Date: __/__/20__ Day: _____

HAND WRITTEN NOTES

LINAR Algebra In ML/AI

Written by:

MIRZA Yasir Abdullah Baig

All topics from basics to Advanced Level

Topics

- Matrices
- Eigenvalues and Eigenvectors
- LU Decomposition
- QR Decomposition
- Singular Value Decomposition (SVD)
- Orthogonalization
- Diagonalization
- Non-Negative Matrix Factorization
- Vectors
- Vector Spaces and Subspaces
- Linear Mapping

Date: __/__/20 Day: _____

Introduction

What is Linear Algebra in Machine Learning?

Linear Algebera is a branch of mathematics that deals with vectors, matrices, and operations on them.

In machine learning (ML), data is often represented as numbers inside vectors and matrices.

Example:

- An image = a big matrix of pixels values
- A datast = a matrix (rows = examples, colpumns = features)
- Neural network = use lot of matrix multiplication to learn.

Why do we use Linear Algebra in ML, AI & DS?

- Data Representation: Store and organize data in vectors
- Transformations: Rotate, scale, or project data
- Model Training: Algorithms like regression, SVMs & DL
- Efficiency: Computers can process large datasets quickly using linear algebra rules.

Purpose of Linear Algebra in AI

The main purpose of is to give us the tools to:

- Represent data and models mathematically
- Perform operations (like dot products, eigenvalues) to extract insights.
- Enable optimizations: for training ML and AI systems.

Date: __/__/20__ Day: _____

Simple Introduction

Think of linear Algebra as the language of AI and ML.

- Scalars: single numbers (like temperature)
- Vectors: list of numbers (like a point in space)
- Matrices: Grid of numbers (like a spreadsheet)
- Tensors: Multi-dimensional arrays (used in dl)

ML = take your data (matrices) + apply operations (linear Algebra) + get predictions

Example in ML:

Predicting house prices:

- Features (sizes, rooms, location) → vector
- Dataset (many houses) → matrix
- Multiply with weights → prediction

Matrices

A matrix is a way to organize numbers in a rectangular grid made up of rows and columns We can assume it like table where:

- Rows go across (left to right)
- Columns go down (top to bottom)
- Size → Defined as rows x columns (3x4 matrix means 3 rows, 4columns)

Uses of matrices:

- Solve linear equations
- Image transformations (rotate, scale)
- Machine learning
- Data Storage

Date: __/__/20__ Day: _____

Creating a Matrices in Python

Method 1: Lists of Lists.

A matrix can be made using 2D lists

```
mat = [[1, 2, 3, 4],
       [5, 6, 7, 8],
       [9, 10, 11, 12]]
print("Matrix = ", mat)
```

Output: [[1, 2, 3, 4,], [5, 6, 7, 8], [9, 10, 11, 12]]

Method 2: User Input

Take rows, columns, and elements from the user

```
row = int(input("rows: "))
col = int(input("columns: "))
matrix = []
print("entries row-wise: ")
for i in range(rows):
    row = []
    for j in range(col):
        row.append(int(input()))
    matrix.append(row)
print("\n2D matrix is: ")
for i in range(rows):
    for j in range(col):
        print(matrix[i][j], end=" ")
    print()
```

Method 3; List Comprehension

Quick way to built a matrix

```
matrix = [[col for col in range(4)] for row in
          range(4)]
```

Date: __/__/20__ Day: _____

print(matrix)

Output: [[0, 1, 2, 3], [0, 1, 2, 3], [0, 1, 2, 3], [0, 1, 2, 3]]

Assigning Value in Matrix

Method 1: Direct Assignment

```
x = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
x[1][1] = 11
print(x)
```

Method 2: Negative Indexing

```
x = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
x[-2][-1] = 21
print(x)
```

Accessing Value In Matrix

Method 1: Indexing

```
print(x[0][2])  # 3
print(x[2][2])  # 9
```

Method 2: Negative Indexing

```
print(x[-1][-2])  # 8
```

Math with Matrices (without Numpy)

Example 1: Addition Loop

```
x = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
y = [[9, 8, 7], [6, 5, 4], [3, 2, 1]]
res = [[0] * 3 for _ in range(3)]
for i in range(len(x)):
    for j in range(len(x[0])):
        res[i][j] = x[i][j] + y[i][j]
for r in res:
    print(r)
```

Date: __/__/20 Day: _____

Example 2: Addition and Substraction (list Comprehsion)

```
x = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
y = [[9, 8, 7], [6, 5, 4], [3, 2, 1]]

add_res = [[x[i][j] + y[i][j] for j in range(3)] for
          i in range(3)]
sub_res = [[x[i][j] - y[i][j] fo j in range(3)] for
          i in range(3)]
```

Example 3: Multiplication and division

```
x = [[2, 4, 6], [8, 10, 12], [14, 16, 18]]
y = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

mult_res = [[x[i][j] * y[i][j] for j in range(3)] for i
          in range(3)]
div_res = [[x[i][j] / y[i][j] for j in range(3)] fo i
          in range(3)]
```

Transpose of Matrix

Using loop

```
x = [[9, 8, 7], [6, 5, 4], [3, 2, 1]]
transpose = [[0] * 3 for _ in range(3)]
for i in range(len(x)):
    for j in range(len(x[0])):
        transpose[j][i] = x[i][j]
```

