MET CS688

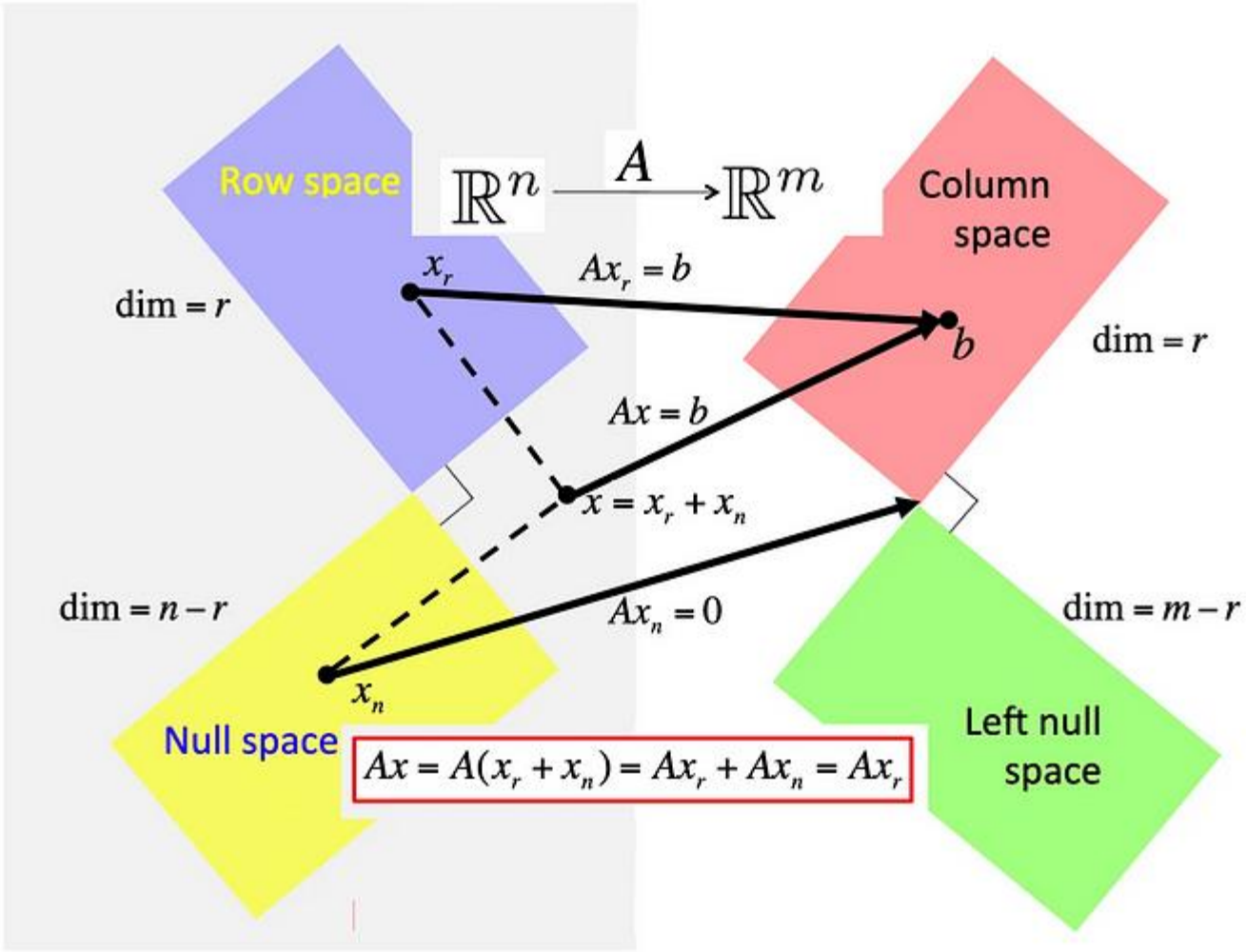
WEB ANALYTICS AND MINING

ZLATKO VASILKOSKI

MATH FOR ML

The Fundamental Mathematics of Machine Learning

A Deep Dive into Vector Norms, Linear Algebra, Calculus



Importance of Math in ML

Understanding the underlying mathematical principles behind algorithms allows you to grasp how models work, why they make specific predictions, and how to improve their performance.

- Two of the most critical areas of mathematics for machine learning are **linear algebra** and **calculus**.

- Linear algebra handles large datasets with operations like matrix multiplication and transformations and is fundamental in building and optimizing machine learning models.
    - The distance between vectors allows us to normalize our data or add regularization terms to loss functions or as part of transformation through a layer of a deep neural network.

- Calculus is essential for understanding the changes and optimizations within these models.
    - For example, computing gradients are necessary for training algorithms (e.g., gradient descent).

Importance of Math in ML

1. **Model Understanding and Development**: Math lets you comprehend how models work at a fundamental level, enabling you to develop or improve new models.

2. **Algorithm Optimization**: Optimization techniques, grounded in calculus, are crucial for minimizing error and enhancing model accuracy.

3. **Data Manipulation**: Linear algebra provides the means to handle and manipulate large datasets efficiently, which is fundamental in preprocessing data and training models.

4. **Performance Improvement**: Math concepts like regularization help prevent overfitting, thus enhancing the model's generalization to new data.

5. **Problem-Solving**: A solid mathematical foundation equips you with analytical skills to systematically approach and solve complex problems.

Linear Algebra in ML Algorithms

Vectors and Matrices: ML algorithms often use vectors and matrices to represent data. For instance, the entire dataset can be represented as a matrix, with each row being described as a vector (i.e., a sample in a dataset). If $\boldsymbol{X}$ is the data matrix, each row $\boldsymbol{x}_i$ represents a data point.

Matrix Operations: Matrix multiplication transforms data, calculates distances, and performs various linear transformations. For example, in a neural network, the input data $\boldsymbol{X}$ is multiplied by a weight matrix $\boldsymbol{W}$, producing $\boldsymbol{Z} = \boldsymbol{XW}$.

Eigenvalues and Eigenvectors: These are used in dimensionality reduction techniques, e.g., Principal Component Analysis, where the data's covariance matrix $\boldsymbol{C}$ is decomposed into its eigenvalues and eigenvectors to transform to a new coordinate system where the data variances rank the axes.

Singular Value Decomposition (SVD): SVD is used in recommendation systems and for solving linear systems. For this, we decompose a matrix $\boldsymbol{A}$ into three matrices:

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T,$$

where $\boldsymbol{U}$ and $\boldsymbol{V}$ are orthogonal matrices, and $\boldsymbol{\Sigma}$ is a diagonal matrix.

Calculus in ML Algorithms

Derivatives and Gradients: Derivatives measure the rate of change of a function (i.e., the slope at any given point). In ML, gradients (vectors of partial derivatives) minimize the loss function. For example, in gradient descent, we update the parameter θ as follows:

$$\theta = \theta - \eta \frac{\partial J}{\partial \theta},$$

where ***J*** is the loss function, and η is the learning rate.

Chain Rule: This is used for backpropagation to calculate the gradient of the loss function for each weight in a neural network. If a function *f* is composed of two functions, *g* and *h*, such that *f(x) = g(h(x))*, then the derivative of *f* is as follows:

$$\frac{df}{dx} = \frac{dg}{dh} \cdot \frac{dh}{dx}$$

Optimization Techniques: Calculus-based techniques (e.g., gradient descent) are essential for training models. These involve computing gradients to update model parameters iteratively to reduce the loss. For example, the update rule in gradient descent for parameter θ is

$$\theta = \theta - \eta \nabla_\theta J(\theta).$$

Vector & Matrices

Vectors and Matrices Math Overview

$\vec{a}$ - **Vectors** with an arrow over their variable name. Vector is $[m \times 1]$ matrix

Matrices are represented by an uppercase letter.

A is $[m \times n]$ matrix, m-**rows**, n-**columns**.

Matrix multiplication $[m \times p] * [p \times n = [m \times n]$

Non commutative - order maters $A * B \neq B * A$

I **Identity matrix** 1's on the diagonal

Inverse matrix (square A only) $A * A^{-1} = I$, similar to 1/3 being inverse to 3

Singular if A does not have inverse (too close to zero matrix)

Transpose Flipped along the diagonal (from $[m \times n]$ to $[n \times m]$)

Transpose of a matrix: $A = \begin{pmatrix} 1 & 2 \\ 2 & 3 \end{pmatrix}$ is $A^T = \begin{pmatrix} 3 & 2 \\ 2 & 1 \end{pmatrix}$

Magnitude of a vector:

$$\|a\| = \sqrt{a_1^2 + a_2^2 + \cdots + a_n^2}$$

meaning: length of a vector.

Unit Vector: $\vec{n} = \frac{\vec{a}}{\|\vec{a}\|}$

meaning: Keeps only vector's directionality. Many times just the vector's directionality is needed.

Dot (scalar) product:

$$\vec{a} \cdot \vec{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b\_n$$

$$\vec{a} \cdot \vec{b} = \|a\| \|b\| \cos \alpha$$

meaning: A dot product of the two vectors $\vec{a} \cdot \vec{b}$ is a measure to what degree two vectors are aligned.

Cosine Similarity is a measure based on the dot product of two vectors, since the angle between the vectors is given as $\cos \alpha = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \|\vec{b}\|} = \frac{\vec{a}}{\|\vec{a}\|} \cdot \frac{\vec{b}}{\|\vec{b}\|}$. It compares alignment of two vectors. For example:

Perpendicular $\cos \alpha = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \|\vec{b}\|} = \cos 90° = 0$

Collinear $\cos \alpha = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \|\vec{b}\|} = \cos 0° = 1$ (exactly opposite $\cos \alpha = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \|\vec{b}\|} = \cos 180° = -1$)

Vector Norms

Norm f satisfies the following properties:

- is *non-negative*: $f(x) \geq 0$
- is *definite*: $f(x) = 0$ implies that $x = 0$.
- is *homogeneous*: $f(tx) = |t|f(x)$
- satisfies the triangle in equality: $f(x + y) \leq f(x) + f(y)$

A norm $||x||$ can be considered a measure of the length of a vector x.

