Linear Algebra for Machine Learning

Youssef SALMAN

Applied Mathematics
Lebanese University — Faculty of Sciences (Section V)

M1: Applied Mathematics & Statistics for Computational Sciences

October 15, 2025

Learning Objectives

- Understand linear maps, kernel/image, rank–nullity, and invertibility.
- Compute eigenvalues/eigenvectors; know when diagonalization is possible.
- Use SVD for low-rank approximation and PCA for dimensionality reduction.
- Implement these concepts in Python (NumPy/Scikit-Learn).

Outline I

Outline

Vectors I

Definition 1

A **vector** in $\mathbb{R}^d$ is an ordered list of d numbers:

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}.$$

It represents a point or data item with d features.

Norm (length):

$$\|\boldsymbol{x}\| = \sqrt{x_1^2 + x_2^2 + \cdots + x_d^2}.$$

Dot product (similarity):

$$\boldsymbol{x} \cdot \boldsymbol{y} = \sum_i x_i y_i.$$

Cosine similarity (angle-based):

$$\cos\theta = \frac{\boldsymbol{x}\cdot\boldsymbol{y}}{\|\boldsymbol{x}\|\|\boldsymbol{y}\|}.$$

Real-life: In text mining, documents are vectors of word counts; cosine similarity finds which texts are most alike (search engines, recommender systems).

Definition 2

Given vectors $v_1, v_2, \ldots, v_k \in \mathbb{R}^n$, the **span** is

$$\operatorname{span}\{v_1, v_2, \ldots, v_k\} = \{\alpha_1 v_1 + \alpha_2 v_2 + \cdots + \alpha_k v_k \mid \alpha_i \in \mathbb{R}\}.$$

It is the set of all linear combinations of these vectors.

Examples:

- In $\mathbb{R}^2$, $\operatorname{span}\{(1,0)\}$ is the x-axis.

Vectors III

- In $\mathbb{R}^2$, $\text{span}\{(1,0),(0,1)\} = \mathbb{R}^2$.
- In $\mathbb{R}^3$, $\text{span}\{(1,0,0),(0,1,0)\}$ is the xy-plane.

Geometric meaning: The span describes the smallest subspace containing the given vectors (line, plane, or the whole space).

Matrices

Definition 3

A **matrix** is a rectangular array of numbers with m rows and n columns:

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

Interpretations:

- Dataset: rows = observations, columns = features.
- Linear map: $A\boldsymbol{x}$ transforms $\boldsymbol{x} \in \mathbb{R}^n$ into $\mathbb{R}^m$.

Example: In a dataset of 100 students and 5 grades, the data form a 100×5 matrix. Multiplying by a weight vector computes each student's final score.

Rank of a Matrix

- $\text{rank}(A)$ = number of linearly independent columns (column rank).
- $\text{rank}(A)$ = number of linearly independent rows (row rank).
- $\text{rank}(A)$ = number of pivots in the row echelon form.

Properties:

- $\text{rank}(A) \leq \min(m, n)$.
- $\text{rank}(AB) \leq \min(\text{rank}(A), \text{rank}(B))$.
- A is invertible $\iff$ $\text{rank}(A) = n$ (when A is square $n \times n$).

Geometric meaning: Rank = the number of independent directions preserved by the transformation A.

Outline

Linear Maps I

Definition 4

A mapping $T : \mathbb{R}^n \to \mathbb{R}^m$ is called *linear* if it preserves **addition** and **scaling**:

$$T(\alpha \boldsymbol{x} + \beta \boldsymbol{y}) \;=\; \alpha T(\boldsymbol{x}) + \beta T(\boldsymbol{y}) \quad \text{for all } \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n,\ \alpha, \beta \in \mathbb{R}.$$

Note 1

Every linear map can be represented by a matrix $A \in \mathbb{R}^{m \times n}$, so that

$$T(\boldsymbol{x}) = A\boldsymbol{x}.$$

This makes linear maps easy to compute on a computer.

Example: A 90° rotation in the plane is linear:

$$R(x, y) = (-y,\, x), \quad R(\boldsymbol{x}) = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \boldsymbol{x}.$$

Linear Maps II

Real-life: In image processing, rotating or scaling an image are linear maps on the pixel coordinate vectors.