Using list comphrehsion

```
transpose = [x[j][i] for j in range(len(x))] for
          i in range(len(x[0]))
```

Date: __/__/20__ Day: _____

NumPy Matrix Operations

Random Matrix

```
import numpy as np
arr = np.random.randint(10, size = (3,3))
print(arr)
```

Basic Math with Numpy

```
x = nparray([[1, 2], [4, 5]])
y = np.array([[7, 8], [9, 10]]
print (np.add(x, y))
print (np.substract(x, y))
print (np.multiply (x, y))
print (np.divide( x, y))
```

Dot and Cross Product

```
x = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
y = [[9, 8, 7], [6, 5, 4,], [3, 2, 1]]
print (np.dot(x, y))
print( np.cross(x, y))
```

Transpose

```
matrix = [[1, 2, 3], [4, 5, 6]]
print (np.transpose (matrix))
```

Empty Matrices

```
a = np.zeros((2, 2), dtype = int)
b= np.zeros ((3,3))
```

Slicing

```
x = np.array([[ 6, 8,10], [9, -12, 15], [12, 16, 20], [15, -20, 25]])
print(x[2:3, 1]) * 16
print(x[2:3, 2]) * 20
```

Date: __/__/20__ Day: _____

Deleting Rows

```
a = np.array([[6, 8, 10], [9, -12, 15], [12, 16, 20], [15, -20, 25]])
print(np.delete(a, 0, axis=0)) # delete 0th row
```

Adding Rows/ Columns

```
x = np.array([[6, 8, 10], [9, -12, 15] [15, -20, 25]])
col.. = np.array([1, 2, 3])
res = np.hstack((x, np.atleast_2d(col).T))
```

Eigenvalues and Eigenvectors

Eigenvalues and eigenvectors are very important in linear Algebra. They are used in matrix diagonalization, stability analysis, and data analysis (like PCA). They are connected with square matrices and give insights into their properties

Eigenvalues:

An eigenvalues is a special scalar (number) linked to a matrix. It tells how much an ~~eigenvalues~~ eigenvectors is stretched or shrunk when multiplied by the matrix.

- If eigenvalue $\lambda > 0$ → vector keep its direction
- If $\lambda < 0$ → direction is reversed.

This Formula is

$$Av = \lambda v$$

Where

A = matrix

v = eigenvector

λ = eigenvalue

To Find eigenvalues, we solve:

$$\det(A - \lambda I) = 0$$

Date: __/__/20__ Day: _____

EigenVectors

- An eigenvectors is a non-zero vector that only streches/ shrinks but does not change direction when multiplied by matrix.

There are two types

- Right eigenvector: $Av = \lambda v$ (column vector, $n \times 1$)
- Left eigenvector: $vA = \lambda v$ (row vector, $1 \times n$)

How to Find Eigenvectors

- Find eigenvalues using $\det(A - \lambda I) = 0$
- Solve for $\lambda_1, \lambda_2, \lambda_3 \ldots$
- For each λ, solve:

$$(A - \lambda I)X = 0$$

to get eigenvector

- Repeat for all eigenvalues.

Types of Eigenvalues vectors

1. Right Eigenvalues vectors

$$AV_R = \lambda V_R$$

$$V_R = [v_1, v_2, v_3 \ldots\ldots v_n]^T$$

2- Left Eigenvector

$$V_L A = \lambda V_L$$

$$V_L = [v_1, v_2, v_3 \ldots\ldots v_n]$$

Eigenvectors of Square Matrices

1- Eigenvector of a 2×2 Matrix

Example: $A = \begin{bmatrix} 1 & 2 \\ 5 & 4 \end{bmatrix}$

Step 1: Solve $\det(A - \lambda I) = 0$

$$\lambda^2 - 5\lambda - 6 = 0 \Rightarrow \lambda = 6, -1$$

Date: __/__/20__ Day: _____

Step 2: For each λ, solve $(A - \lambda I)v = 0$

- For $\lambda = 6 \rightarrow$ eigenvector $= [2, 5]^T$
- For $\lambda = -1 \rightarrow$ eigenvector $= [1, -1]^T$

So, eigenvectors are multiples of:

$$[2, 5]^T, \quad [1, -1]^T$$

2- Eigenvector of a 3×3 Matrix

Example:

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

Step 1: Solve $\det(A - \lambda I) = 0$

$$\lambda^2(6 - \lambda) = 0 \Rightarrow \lambda = 0, 0, 6$$

Step 2: Find eigenvectors

- For $\lambda = 0 \rightarrow a + b + c = 0$, eigenvectors like $[-1, 1, 0]^T, [-1, 0, 1]^T$
- For $\lambda = 6 \rightarrow$ eigenvector $= [1, 1, 1]^T$

So, eigenvectors are:

$$[-1, 1, 0]^T, \quad [-1, 0, 1]^T, \quad [1, 1, 1]^T$$

Eigenspace

The eigenspace is the set of all eigenvectors corresponding to the eigenvalues.