The distance between two vectors $(x, y) \in \mathbb{R}^n$ can be measured as the following norm $||x - y||$.

Euclidean norm:

$$\|x\|_2 = \sqrt{x^\top x} = \sqrt{\sum_{i=1}^{n} x_i^2} = \sqrt{x_1^2 + x_2^2 + \ldots + x_n^2}.$$

Another norm (*Chebyshev*):

$$\|x\|_\infty = \max\{|x_1|, |x_2|, \ldots, |x_k|\}.$$

Linear Algebra

Vectors - A vector is an array of numbers that encapsulates magnitude and direction.

- Vectors can represent a dataset's features, with each element corresponding to a specific feature.

Matrices - A matrix is a table of numbers.

- It can be seen as a set of vectors, where each row of the matrix is a vector.
- Matrices can represent datasets (rows are data points, columns are features)
- Transformations, and much more.

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

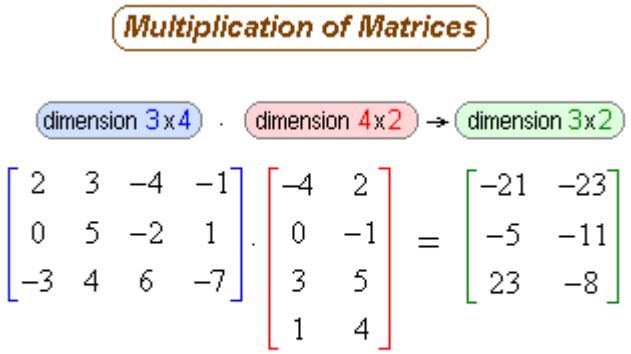
Matrix Operations

- Addition, Multiplication (a dot product of a row and a column)
- Transpose ($\boldsymbol{A^T}$): Flipping a matrix over its diagonal (rows and columns are swapped).
- Eigenvalues ($\boldsymbol{\lambda}$) and Eigenvectors ($\boldsymbol{v}$) of matrix A satisfy: $\boldsymbol{A\,v = \lambda\,v}$
  - Fundamental to understanding linear transformations.
  - The equation transforms $\boldsymbol{v}$ by $\boldsymbol{A}$, yielding a scaled version of vector $\boldsymbol{v}$.

$$\mathbf{A}^T = \begin{bmatrix} a_{11} & a_{21} & \cdots & a_{m1} \\ a_{12} & a_{22} & \cdots & a_{m2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1n} & a_{2n} & \cdots & a_{mn} \end{bmatrix}$$

Basic Matrix Math

- Column and row vectors
- Matrix multiplication



- Transpose

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

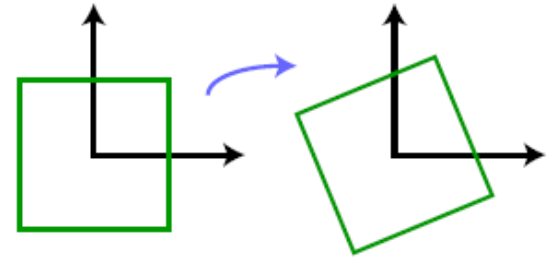
- Determinant

$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix}$$

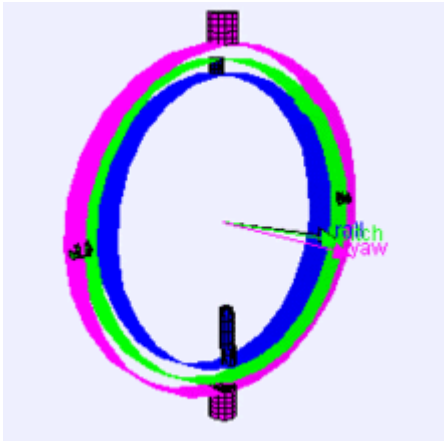
Transformations & Matrices

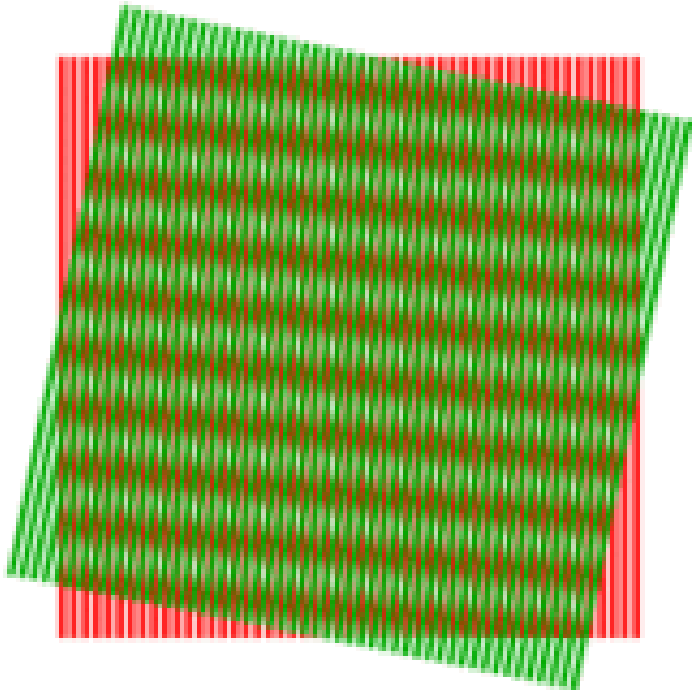
Rotation about the origin is given by a matrix R.

