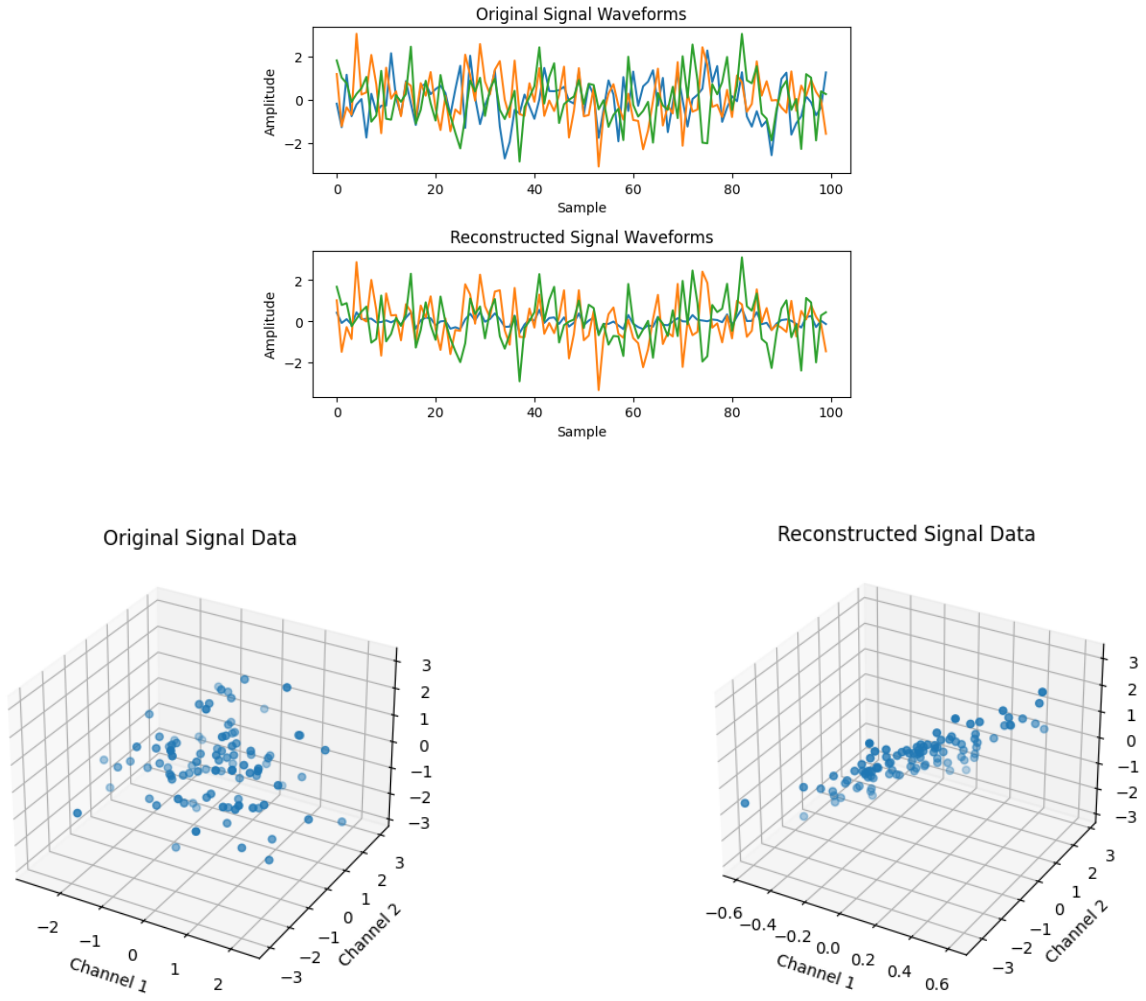


Figure 16: Digital Signal Processing on randomly generated data

PCA using matrix decomposition is a valuable tool in digital signal processing. It enables efficient computations, noise reduction, feature extraction, and dimensionality reduction by decomposing signals into uncorrelated components that capture the most significant variations in the data. Note: Python codes implementing both of these applications are included in a separate python notebook.