For the 3×3 Example

$$\{[-1, 1, 0]^T, [-1, 0, 1]^T, [1, 1, 1]^T\}$$

Diagonalization

A matrix can be diagonalized using eigenvalues and eigenvectors.

$$A = XDX^{-1}$$

where

- D = Diagonal matrix with eigenvalues
- X = matrix of eigenvector
- X^{-1} = inverse of X

Date: __/__/20__ Day: _____

Example (3×3 matrix above)

$$D = \begin{bmatrix} 6 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad X = \begin{bmatrix} 1 & -1 & -1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

Applications of Eigenvalues and Eigenvectors

- Google pagerank: Importance of web pages.
- Markov chain: Steady-state probabilities
- PCA (Principal Component Analysis): Dimensionality reduction
- NLP (Latent Semantic Analysis): Finding word-doc relations
- Graph Theory: Community detection, connectivity
- Computer Vision (Eigenfaces): Face recognition
- Control Systems: Stability of robots / AI
- Signal Processing: Noise filtering, channel optimization

Solved Example

Example 1

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

Eigenvalues: (1 repeated three times)

Eigenvector (for $\lambda = 1$): $[1, 0, 0]^T$

Example 2

$$A = \begin{bmatrix} 5 & 0 \\ 0 & 5 \end{bmatrix}$$

Eigenvalues, $\lambda = 5, 5$

Eigenvectors:

$[1, 0]^T$, $[0, 1]^T$

Date: __/__/20__ Day: _____

What is LU Decomposition?

It expresses a square matrix A as:

$$A = L \times U$$

- L → Lower triangular (all values below diagonal, 1s on diagonal)
- U → Upper triangular (all values above diagonal)

Example 1

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

Step1: Gaussian elimination

Subtract 6/4 × Row 1 from Row 2

$$U = \begin{bmatrix} 4 & 3 \\ 0 & -1.5 \end{bmatrix}$$

Step 2: Find L

$$L = \begin{bmatrix} 1 & 0 \\ 1.5 & 1 \end{bmatrix}$$

Step 3: Verify

$$A = L \times U = \begin{bmatrix} 4 & 3 \\ 6 & 3 \end{bmatrix}$$

Works! So

$$L = \begin{bmatrix} 1 & 0 \\ 1.5 & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 4 & 3 \\ 0 & -1.5 \end{bmatrix}$$

Steps For LU Decomposition

- Start with a square matrix A
- Use Gaussian elimination to make it into an upper triangular matrix, U.
- Track the multipliers you used → they become entries of L
- L always has 1s on its diagonal
- Verify that

$$A = L \times U$$

Date: __/__/20__ Day: _____

Example 2: Solve a system using LU

Solve:

$$\begin{cases} x_1 + x_2 + x_3 = 1 \\ 4x_1 + 3x_2 - x_3 = 6 \\ 3x_1 + 5x_2 + 3x_3 = 4 \end{cases}$$

Matrix form:

$$AX = C$$

Where:

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 4 & 3 & -1 \\ 3 & 5 & 3 \end{bmatrix}, \quad X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, \quad C = \begin{bmatrix} 1 \\ 6 \\ 4 \end{bmatrix}$$

Step 1: Gaussian elimination → Find U

After elimination

$$U = \begin{bmatrix} 1 & 1 & 1 \\ 0 & -1 & -5 \\ 0 & 0 & -10 \end{bmatrix}$$

Step 2: Built L

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 4 & 1 & 0 \\ 3 & -2 & 1 \end{bmatrix}$$

Step 3: Solve $LZ = C$

$$Z = \begin{bmatrix} 1 \\ 2 \\ 5 \end{bmatrix}$$

Step 4: Solve $UX = Z$

$$X = \begin{bmatrix} 1 \\ 0.5 \\ -0.5 \end{bmatrix}$$

Final Answer:

$$x_1 = 1, \quad x_2 = 0.5, \quad x_3 = -0.5$$

Date: __/__/20__ Day: _____

Applications of LU Decomposition

- Structural Engineering: bridge & building analysis
- Computer Graphics: rotations, scaling, of 3D objects
- Robotics: Solving movement equations
- Weather Prediction: Foster climate model simulations
- Electrical Engineering: Circuit Analysis
- Economics / Finance: Solving large economic models

QR Decomposition

QR Decomposition is a way of breaking a matrix into two simpler matrices

- Q → an orthogonal matrix (columns are orthonormal vector)
- R → an upper triangular matrix

So, for a matrix, A, we write

$$A = QR$$

What is QR Decomposition?

Decomposition (or factorization) means breaking one object into parts to make it easier to work with.

QR decomposition is widely used in linear Algebra, optimization, and machine learning. It helps in solving equations, least squares problems, eigenvalue computations, and more. It is known for being numerically, stable and efficient.

Key Concepts:

- Matrix Factorization: Writing one matrix as a product of two or more.
- Orthogonal Matrix (Q): Has the property $Q^TQ = I$

Date: __/__/20__ Day: _____

preseves Vector lengths and dot products

- Upper Triangleiar Matrix (R): All entries below the main diagonal are zero.
- Gram-Schmidt Process: A method to make a set of vectors orthogonal. In QR, it is applied to the columns of A.

How QR Decomposition Works.