$$p_1 = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} p_0 = Rp_0$$

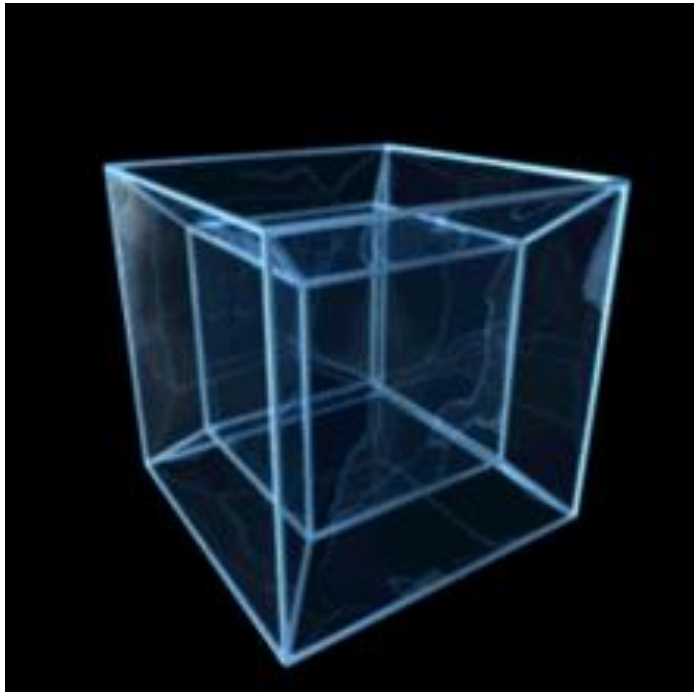


Affine





Moiré pattern by rotation



Four-dimensional rotation

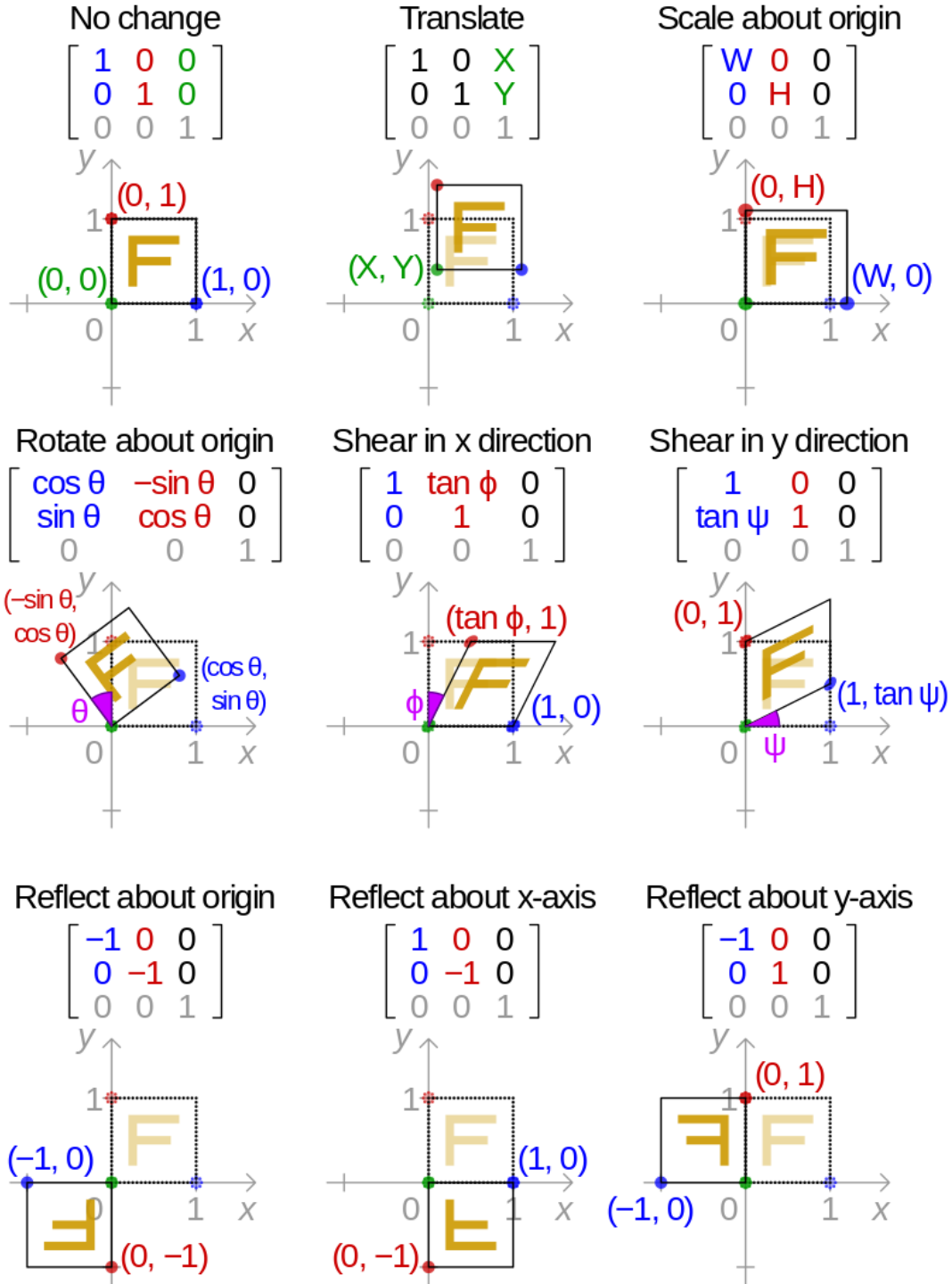
Types of Transformations

Types of Transformations:

1. Translations
2. Scaling
3. Rotations
4. Reflections
5. Shear

Various 2D **affine** (preserves parallelism) transformation matrices on a unit square.

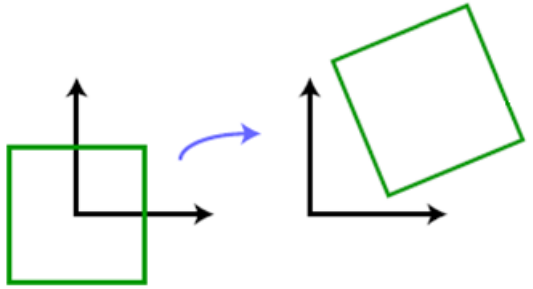
Note: that the reflection matrices are special cases of the scaling matrix.



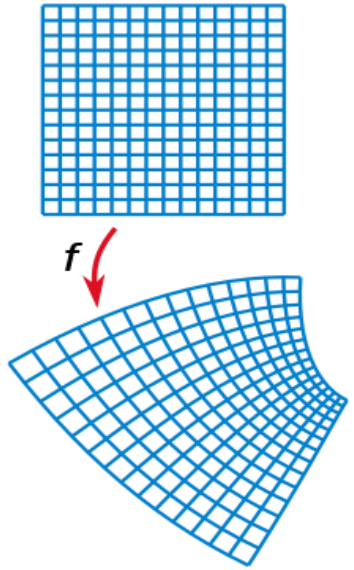
Transformations & Matrices

Types of Transformations:

1. Translations
2. Scaling
3. Rotations
4. Reflections
5. Shear



1) Rigid body transformation



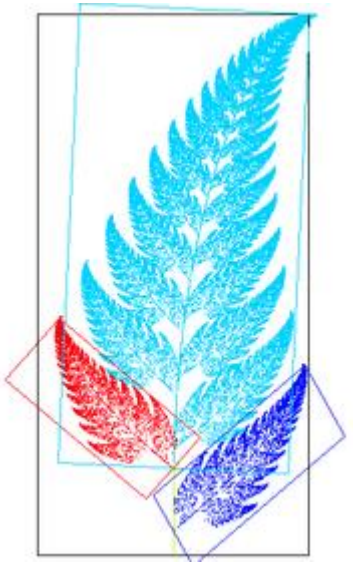
2) Conformal transformation

There are three basic classes of transformations:

1) Rigid body - Preserves **distance** and **angles**.
   Examples: translation and rotation.

2) Conformal - Preserves **angles**.
   Examples: translation, rotation, and uniform scaling.

3) Affine - Preserves **parallelism**. Lines remain lines.
   Examples: translation, rotation, scaling, shear, and reflection.

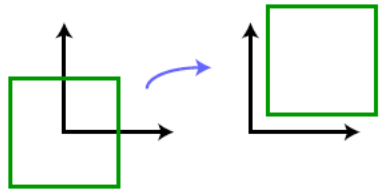


3) Affine transformation

2D Matrix Transformations

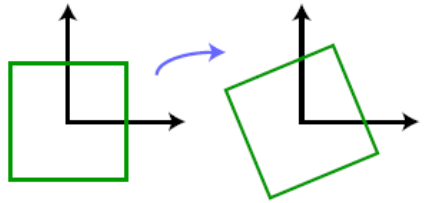
Translation of a point by a vector t

$$p_1 = p_0 + \vec{t}$$



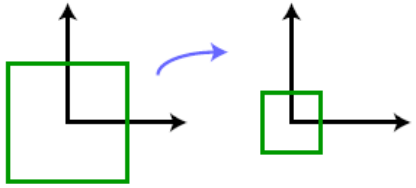
Rotation counterclockwise by angle θ

$$p_1 = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} p_0$$



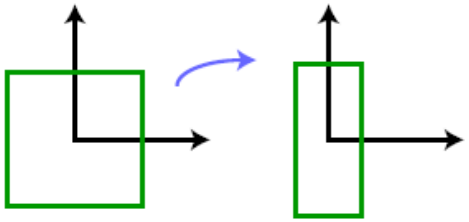
Uniform scaling by scalar a

$$p_1 = \begin{bmatrix} a & 0 \\ 0 & a \end{bmatrix} p_0$$



Non-uniform scaling

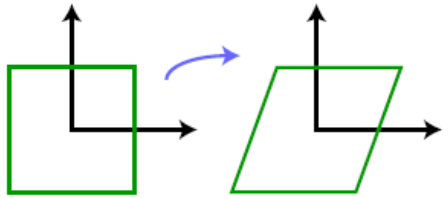
$$p_1 = \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix} p_0$$



2D Matrix Transformations

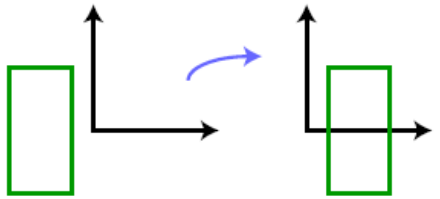
Shear by scalar h

$$p_1 = \begin{bmatrix} 1 & h \\ 0 & 1 \end{bmatrix} p_0$$



Reflection about the y-axis

$$p_1 = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} p_0$$



Singular Value Decomposition (SVD)

In linear algebra, the singular-value decomposition (SVD) is a factorization of a $m \times n$ matrix into three components, 2 **unitary matrices** (U and V) and one **diagonal** (Σ).

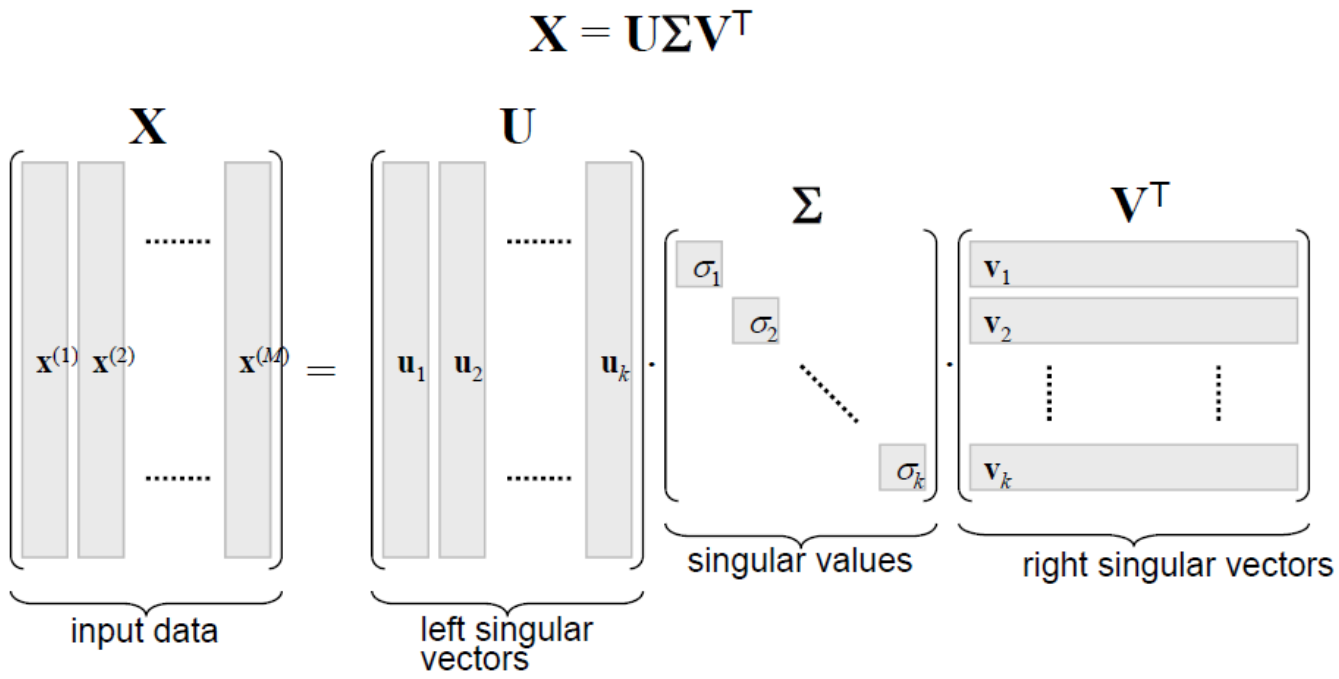
$$X = U\Sigma V^T$$

- Unitary matrix U of size $m \times m$,
- Diagonal matrix Σ of size $m \times n$ and
- Unitary matrix V of size $n \times n$.