Kernel & Image I

Definition 5

For a linear map $T : \mathbb{R}^n \to \mathbb{R}^m$:

$$\ker(T) = \{\boldsymbol{x} \in \mathbb{R}^n \mid T(\boldsymbol{x}) = \mathbf{0}\}, \quad \mathrm{Im}(T) = \{T(\boldsymbol{x}) \mid \boldsymbol{x} \in \mathbb{R}^n\} \subseteq \mathbb{R}^m.$$

Definition 6

A linear map T (or matrix A) is **injective** if

$$T(x_1) = T(x_2) \Rightarrow x_1 = x_2,$$

that is, it never sends two different vectors to the same image.
Equivalently,

$$\ker(A) = \{0\}.$$

Kernel & Image II

Definition 7

Surjective (onto). A linear map T (or matrix A) is **surjective** if

$$\mathrm{Im}(A) = \mathbb{R}^m,$$

meaning every vector in the target space is the image of some x.

Proposition 1

- T *is* **one-to-one (injective)** $\iff \ker(T) = \{\mathbf{0}\}$.
- T *is* **onto (surjective)** $\iff \mathrm{Im}(T) = \mathbb{R}^m$.

Example 1 (Kernel). Let

$$A = \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}, \quad T(\boldsymbol{x}) = A\boldsymbol{x}.$$

Kernel & Image III

Here $\ker(T) = \text{span}\{(-2, 1)\}$, a line in $\mathbb{R}^2$. $\Rightarrow$ Many different inputs collapse to $\mathbf{0}$, hence not injective.

Example 2 (Image). Same A: every output lies on the line $\text{span}\{(1, 2)\} \subset \mathbb{R}^2$. So $\text{Im}(T)$ is 1D, not the whole $\mathbb{R}^2$ $\Rightarrow$ not onto.

Real-life analogy:

- $\ker(T)$ = information lost. (e.g. projecting a photo onto the x-axis deletes vertical detail).
- $\text{Im}(T)$ = possible outputs. (e.g. grayscale images are projections of RGB, image lives in a lower-dimensional space).

Rank–Nullity & Invertibility

Theorem 8 (Rank–Nullity)

For $A \in \mathbb{R}^{m \times n}$,

$$\operatorname{rank}(A) + \dim(\ker A) = n.$$

Proposition 2

If $A \in \mathbb{R}^{n \times n}$, *then* A *is invertible*

$$\iff \operatorname{rank}(A) = n \iff \ker(A) = \{0\} \iff \det(A) \neq 0.$$

Outline

Eigenvalues & Eigenvectors I

Definition 9

Let $A \in \mathbb{R}^{n \times n}$. A scalar $\lambda \in \mathbb{R}$ is called an **eigenvalue** of A if there exists a nonzero vector $\boldsymbol{v}$ such that

$$A\boldsymbol{v} = \lambda \boldsymbol{v}.$$

The vector $\boldsymbol{v}$ is then an **eigenvector** associated with λ.

Note 2

Eigenvalues satisfy the **characteristic equation***:*

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

Eigenvalues & Eigenvectors II

Intuition: An eigenvector is a special direction that A does not change (it may stretch or shrink, but does not rotate away). The eigenvalue tells us the factor of stretching.

Example 1 (Scaling).

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

- $(1, 0)$ is an eigenvector with eigenvalue $\lambda = 2$. - $(0, 1)$ is an eigenvector with eigenvalue $\lambda = 3$. $\Rightarrow$ A scales x-axis by 2 and y-axis by 3.

Example 2 (Rotation).

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

This matrix rotates vectors by 90°. - No real vector keeps its direction. - $\Rightarrow$ R has no real eigenvalues, but has complex eigenvalues $\lambda = \pm i$.

Example 3 (Shear).

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

- $(1, 0)$ is an eigenvector with eigenvalue 1. - Geometrically, shear keeps the x-axis fixed but shifts other directions.