1- Gram-Schmidt Orthogonalization:

Given a matrix $A = [a_1, a_2, \cdots a_n]$

- Start with $q_1 = \frac{a_1}{\|a_1\|}$
- For each next column q_i: subtract its projections also onto the earlier q_j vectors to make it orthogonal, then normalize it.

$$v_i = q_i - \sum_{j=1}^{i-1} \text{Proj}_{q_j}(a_i), \quad q_i = \frac{v_i}{\|v_i\|}$$

This gives orthogonal vectors, $q_1, q_2 \cdots q_n$

The Q matrix is formed as $Q = [q_1, q_2 \cdots q_n]$

2- Forming R

Once Q is found

$$R = Q^T A$$

This gives the upper triangular matrix

So finally

$$A = QR$$

Example with python

```
import numpy as np
A = np.array([[1, 2, 4],
```

Date: __/__/20 Day: _____

```
                [0, 0, 5],
                [0, 3, 6]])
Q, R = np.linalg.qr(A)
print("Q = \n", Q)
print("R = \n", R)
print(np.allclose(A, np.dot(Q, R)))
```

output:

```
Q = [[ 1.  0.  0.]
     [ 0.  0. -1.]
     [ 0. -1.  0.]]
R = [[ 1.  2.  4.]
     [ 0. -3. -6.]
     [ 0.  0. -5.]]
True
```

Manual Example: (Gram-Schmidt)

Matrix:

$$A = \begin{bmatrix} 1 & 2 & 4 \\ 0 & 0 & 5 \\ 0 & 3 & 6 \end{bmatrix}$$

- Normalize column 1 → $q_1 = [1, 0, 0]^T$
- Orthogonalize column 2 → residual = [0, 0, 3], normalize → $q_2 = [0, 0, 1]^T$
- Orthogonalize column 3 → residual = [0, 5, 0], normalize → $q_3 = [0, 1, 0]^T$

So,

$$Q = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, R = \begin{bmatrix} 1 & 2 & 4 \\ 0 & 3 & 6 \\ 0 & 0 & 5 \end{bmatrix}$$

Note: Numpy may give slightly different signs

Date _/_ /20 Day: _____

(because QR is not, unique; signs of Q an R can flip)

Applications In Machine learning:

- Linear Regression and least Squares: Used to solve overdetermined systems stably.
- Feature Selection and Rank Deficiency: Helps detect linear dependencies in features
- Principal Component Analysis (PCA): Helps compute principal components efficienay.
- Regularization (e.g Ridge Regression): Used for stable solutions.
- Eigenvalue Problem: Forms part of eigenvalue algorithm
- Signal Processing: Used in filtering and denoising

Advantages & Disadvantages

Advantage:

- Numerically stable and efficient
- Works for both square & rectangular matrix (unlike LU)

Disadvantage:

- Computationally heavy ($O(n^3)$ per step)
- Can converge slowly if eigenvalues are very close

Singular Value Decomposition (SVD)

Singular Value Decomposition (SVD) is a way in linear algebra to break a matrix into three parts. It helps us understand data in terms of singular values (which tell us how important each direction or factor is)

For a matrix A:

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

Date: __/__/20__ Day: _____

- U → tells us about the rows (like peoples preferences)
- Σ (Sigma) → diagonal matrix with singular values (importance of each factor

Simple Example

Suppose we have ratings of two movies,

Name	Movie 1	Movie 2
Amit	5	3
Sanket	4	2
Harsh	2	5

SVD breaks this into:

- U: people's preferences
- Σ: importance of movies
- V^T: similarity b/w the two movies.

The Math behind SVD

For an $m \times n$ matrix A:

- U: $m \times m$ orthogonal matrix (left singular vectors) order
- Σ: $m \times n$ diagonal matrix with singular values in descending
- V^T: transpose of $n \times n$ orthogonal matrix (right singular vectors))

Step-by-step Example

Matrix:

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

Step 1: Compute AA^T

$$AA^T = \begin{bmatrix} 17 & 8 \\ 8 & 17 \end{bmatrix}$$

Step 2: Find Eigenvalues of AA^T

Date: _/_/20 Day: _____

Characteristics equation gives eigenvalues:

$$\lambda_1 = 25, \quad \lambda_2 = 9$$

So singular values:

$$\sigma_1 = 5, \quad \sigma_2 = 3$$

Step 3: Find Right Singular Vectors (from A^TA)

Solve $(A^TA - \lambda I)v = 0$:

- For $\lambda = 25$: $v_1 = \begin{bmatrix} 1/2 \\ 1/2 \\ 0 \end{bmatrix}$
- For $\lambda = 9$: $v_2 = \begin{bmatrix} 1/18 \\ -1/18 \\ 4/8 \end{bmatrix}$
- A third vector v_3 is perpendicular to both: $v_3 = \begin{bmatrix} 2/3 \\ 2/3 \\ -1/3 \end{bmatrix}$

So, $V = [v_1, v_2, v_3]$

Step 4: Compute Left singular Vectors (U)

Formula:

$$u_i = \frac{1}{\sigma_i} A v_i$$

This gives:

$$U = \begin{bmatrix} 1/2 & 1/2 \\ 1/2 & -1/2 \end{bmatrix}$$

Step 5: Final SVD

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

Where:

$$U = \begin{bmatrix} 1/2 & 1/2 \\ 1/2 & -1/2 \end{bmatrix}, \quad \Sigma = \begin{bmatrix} 5 & 0 \\ 0 & 3 \end{bmatrix}$$