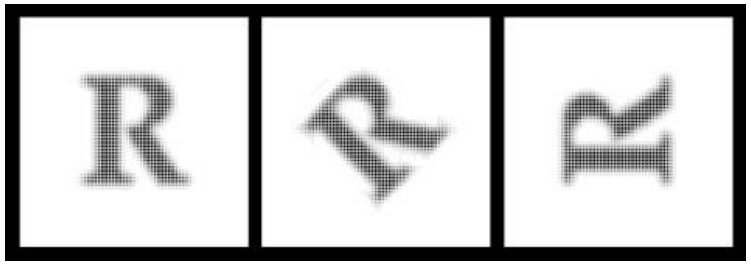
Matrix A is unitary if

$$A^*A = AA^* = I$$

where A^* is conjugate transpose of matrix A.



One way to think of **unitary matrices** is they rotate objects but don't change the length or the angles of an object, such as the letter R.



Singular Value Decomposition (SVD)

The geometric interpretation of SVD of matrix X

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

Σ stretches and U and V only rotate matrix (object) X. This can be seen by representing the rotation as

$$p_1 = Rp_0 = R\Sigma I = U\Sigma V^T$$

$$R = U\Sigma \text{ and } V = I$$

The columns of U are called the **left-singular vectors** (orthonormal eigenvectors of $XX^* = U\Sigma^2$)

$$XX^* = U\Sigma V^* V\Sigma U^* = U\Sigma I\Sigma U^* = U\Sigma^2 U^* = U\Sigma^2$$

The columns of V are called **right-singular vectors** (orthonormal eigenvectors of $X^*X = V\Sigma^2$)

$$X^*X = V\Sigma U^* U\Sigma V^* = V\Sigma I\Sigma V^* = V\Sigma^2 V^* = V\Sigma^2$$

The non-zero singular values of X are found on the diagonal entries of Σ.

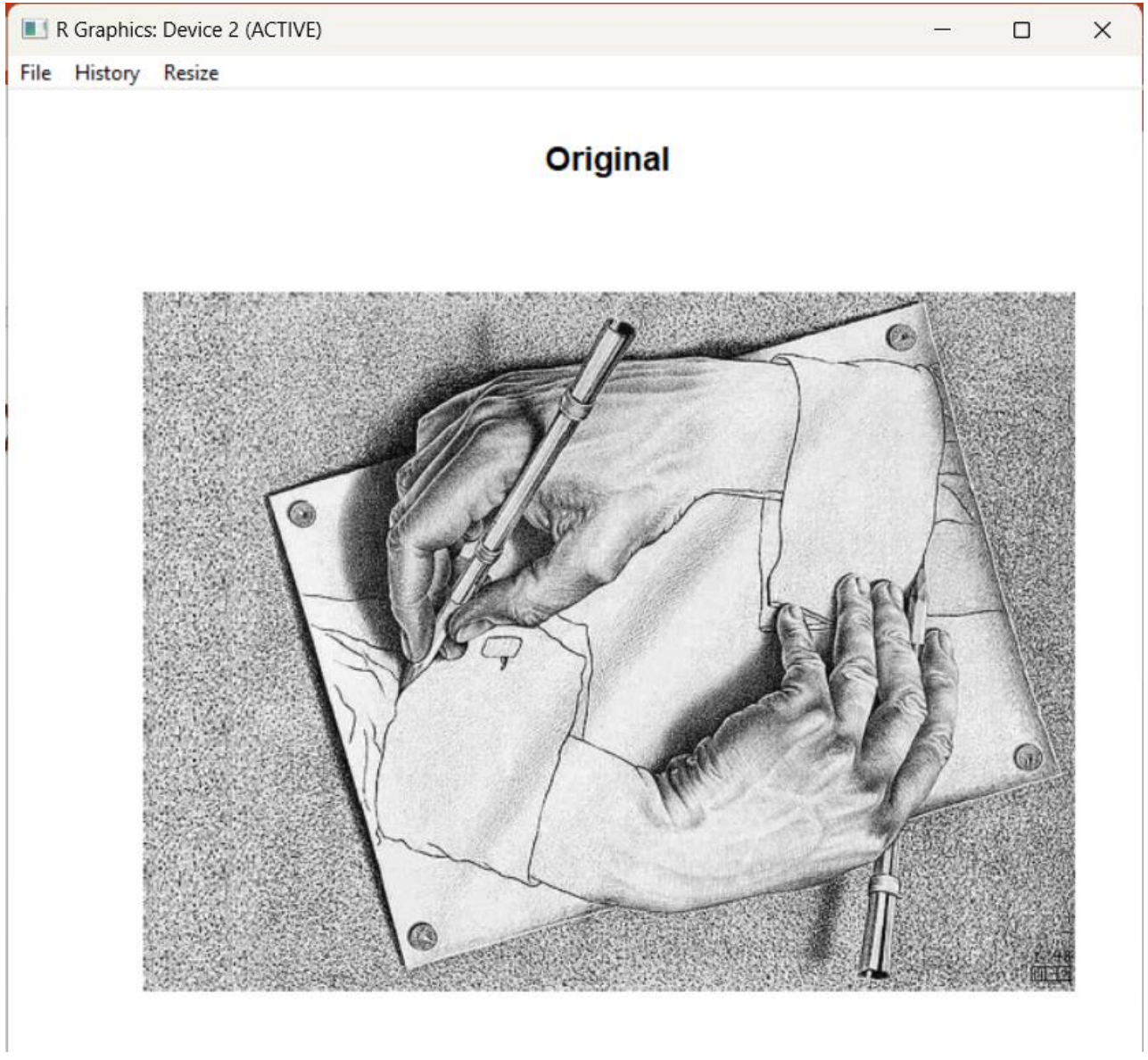
Applications of SVD

- Applications that employ the SVD include
  - Computing the pseudoinverse,
  - Least squares fitting of data,
  - Multivariable control,
  - Matrix approximation, and
  - Determining the rank,
  - Range and null space of a matrix and
  - File/image compression.

Using SVD for Image Compression

- Import image using package “Magick”.
- Convert magick object to hex bitmap
- Convert bitmap into matrix of integers to apply SVD

```r
rm(list=ls()); cat("\014") # clear all
library(magick)

# ==== Read the image ====
img <- image_read('data/ESC_Hands.png') # Read a PNG Image
plot(img); title("Original")

im_a <- image_data(img, 'gray') #  Convert magick object to hex bitmap
im <- as.integer(im_a[1,,]) # Convert bitmap into integers
im <- matrix(im, dim(im_a[1,,])) # Create a Matrix
```



Using SVD for Image Compression

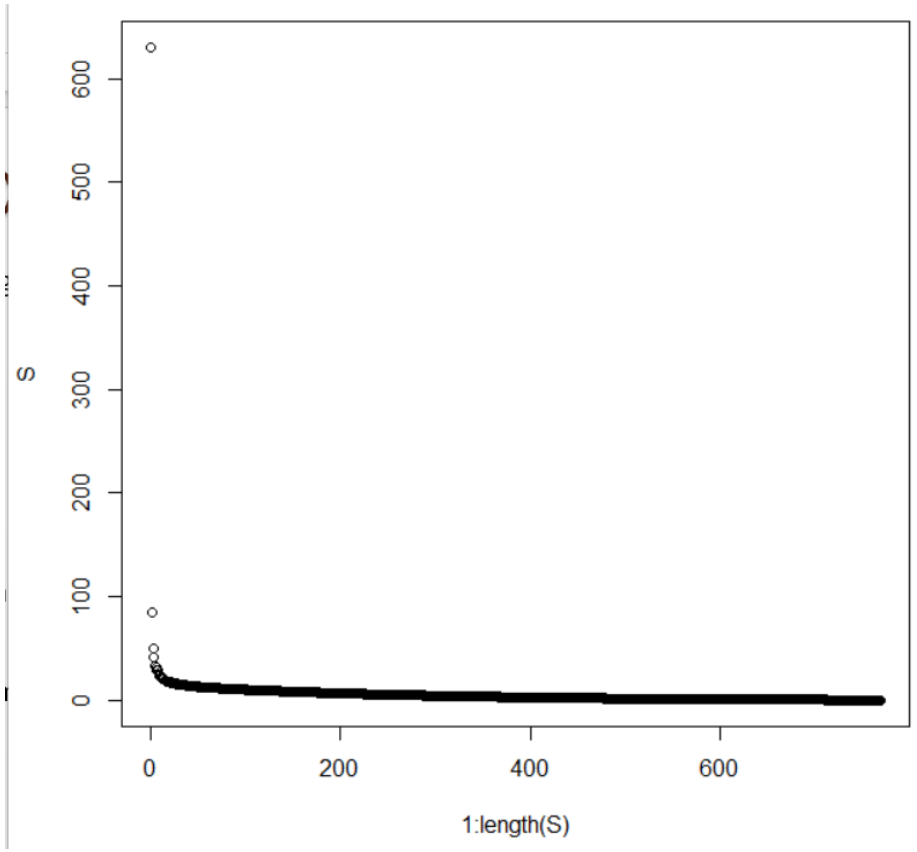
Get the Single Value decomposition of the Image