7.3 Other Applications

Apart from the two applications shown matrix decomposition has numerous another applications in various fields, including mathematics, computer science, physics, and engineering. Here are some other applications of matrix decomposition:

1. **Systems of Linear Equations:** Matrix decomposition techniques, such as LU decomposition and QR decomposition, can be used to solve systems of linear equations efficiently. By decomposing the coefficient matrix, the system can be transformed into a simpler form that is easier to solve.

2. **Eigenvalue Problems:** Matrix decomposition techniques, such as diagonalization and Schur decomposition, are used to solve eigenvalue problems. These problems arise in various applications, including quantum mechanics, structural analysis, and network analysis.

3. **Image Processing:** Matrix decomposition methods, like Non-negative Matrix Factorization (NMF), can be employed for image analysis and processing. NMF allows for extracting mean-

ingful features and components from images, which can be useful in tasks like image denoising, image recognition, and image compression.

4. **Graph Theory and Network Analysis:** Matrix decomposition techniques, such as spectral clustering and graph partitioning, are used in graph theory and network analysis. Decomposing the adjacency matrix of a graph allows for understanding its structural properties, identifying communities or clusters, and analyzing network dynamics.

5. **Control Systems:** Matrix decomposition plays a vital role in control theory and system dynamics. Techniques like modal decomposition (or modal analysis) and state-space representation involve decomposing matrices to analyze the behavior, stability, and controllability of complex systems.

6. **Data compression:** Matrix decomposition algorithms, including SVD and wavelet transforms, are essential in data compression techniques like JPEG (image compression) and MPEG (video compression). By decomposing the data into simpler components or exploiting redundancies, these methods can reduce the amount of information needed to represent the data efficiently.

8 Tentative Timeline Plan

The tentative timeline plan is as follows:

1. Phase 1: Familiarizing with the concepts of matrix decomposition, including Singular Value Decomposition (SVD), Eigenvalue Decomposition, LU Decomposition, and QR Decomposition, and conducting a literature review to understand the state of the art and gather relevant research papers and resources related to each decomposition technique.

2. Phase 2: Studying the theoretical aspects of matrix decomposition including mathematical foundations, properties, and algorithms associated with the various decompositions. Exploring practical applications like recommender systems, image compression, and data analysis using matrix decomposition techniques

3. Phase 3: Analyzing the computational aspects including computational complexity, efficiency, and numerical stability of the algorithms used in each decomposition technique. Developing Python code snippets to explain the application of SVD in Recommender Systems and PCA using Eigenvalue decomposition in Signal Processing.

4. Phase 4: Concluding the project by summarising the information gathered throughout the project into a comprehensive report and a presentation, including an overview of matrix decomposition techniques, complete algorithm analysis, and practical applications.

9 Individual Work Contribution

The individual work contribution for this project will be divided as follows:

- **Saravana Mitra Somayaji Tangirala (2022102016):** Implementation of SVD algorithm and eigenvalue decomposition algorithm, performance evaluation, and comparative analysis.

- **Vyakhya Gupta (2022101104):** Implementation of LU decomposition algorithm and QR decomposition algorithm and and their performance evaluation, and comparative analysis.

- **Khooshi Asmi(2022114006):** Exploration of applications of matrix decomposition in various fields, signal processing using PCA and Recommender Systems using SVD. Compilation of individual work in LaTeX.

10 Conclusion

In this project, we have tried tp explain the mathematical formulations of several matrix decomposition methods, such as SVD, Eigenvalue Decomposition, LU Decomposition, and QR Decomposition. We also demonstrated the use of matrix decomposition techniques and explore their practical applications in fields like Signal Processing and Machine learning in Recommender systems.

11 References

LU inverses: https://web.mit.edu/18.06/www/Spring17/LU-and-Inverses.pdf

SuperLU: https://www.osti.gov/biblio/860217

Solving AX=B using LU: Numerical Analysis by Timothy Sauer, 2nd or 3rd edition.

Scipy library; https://scipy-lectures.org/

QR decomposition using Gram Schmidt: https://www.math.ucla.edu/ yanovsky/Teaching/Math151B/handouts/Gram

QR algorithms: https://people.inf.ethz.ch/gander/papers/qrneu.pdf

PTLU: https://www.sciencedirect.com/science/article/abs/pii/0196677482900074