Applications of SVD

- Pseudo-inverse (Moore-Penrose Inverse)
  - Used when a matrix is not invertible
  - Formula: $M^+ = V\Sigma^{-1}U^T$

Date: / /20 Day:

- Solving Linear Equations
  - For $Mx = b$, solution is $x = M^+b$
- Rank, Range, Null Space
  - Rank = number of non-zero singular values
  - Range = Span of columns in U (for non-zero values).
  - Null Space = span of vectors in V for zero singular value
- Curve Fitting (Least Squares)
  - SVD minimizes error and finds best-fit curves.
- Digital Signal Processing (DSP) & Image Processing
  - Noise reduction, compression, denoising
  - Keep only the largest singular values → smaller file; similar image.

Python Implementation

```
import numpy as np
from scipy.linalg import svd
import matplotlib.pyplot as plt
from skimage.color import rgb2gray
from skimage import data
X = np.array([[3, 3, 2], [2, 3, -2]])
U, S, Vt = svd(X)
print("U: ", U)
print("Singular Values: ", S)
print("V^T: ", Vt)
cat = data.chelsea()
gray_cat = rgb2gray(cat)
U, S, Vt = svd(gray_cat, full_matrices = False)
S = np.diag(S)
for r in [5, 10, 70, 100, 200]:
    approx = U[:, :r] @ S[:r, :r] @ Vt[:r, :]
    plt.imshow(approx, cmap = "gray")
```

Date: _/__/20 Day: _____

```
plt.title(f"K = {r}")
plt axis('off')
plt.show()
```

Output

- Print U, Σ, V^T.
- Shows compressed images for different value of r.
  - Small r: blurry, blocky
  - Large r: closer to original image

Summary

- SVD = $U\Sigma V^T$ (decomposition of matrix)
- Helps in pseudo-inverse, solving equations, rank calculation, curve fitting.
- Widely used in machine learning, signal processing, and image compression
- Keeping only top singular values gives smaller storage with little less of quality.

Orthogonalization

Orthogonalization is a concept from linear algebra that helps simplify machine learning models. It makes models easier to understand, debug and optimize. In this article, we'will cover what orthonogalization is, main techniques & its applications in ML.

What is Orthogonalization?

Organi Orthogonalization means finding a new set of orthogonal (independent) vectors that span the same space as the original vectors

Given vectors $a_1, \ldots, a_k \in R^n$, orthogonalization finds

Date: __/__/20 Day: _____

vectors $q_1, \ldots, q_r$ such that:

$$\text{span}(a_1, \ldots, a_k) = \text{span}(q_1, \ldots, q_r)$$

Here, r = dimension of subspace

The resulting vectors satisfy:

- $q_i^t q_j = 0$ if $i \neq j$ (orthogonality)
- $q_i^T q_i = 1$ (unit length)

In simple words: orthogonalization gives an orthonormal basis for the space as the original vectors

Orthogonalization Techniques in ML

1- Gram- Schmidt Process

A step by step method to turn a set of vectors into orthogonal ones by substracting projections of earlier vectors

2- QR Decomposition

Factorizes a matrix into:

$$A = QR$$

Q: orthogonal matrix

R: upper triangular matrix

- Useful for solving linear systems, eigenvalue problems and least squares

3- Principal Component Analysis (PCA)

A dimensionally reduction method. Projects data into principal components (orthogonal eigenvectors of the covariance matrix). Widely used For feature extraction and visualization.

4- Singular Value Decomposition (SVD)

Date: __/__/20__ Day: _____

Factorizes a matrix into:

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

- U, V: orthogonal matrices
- Σ: diagonal matrices with singular values
- Applications: image compression, recommandation, systems, data processing.

5. Lattice Reduction

Used in cyptography and communications. Finds a lattice basis with shorter, more orthogonal vectors, improves efficiency and seaurity.

6. Gram Matrix and Cholesky Decomposition

- Gram Matrix: measures pairwise similarities of vectors
- Cholesky Decomposition: breaks a positive-definite matrix into a lower triangular matrix and its transpose. Helps solve optimization and linear systems efficiency.

Applications in Machine Learning

- Feature Engineering: PCA, one-hot encoding, create independent (orthogonal) features.
- Model Architecture: Breaking tasks into separate layers/components makes models easier to design & maintain.
- Optimization and Regularization: Using orthogonal techniques (e.g decoupling learning rates, L1/L2 regularization) → more stable training and better generalization

Date: __/__/20__ Day: _____

Benefits of Orthogonalization.

- Better Performance: Reduces complexity and makes each part of the model work efficiently.
- Easier Debugging: Problems in one component don't affect others.
- Scalability: New components can be added without breaking existing ones.

Diagonalization

Matrix diagonalization is the process of turning a square matrix into a diagonal matrix using a similarity transformation.

Diagonal matrices are very useful because they are much easier to work with, especially when raising them to powers.