- Note that SVD contains three components i.e. U, S and V^T.

```
# ==== SVD ====
# Get the Single Value decomposition of the Image
svd <-svd(im/255)

# Note that SVD contains three components i.e. U, D and V^T.
str(svd)

U = svd$v # U Left Singular Matrix of size 768 x 768
S = svd$d  # Σ Singular Values Vector of length 768
V = svd$u # V Right Singular Matrix of size 1024 x 768

# Check
im_svd1 = t(U %*% diag(S) %*% t(V)) # Gives the same image matrix
```

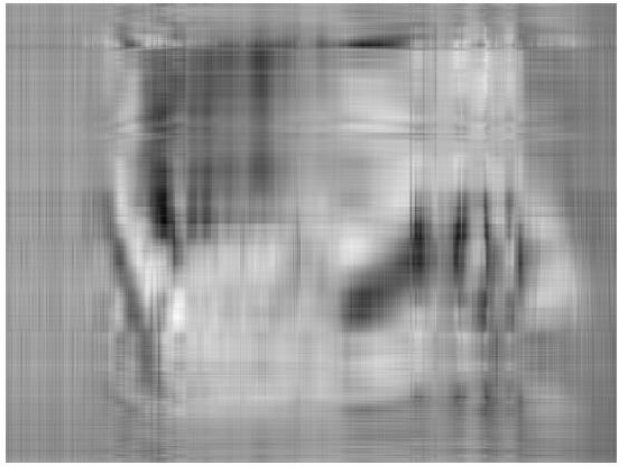
Plotting the variance in Σ

- After 10 vectors or so the variance in S is very low and almost steady.

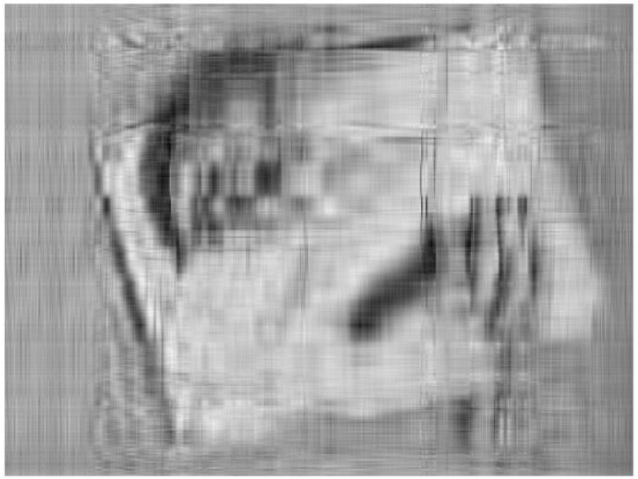


Using SVD for Image Compression

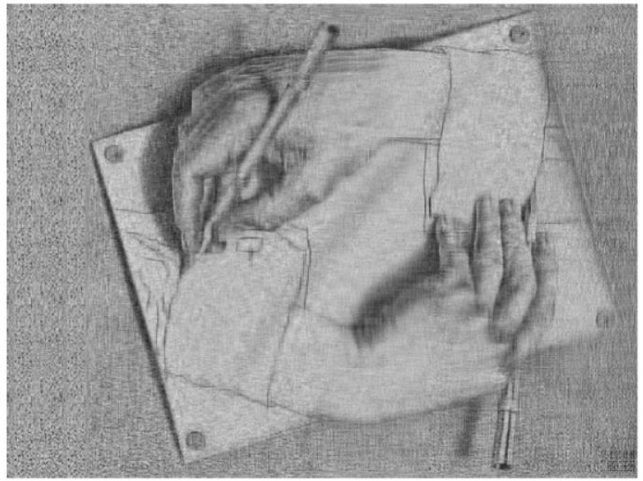
Approximated Image using 5 singular Vectors



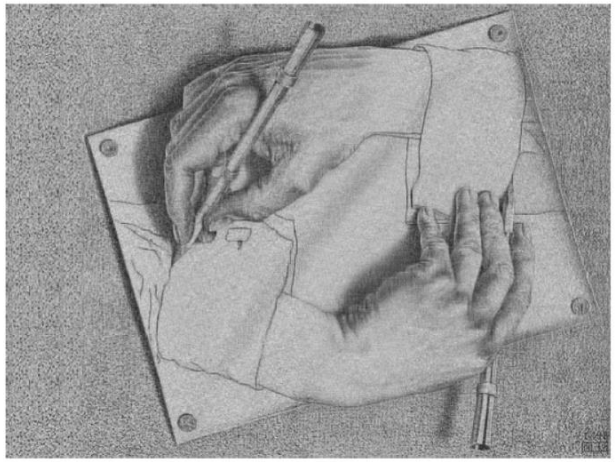
Approximated Image using 10 singular Vectors



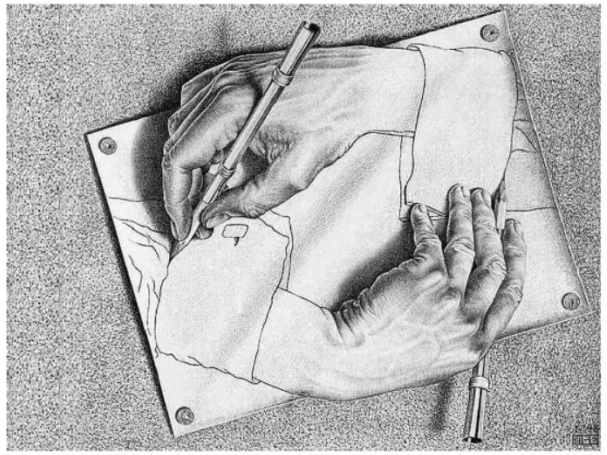
Approximated Image using 50 singular Vectors



Approximated Image using 100 singular Vectors

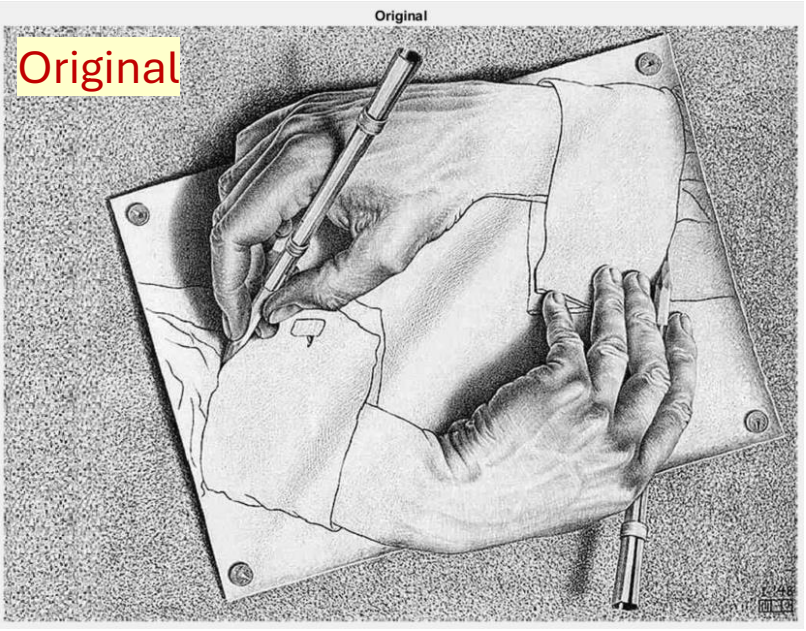


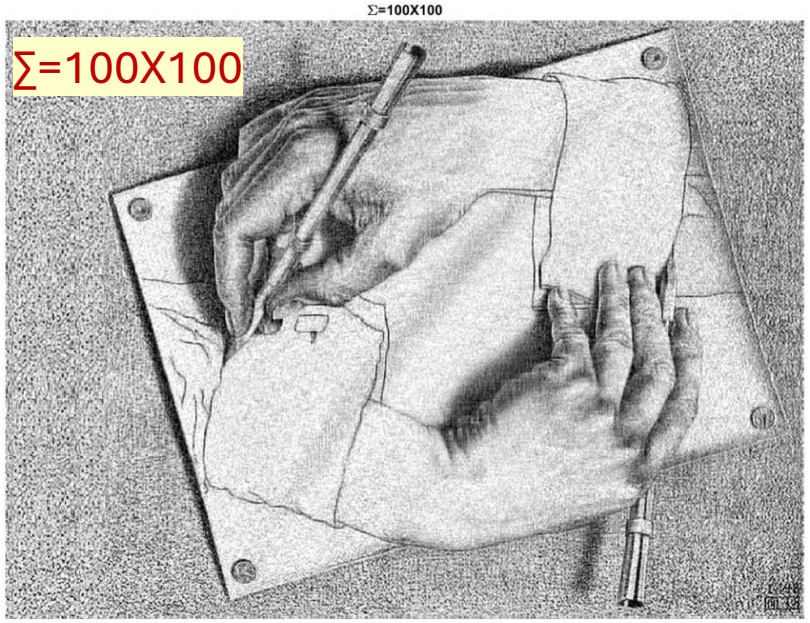
Original

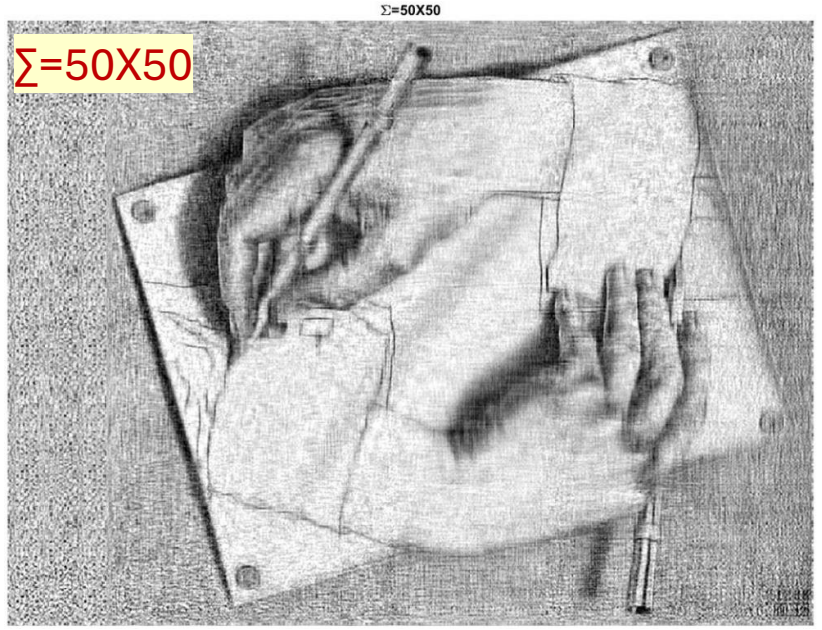


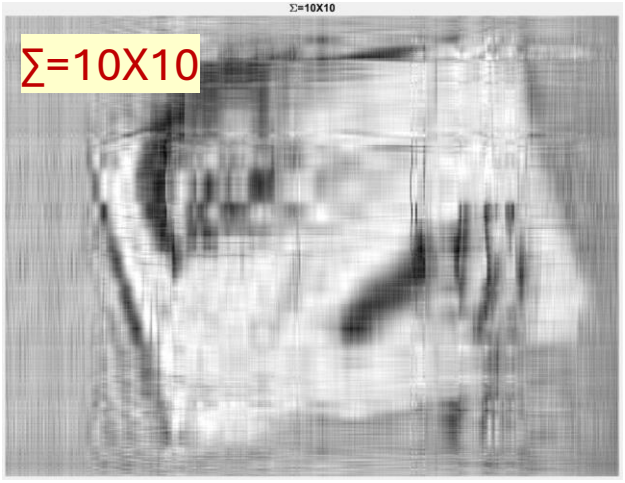
Let’s reduce the number of singular vectors to only the first: 5, 10, 50, 100.

Singular Value Decomposition (SVD)



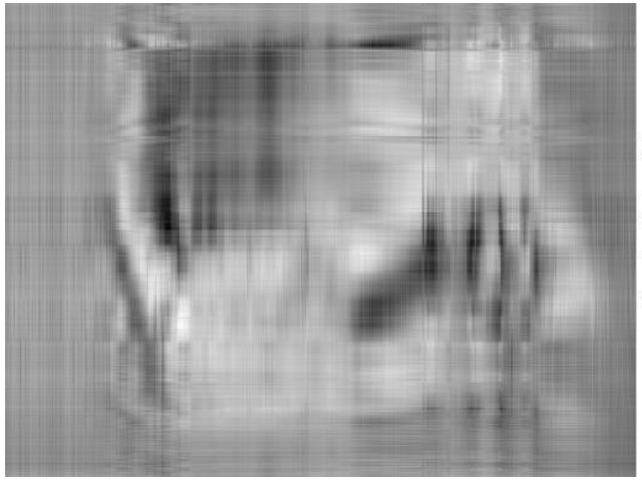


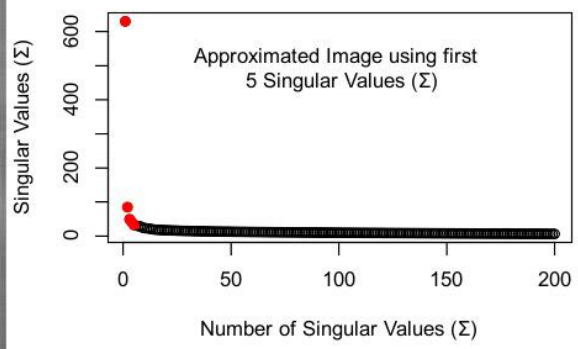


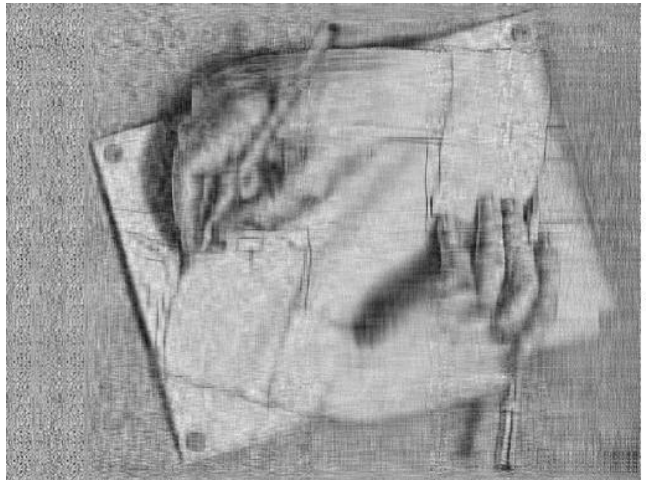


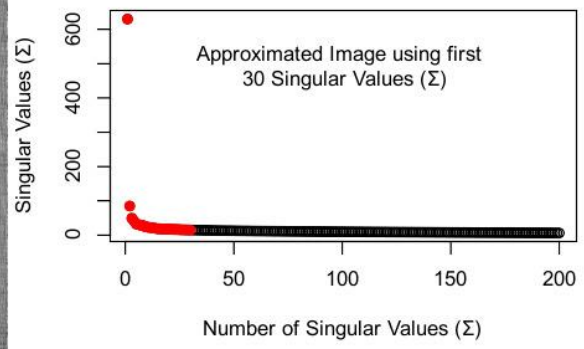
∑=10X10

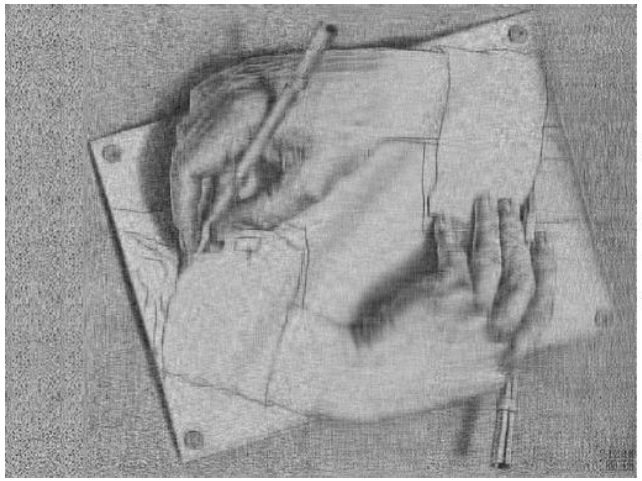
630.1644	0	0	0	0	0	0	0	0	0	0	0
0	84.7838	0	0	0	0	0	0	0	0	0	0
0	0	49.1495	0	0	0	0	0	0	0	0	0
0	0	0	41.1680	0	0	0	0	0	0	0	0
0	0	0	0	32.7525	0	0	0	0	0	0	0
0	0	0	0	0	30.8359	0	0	0	0	0	0
0	0	0	0	0	0	29.1636	0	0	0	0	0
0	0	0	0	0	0	0	28.1809	0	0	0	0
0	0	0	0	0	0	0	0	24.8132	0	0	0
0	0	0	0	0	0	0	0	0	23.7083	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

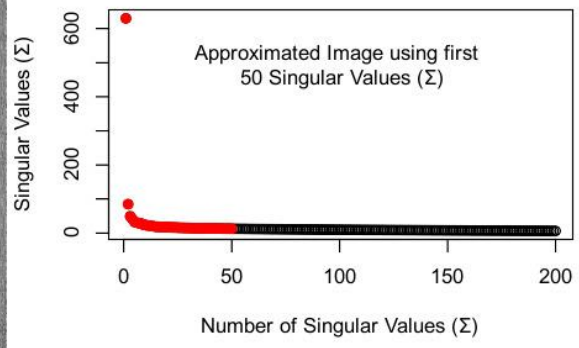


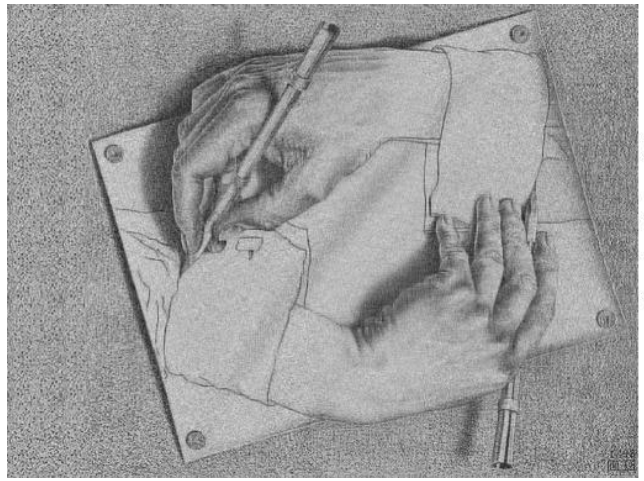

Approximated Image using first
5 Singular Values (Σ)
Singular Values (Σ)
0
200
400
600
0
50
100
150
200
Number of Singular Values (Σ)