Note: Not every matrix can be diagonalized. A matrix is diagonalizable only if every eigenvalue has equal geomatric multiplicity (number of independent eigenvectors) and algebra multiplicity (its repetition in characteristics equation)

Matrix Similarity Transformation

Two matrices A and B are called similar if there is an invertible matrix P such that:

$$B = P^{-1}AP$$

- Similar matrices have same rank, determinant, trace and eigenvalues.
- Eigenvalues also keep their multiplicities.

Date: __/__/20__ Day: _____

Diagonalization of Matrix

If a matrix A is diagonalizable, then:

$$D = P^{-1}AP$$

- D is a diagonal matrix
- P is called model matrix (made from ~~eigenvalue~~ (eigenvectors of A)

In short: Diagonalization means converting a square matrix into diagonal one using its eigenvectors

Steps to Diagonalize in Matrix

Step 1: Write the diagonal matrix:

$$D = \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{bmatrix}$$

Step 2: Find eigenvalues from:

$$\det(A - \lambda I) = 0$$

Step 3: Find eigenvectors for each eigenvalue by solving:

$$(A - \lambda I)x = 0$$

Step 4: Form model matrix P by putting eigenvectors as columns:

$$P = [x_1, x_2, \cdots, x_n]$$

Step 5: Compute P^{-1}, then use:

$$D = P^{-1}AP$$

Example Problem

Given:

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

Step 1: Setup diagonal form D

Step 2: solve $\det(A - \lambda I) = 0$

$$(\lambda - 1)(\lambda - 2)(\lambda - 3) = 0$$

Date: __/__/20__ Day: _____

So, eigenvalues = 1, 2, 3

Step 3: Find eigenvectors

- For $\lambda = 1$: $X_1 = [1, -1, 0]^T$
- For $\lambda = 2$: $X_2 = [-2, 1, 2]^T$
- For $\lambda = 3$: $X_3 = [1, -1, -2]^T$

Step 4: Form model matrix:

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

Step 5: Since $\det(P) \neq 0$, P is invertible. Compute:

$$D = P^{-1}AP = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix}$$

Done, Matrix is diagonalized

Matlab Implementation

```
clear all
clc
A = input ("Enter a matrix A : ");
[P, D] = eig (A);
D1 = inv(P) * A * P;
disp ("Diagonal Form 'D' is : ")
disp(D1)
```

Diagonalization is Not Unique

- The order of eigenvalues in D can be changed
- Eigenvectors in P can be multiplied by scalars.
- If eigenvalues repeat, we can choose different bases for their eigenspaces.
- So, diagonalization gives different valid answers, but all represent the same property.

Date: _/_/20 Day: _____

Inverse Matrix with Diagonalization:

If A is diagonalizable and invertible.

$$A^{-1} = PD^{-1}P^{-1}$$

where

$$D^{-1} = \text{dia}\left(1/\lambda_1, 1/\lambda_2, \ldots, 1/\lambda_n\right)$$

(valid only if no eigenvalue = 0)

Non-Negative Matrix Factorization

Non-Negative Matrix Factorization (NMF) is a technique to break a large dataset into smaller, useful parts while keeping all values non-negative (≥ 0).

It helps in extracting important features and makes data easier to analyze

Matrix Decomposition in NMF

For a matrix $A(m \times n)$ where all values ≥ 0, NMF factorizes it into two matrices $W(m \times K)$ and $H(K \times n)$:

$$A \approx W \times H$$

- A → Original Data
- W → Feature matrix (basis components)
- H → Coefficient matrix (weights of features)
- K → Rank (reduced dimension, $K \leq \min(m, n)$)

Idea: Each column of A (a data point) is written as a combination of features from W, weighted by H.

Date: __/__/20 Day: _____

Intuition Behind NMF

- NMF simplifies complex data into fewer, meaningful patterns.
- Each data point is a combination of non-negative features.
- Example: In facial recognition, NMF breaks an image into parts like eyes, nose, mouth,
  - W = Features parts
  - H = How strongly each part is used in image

Working of NMF

NMF uses an iterative optimization process to minimize reconstruction error b/w A and WxH

Steps:

- Initialization: Start with random non-negative W & H.
- Iterative Update - Adjust W and H to reduce the error
- Stopping Criteria: Stop when:
  - Error become stable, or
  - A maximum number of iterations is reached.

Techniques Used:

- Multiplicative Update Rules: → Keep values non-negative while updating.
- Alternating Least Squares (ALS): Solve for W while keeping H fixed, then solve for H while keeping W fixed.

Real-Life- Example:

- Suppose matrix A = pixel values of face image
- W = Feature set (eyes, nose, mouth)
- H = Weights showing how strongly each feature forms the image

Date: __/__/20__ Day: _____

- Result: The image is represented as a combination of its parts

Applications of NMF

- Image Processing: → Feature extraction in faces and objects
- Text Mining and NLP: Topic modeling from document-term matrices
- Spectral Data Analysis → Finding hidden patterns in sound, medical, or chemical signals
- Bioinformatices: → Gene expression analysis to find biological patterns

Conclusion

NMF is useful for discovering hidden stutures in data while keeping results easy to interpt (non-negative values)

By breaking data into meaningful features (W) and their weights (H), NMF supports.