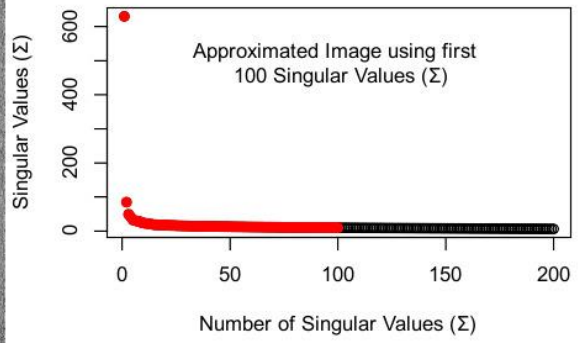



Approximated Image using first
30 Singular Values (Σ)
Singular Values (Σ)
0
200
400
600
0
50
100
150
200
Number of Singular Values (Σ)



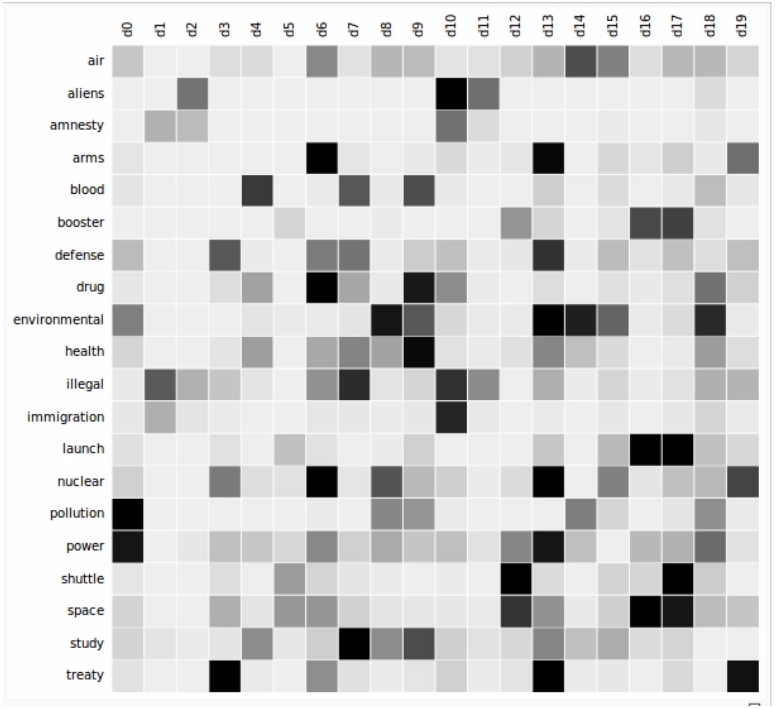

Approximated Image using first
50 Singular Values (Σ)
Singular Values (Σ)
0
200
400
600
0
50
100
150
200
Number of Singular Values (Σ)

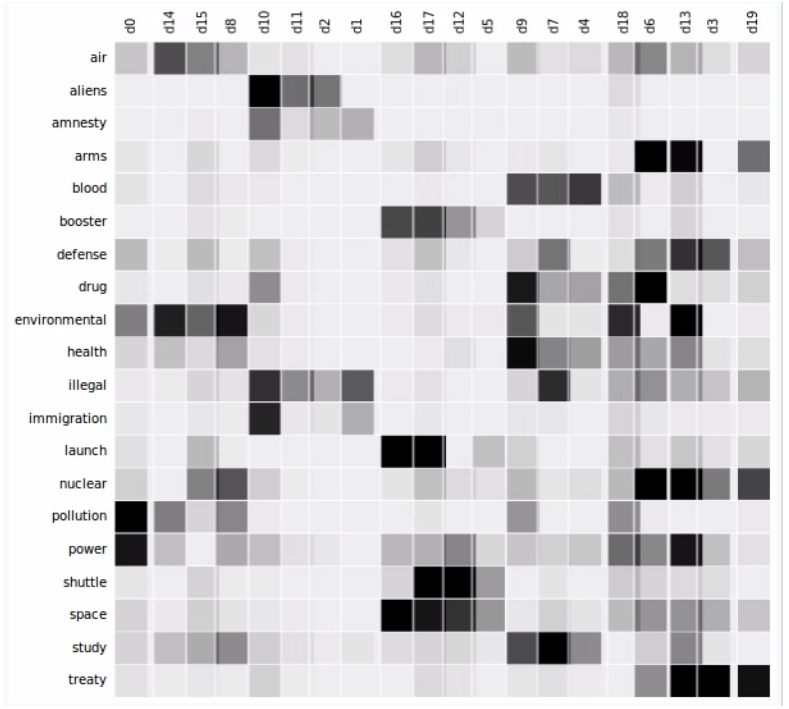


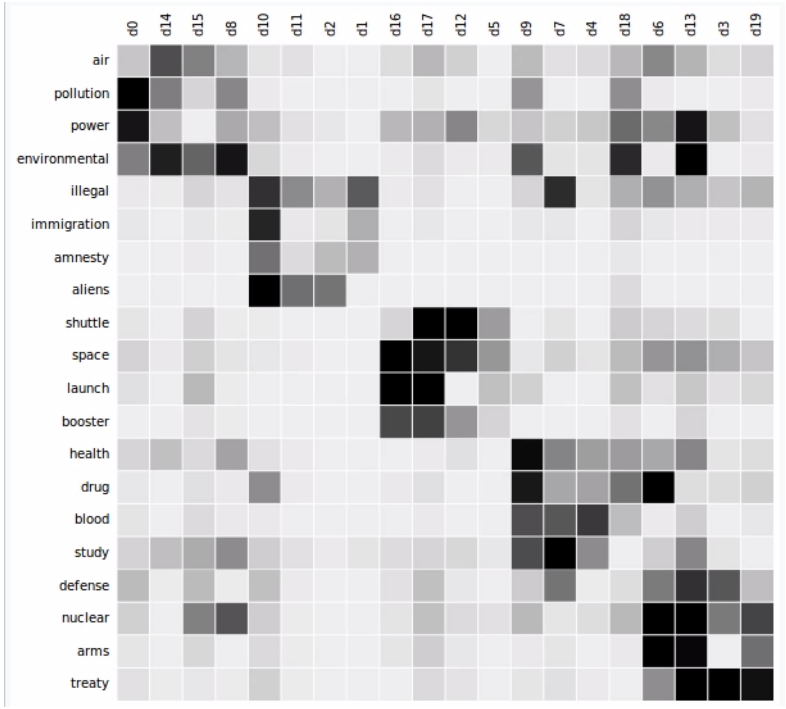

Approximated Image using first
100 Singular Values (Σ)
Singular Values (Σ)
0
200
400
600
0
50
100
150
200
Number of Singular Values (Σ)

SVD and Text Mining

- https://en.wikipedia.org/wiki/Latent_semantic_analysis







A is document term [mxn] matrix (m rows, n columns).
A^TA is term to term similarity [mxm] matrix.
AA^T is document to document similarity [nxn] matrix.

Matrix Factorization and Decomposition

Principal Component Analysis (PCA) is a dimensionality reduction technique that transforms the data into a new coordinate system where the greatest variances lie on the first coordinates (i.e., the principal components). This technique has the following steps:

1. **Standardize the data**: Subtract the mean and divide by the standard deviation for each feature.
2. **Compute the covariance matrix**: $\mathbf{C} = \frac{1}{n-1}\mathbf{X}^T\mathbf{X}$
3. **Compute the eigenvalues and eigenvectors of the covariance matrix**.
   - The directions of the axes where there is the most variance (most information) are reflected in the **eigenvectors**,
   - The amount of (i.e., magnitude) of variance is reflected in the **eigenvalues**.
4. **Sort the eigenvectors by decreasing eigenvalues** and select the top *k* eigenvectors.
   - The top *k* eigenvalues capture the most variance.
5. **Transform the data** using the selected *k* eigenvectors.
   - ***Singular Value Decomposition (SVD)*** is a matrix factorization technique that decomposes a matrix ***A*** into three matrices:

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

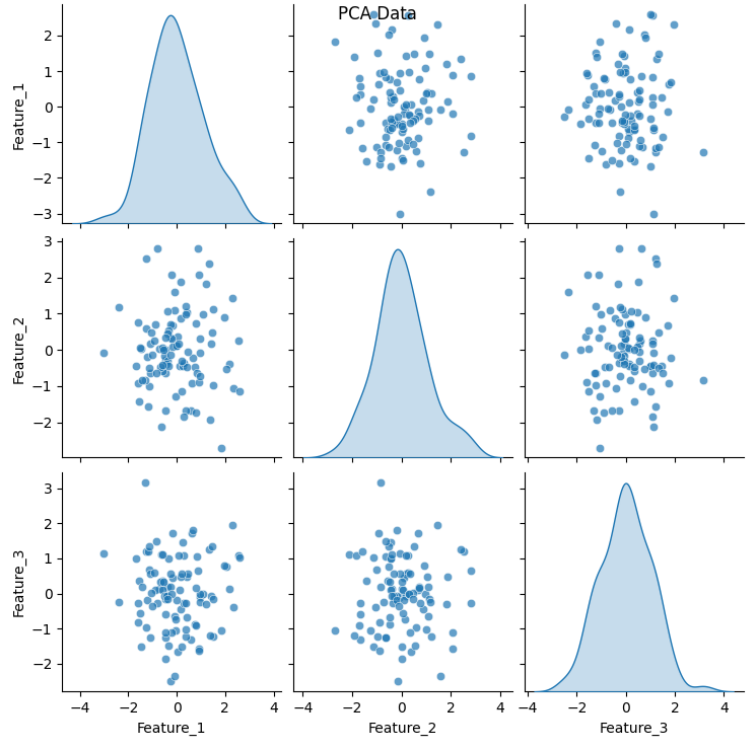
   - where ***U*** and ***V*** are orthogonal matrices, and ***Σ*** is a diagonal matrix of singular values.
   - SVD is used in recommendation systems, latent semantic analysis, and more.

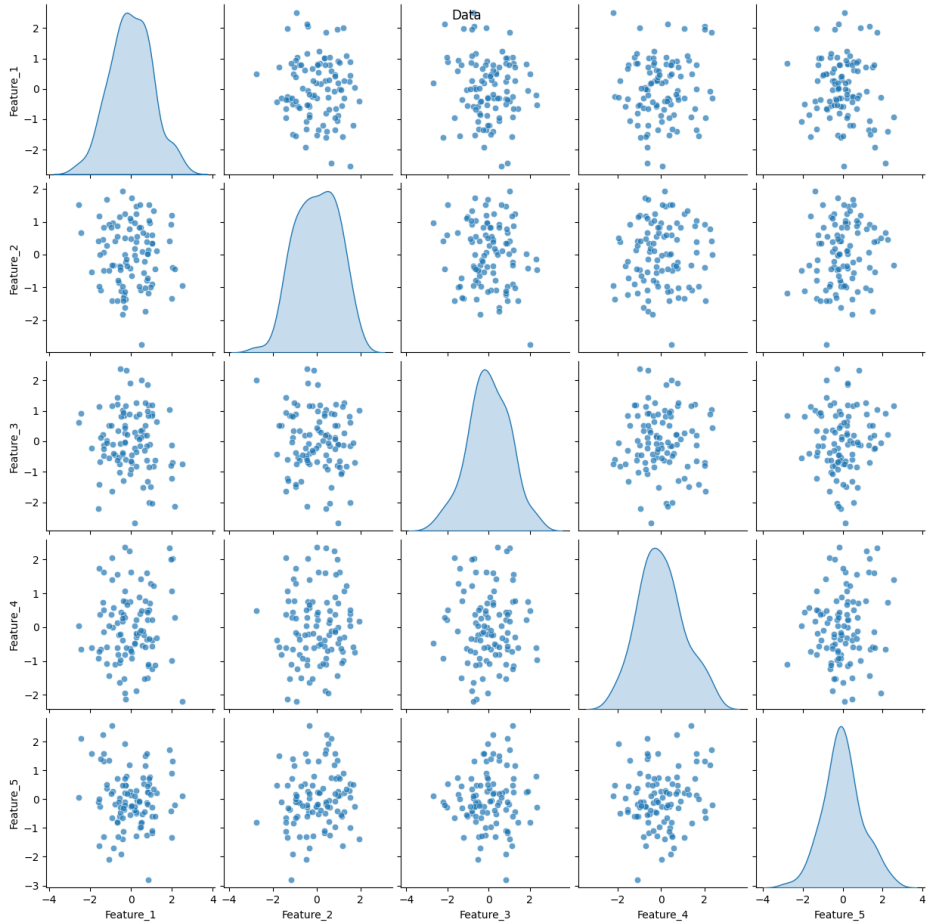
PCA makes it easier to visualize the data and reduces the computational cost.

PCA Lab

Task: Use PCA to

- Reduce the dimensionality of a dataset
- While retaining as much variance as possible.






```
Array of the variance of each dimension [0.24267417 0.23235282 0.20974076]
Variance per feature                    0         1         2
Feature_1 -0.244663  0.692295  0.310833
Feature_2  0.419437  0.119788 -0.522196
Feature_3  0.080971 -0.575549  0.621509
Feature_4  0.630407  0.394859  0.484043
Feature_5  0.600203 -0.138593 -0.100616
```

SVD & Recommendation Systems

Singular Value Decomposition (SVD) is used in recommendation systems to predict user preferences.

Task:

- Given user-item rating matrix R,
- Use SVD
- To predict user-item ratings

```
# Example user-item rating matrix
R = np.array([
    [5, 3, 0, 1],
    [4, 0, 0, 1],
    [1, 1, 0, 5],
    [1, 0, 0, 4],
    [0, 1, 5, 4],
])*1.0
```

```
[[ 5.13406479  1.90612125 -0.72165061  1.5611261 ]
 [ 3.43308995  1.28075331 -0.45629689  1.08967559]
 [ 1.54866643  1.0449763   1.78873709  3.96755551]
 [ 1.17598269  0.80359806  1.40136891  3.08786154]
 [-0.44866693  0.5443561   3.09799526  5.15263893]]
```

Calculus in ML

Plays role in understanding and optimizing machine learning models.

- **Derivatives and Integrals**

- **Partial Derivatives and Gradients**
  - **Gradients** are a vector of partial derivatives and point toward the steepest increase of the function.
  - In ML, gradients are used in optimization to minimize the loss function by updating the model parameters in the negative gradient direction.

- **Chain Rule and Backpropagation**
  - **Chain Rule** is used to compute the derivative of a composition of functions (depending on several variables).
  - **Backpropagation** is an algorithm for training neural networks. It uses the chain rule to compute the gradient of the loss function w.r.t. each weight in the network. This allows the weights to be updated to minimize the loss.

Gradient Descent

Gradient Descent is an optimization algorithm that minimizes the loss function by iteratively moving toward the steepest descent, as defined by the negative gradient.

Gradient Descent Algorithm:

1. Initialize the parameters (weights) randomly.
2. Compute the gradient of the loss function w.r.t. the parameters.
3. Update the parameters in the opposite direction of the gradient by a step size (learning rate).
4. Repeat steps 2 and 3 until convergence.

The parameter update rule for gradient descent is:

$$\theta = \theta - \eta \nabla_\theta J(\theta)$$

θ are the parameters,
η is the learning rate,
$J(\theta)$ is the loss function,
$\nabla_\theta J(\theta)$ are parameter gradients from loss function.

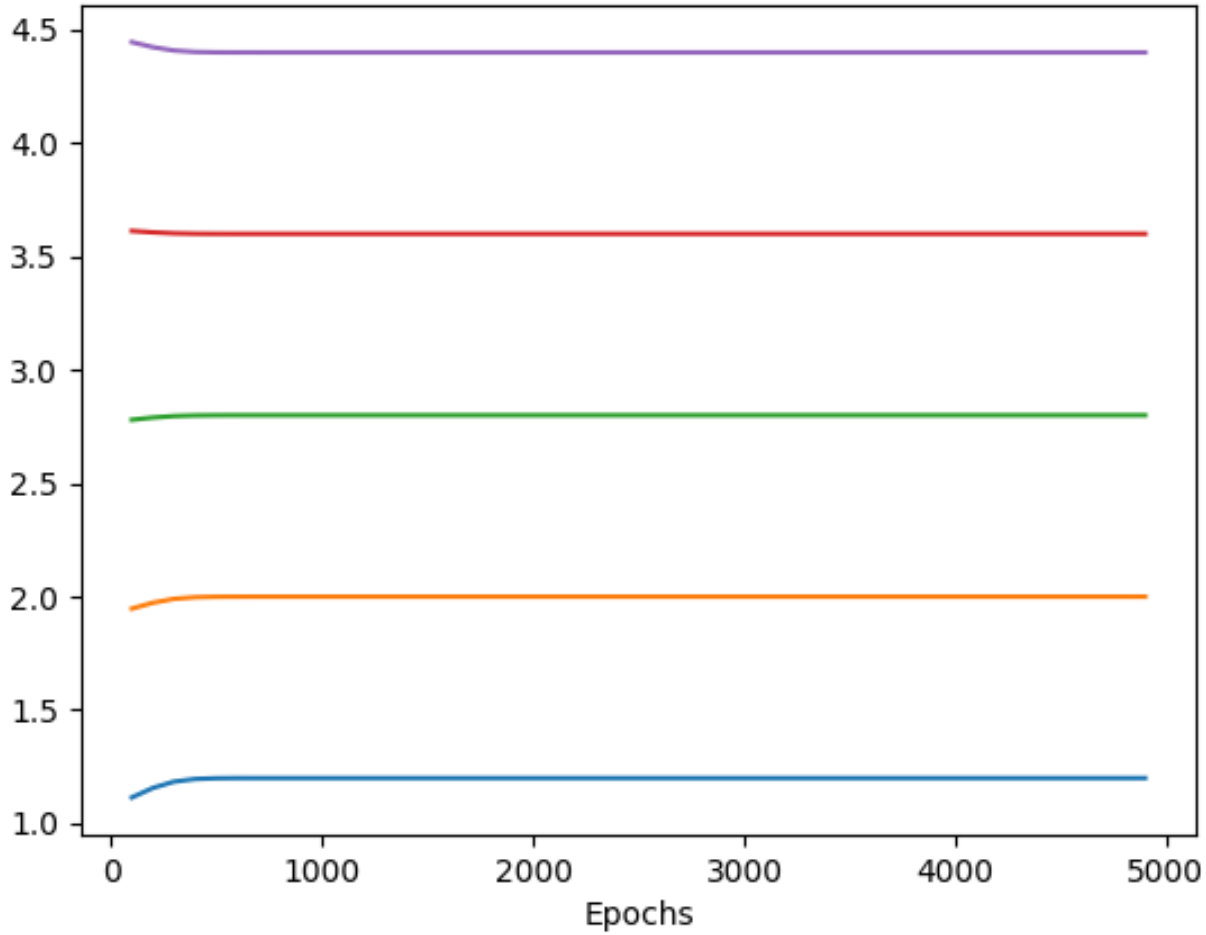
Gradient Descent - Practical Example

Task:

Optimize parameters for Linear regression

- Calculate the numerical solution y(x) = m*x+b for various number of epochs in which you calculated m and b.
- Start with 100 epochs and in increments of 100 end with 5000 epochs
- Plot the numerical solution of y(x) for all the epochs you used.

Note: the solution approaches its approximate numerical result with a y-intercept of 0.4 and a slope m of 0.8.



A

A

A

A