- Feature Extraction
- Dimensionally reduction
- Pattern recognition

Vectors

How to Create a Vector in Python using Numpy

A vector is a 1-D array that can store numbers, coordinates, or measurements. In numpy, vectors are created as 1-D arrays. We can easily perform math operations like addition, substraction, scalar multiplication, and dot products

Date: __/__/20__ Day: _____

Creating Vectors in Numpy

There are different ways to create vectors.
Let's see the most common ones.

1- Using np.array()

Convert a python list into Numpy array

Syntax:

```
np.array(list)
```

Example

```
import numpy as np
list1 = [1, 2, 3]
list2 = [[10], [20], [30]]
vector1 = np.array(list1)
vector2 = np.array(list2)
print("Horizental Vector:", vector1)
print("Vertical Vector:", vector2)
```

2- Using np.arange()

Creates a sequence of regularly spaced values

Syntax

```
np.arange(start, stop, step)
```

Example

```
Import numpy as np
vector = np.arange(1, 6)
print("Vector Using np.arange():", vector)
```

* [1, 2, 3, 4, 5]

3. Using np.linspace()

Creating evenly spaced values in a ronge

Date: _/_/20 Day: _____

Syntax:

np.linspace(start, stop, num = 50)

Example

```
import numpy as np
vector = np.linspace(0, 10, 5)
print("Vector using np.linspace():", vector)
```

* [0. 2.5 5. 7.5 10.]

4. Using np.zeros() and np.ones()

Creates vectors filled with 0s or 1s

Syntax

np.zeros(shape)

np.ones(shape)

Example

```
import numpy as np
vector_zeros = np.zeros(5)
print("vector using np.zeros():", vector_zeros)
vector_ones = np.ones(5)
print("vector using np.ones():", vector_ones)
```

Operations on Vectors

- Arithemetic Operations (Element-Vise)

```
import numpy as np
vector1 = np.array([5, 6, 9])
vector2 = np.array([1, 2, 3])
print("Add :", vector1 + vector2)
print("Sub :", vector1 - vector2)
print("Multi :", vector1 * vector2)
Print("Div :", vector1 / vector2)
```

Date: __/__/20 Day: _____

2. Dot Product

The dot product multiplies elements and sums them giving a scalar

```
import numpy as np
vector1 = np.array([5, 6, 9])
vector2 = np.array([1, 2, 3])
dot_product = vector1.dot(vector2)
print("Dot product:", dot_product)
```

- Vector-Scalar Multiplication

Multiply every element of a vector by a scalar

```
import numpy as np
vector = np.array([1, 2, 3])
scalar = 2
print("Scalar Multiplication:", vector * scalar)
```

Conclusion

- A vector in Numpy is just a 1-D array
- We can create it using np.array(), np.arange(), np.linspace(), np.zeros() or np.ones().
- We can perform arithematic operations, dot products, and scalar multiplication easily.

Vector Spaces and Subspaces

Vector Space:

A vector space V over a field F is a set of vectors that is closed under addition and scalar multiplication.

Those operations follow 10 roles (axioms). Vector spaces are used in linear Algebra, geometry, physics, computer science and machine learning.

Date: __/__/20 Day: _____

Operations in Vector Space

- Vector Addition: Adding two vectors $u, v \in V$ gives another vector $u + v \in V$.
- Scalar Multiplication: Multiplying a scalar $c \in F$ with a vector $v \in V$ gives a new vector $cv \in V$.

Real-life - Example

- Position: Describe Location
- Movement: Shows direction and distance
- Forces: Show both magnitude and direction
- Graphics/ Animation: Define shapes and motion

Ten Axioms of Vector Space

let $x, y, z \in V$ and $a, b \in F$

- Closed under addition: $-x + y \in V$
- Closed under Scalar Multiplication $-ax \in V$
- Commutative Addition $-x + y = y + x$
- Associative Addition $-(x+y) + z = x + (y + z)$
- Addictive Identify - There is 0 such that $x + 0 = x$
- Addictive Inverse - For each x, there is $-x$ like $x + (-x) = 0$
- Multiplicative Identify - $1x = x$
- Associatively of scalar Multiplication - $(ab)x = a(bx)$
- Distributive Property 1 - $a(x + y) = ax + ay$
- Distributive Property 2 - $(a + b)x = ax + bx$

Examples of Vector Spaces

- Real Numbers (R): closed under + & scalor multiplication
- Euclidean Space (R^n): e.g in R^3, vectors look like (x, y, z)
- Polynomials: Set of polynomials with real coefficients
- Matrices: All $m \times n$ matrices with real entries

Date: __/__/20 Day: _____

Solved Example

Q: Is the set of all 2 × 2 real matrices a vector space?

Answer: Yes,

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}, B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

- Closed under Addition - A + B is another 2×2 matrices.
- Closed under Scalar Multiplication - cA is also 2×2
- Addition is Commutative - A + B = B + A
- Addition is Associative - (A+B) + C = A + (B+C)
- Addictive Identify - Zero matrix exists
- Addictive Inverse - For each A, -A exists
- Multiplicative Identify - 1 · A = A
- Associative Scalar Multiplication - c(dA) = (cd) A
- Distributive Property 1 - c(A+B) = cA + cB
- Distributive Property 2 - (c+d)A = cA + dA

So, the set of all 2 × 2 real matrices is a vector space.

Dimension of a Vector Space

- The numbers of vectors in basis = dimension
- Example: $\dim(R^n) = n$
- Polynomials of degree ≤ 2 → dimension = 3

Basis of a vector Space

A set $B = \{v_1, v_2, \ldots, v_k\}$ is a basis if:

- Linearly independent
- Spans V (all vectors in V can be written using them)

Date: __/__/20__ Day: _____

Vector Addition and Scalar Multiplication

- Vector Addition: Add components

$$(v_1, v_2, v_3) + (w_1, w_2, w_3) = (v_1 + w_1, v_2 + w_2, v_3 + w_3)$$

- Scalar Multiplication: Multiply each component

$$k(v_1, v_2, v_3) = k(kv_1, kv_2, kv_3)$$

linear Combinations and Span

Any vector can be written as:

$$k_1v_1 + k_2v_2 + \cdots + k_rv_r$$

The span of $\{v_1, v_2, \ldots, v_r\}$ is all possible linear combinations of them

Properties of Vector Spaces

- Closed under addition and scalar multiplication
- Addition is associated: and commutative
- Zero vector exists
- Every vector has an additive inverse.
- Scalar multiplication follows distributive roles
- 1 acts as the multiplicative identify.

Subspaces

A subset $W \subseteq V$ is a subspace if:

- Contains the zero vector
- Closed under addition
- Closed under scalar Multiplication

Applications of Vector Spaces

- Data representation (images, text, numbers)
- Machine Learning (PCA, SVD, dimensionality reduction)
- Features vectors in AI/ML models.
- Algorithms (dot products, norms, similarity search)
- Search engines (document similarity)

Date: __/__/20 Day: _____

Difference: Vector Space Vs Euclidean Space

Vector Space	Euclidean Space
Abstract, algebraic	Geometric, spatial
Defined by addition and scalar multiplication	Defined by distances and coordinates
Focuses on algebraic structure	Focuses on geometry
Used in Linear Algebra, ML etc	Used in geometry, physics, engineering

Practice Problems

- For $\alpha = 3$, $\beta = 4$, $u = (1, 2)$, $v = (3, 4)$, verify:
  - $\alpha(u + v) = \alpha u + \alpha v$
  - $(\alpha + \beta)u = \alpha u + \beta u$
- Let $V = \{(x, y) \in R^2 \mid x \geq 0, y \geq 0\}$ it closed under addition?
- Let $V = (3, -4) \in R^2$. Find addictive inverse of v & verify $v + (-v) = 0$
- Is the set of all 2 x 2 real matrices a vector space?

Linear Mapping

linear mapping (or linear transformation) is a function that changes input values into output values using a linear rule.

It is widely used in data preprocessing, linear regression, machine learning, and mathematics

General Form:

$$y = Wx + b$$

$x \rightarrow$ input vector

$W \rightarrow$ weight matrix

$b \rightarrow$ bias vector

$y \rightarrow$ output vector

Date: __/__/20__ Day: _____

For a function $f: V \to W$ (where V, W are vector spaces), fi is linear if

- Additivity: $f(u+v) = f(u) + f(v)$
- Homogeneity: $f(w) = cf(u)$

1- Zero and Identify Transformation

- Zero Transformation: $T(v) = 0$ for all $v \in V$
- Identify Transformation: $T(v) = v$ for all $v \in V$

Properties of linear Transformation

If $T: V \to W$, then:

- $T(0) = 0$
- $T(-v) = -T(v)$
- $T(u-v) = T(u) - T(v)$
- If $v = c_1v_1 + c_2v_2 + \cdots c_nv_n$ then $T(v) = c_1T(v_1) + c_2T(v_2) + \cdots + c_nT(v_n)$

Linear Transformation of Matrix

For a matrix $A(m \times n)$

$$T(v) = Av$$

- zero matrix → zero transformation
- Identity matrix → identify transformation

Example:

let $L: R^2 \to R^3$,

$$L([v_1, v_2]) = [v_2, v_1 - v_2, v_1 + v_2]$$

- check homogeneity: $L(cv) = cL(v)$
- check addictivity: $L(v+w) = L(v) + L(w)$

Both true, L is a linear transformation

Non-Linear examples:

trigonometric or polynomial transformation.

Date: __/__/20 Day: _____

Kernal and Range Space

- Kernal (Null space): set of all $v \in V$ such that $T(v) = 0$
- Range (Image): Set of all outputs $T(v)$
- Nullity = dimension of Kernal
- Rank = dimension of range
- Rule: nullity + rank = dim(V)

linear Transformation as Rotation Matrix

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

This rotate any vector in R^2 by angle θ (anti-clockwise)

Linear Transformation as Projection Matrix:

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

If $v = (x, y, z)$, then $T(v) = (x, y, 0)$

This is projection of v onto the xy-Plane

Advantages of Linear Mapping

- Sample: Easy to understand
- Fast: Works well on large data
- Clear: Easy to interpret results
- Versatile: Used in regression, classification, clustering

Limitations of Linear Mapping

- Limited Power: Only models linear relationships
- Outliers: Sensitive to extreme Values
- Weak Feature capture: Cannot detect complex feature interactions.