Eigenvalues & Eigenvectors IV

Applications in ML/CS:

- **PCA:** eigenvectors of the covariance matrix give the directions of maximum variance.
- **Google PageRank:** the dominant eigenvector of a stochastic matrix gives webpage importance.
- **Clustering:** spectral clustering uses eigenvectors of graph Laplacians.
- **Stability analysis:** signs of eigenvalues determine whether a system converges or diverges.

Diagonalization

Proposition 3

A matrix $A \in \mathbb{R}^{n \times n}$ is **diagonalizable** *if there exists a basis of $\mathbb{R}^n$ consisting of eigenvectors of A. Equivalently, A is diagonalizable $\iff$ the sum of the dimensions of its eigenspaces equals n.*

Meaning: Diagonalization means that A can be rewritten as

$$A = PDP^{-1},$$

where D is diagonal (containing eigenvalues) and the columns of P are eigenvectors. This representation simplifies computations such as powers of A, solving systems, or analyzing dynamics.

Note 3

Spectral Theorem: *Every real symmetric matrix is diagonalizable with an orthonormal eigenbasis. This ensures numerical stability and is the foundation for many algorithms in data science (e.g. PCA).*

Outline

SVD: Definition I

Theorem 10 (Singular Value Decomposition)

For any $A \in \mathbb{R}^{m \times n}$, there exist

- *an orthogonal matrix $U \in \mathbb{R}^{m \times m}$,*
- *an orthogonal matrix $V \in \mathbb{R}^{n \times n}$,*
- *a diagonal matrix $\Sigma \in \mathbb{R}^{m \times n}$ with nonnegative entries,*

such that

$$A = U\Sigma V^\top, \qquad \Sigma = \text{diag}(\sigma_1, \ldots, \sigma_r),\ \sigma_1 \geq \cdots \geq \sigma_r > 0.$$

Interpretation: The SVD expresses A as a product of three simple transformations:

1. $V^\top$: rotates the input space,
2. Σ: stretches or compresses along orthogonal axes (singular values),
3. U: rotates into the output space.

SVD: Definition II

Note 4

The singular values σ_i are the square roots of the nonzero eigenvalues of $A^\top A$ (or $AA^\top$). They measure how much A stretches vectors in different directions.

Best Low-Rank Approximation

Theorem 11 (Eckart–Young–Mirsky)

Given the SVD

$$A = \sum_{i=1}^{r} \sigma_i \, \boldsymbol{u}_i \boldsymbol{v}_i^\top \quad (r = \operatorname{rank}(A)),$$

the best approximation of A by a matrix of rank $k < r$ is

$$A_k = \sum_{i=1}^{k} \sigma_i \, \boldsymbol{u}_i \boldsymbol{v}_i^\top,$$

in both the spectral norm ($\|\cdot\|_2$) and Frobenius norm.

Meaning:

- A_k keeps only the k largest singular values.
- It is the "closest" rank-k matrix to A.
- This underlies many applications (data compression, PCA, noise reduction).

Outline

PCA: Intuition

Definition 12

Principal Component Analysis (PCA) finds new orthogonal directions (principal components) in the feature space that capture the maximum variance in the data. Projecting onto the first k components gives a k-dimensional representation that preserves as much variability (information) as possible.

Geometric view: PCA rotates the coordinate system so that:

- PC1 = direction of greatest variance.
- PC2 = next orthogonal direction of variance.
- Higher PCs capture progressively less variance.

Machine learning motivation:

- Reduce dimensionality while keeping essential information.
- Remove redundancy due to correlated features.
- Facilitate visualization of high-dimensional datasets.

PCA: Computation

Note 5

If $X \in \mathbb{R}^{n \times d}$ is a centered data matrix (rows = samples, columns = features):

1. *Compute covariance matrix $C = \frac{1}{n-1} X^\top X$.*
2. *Solve eigenvalue problem $Cv = \lambda v$.*
3. *Sort eigenvectors by decreasing eigenvalues.*
4. *Keep the first k eigenvectors $W_k = [v_1, \ldots, v_k]$.*
5. *Project data: $Z = XW_k$.*

Alternative: Compute PCA directly by the SVD of $X = U\Sigma V^\top$. The right singular vectors (columns of V) give the principal components, and singular values squared (σ_i^2) give the variance explained.

PCA: Interpretation

What PCA does:

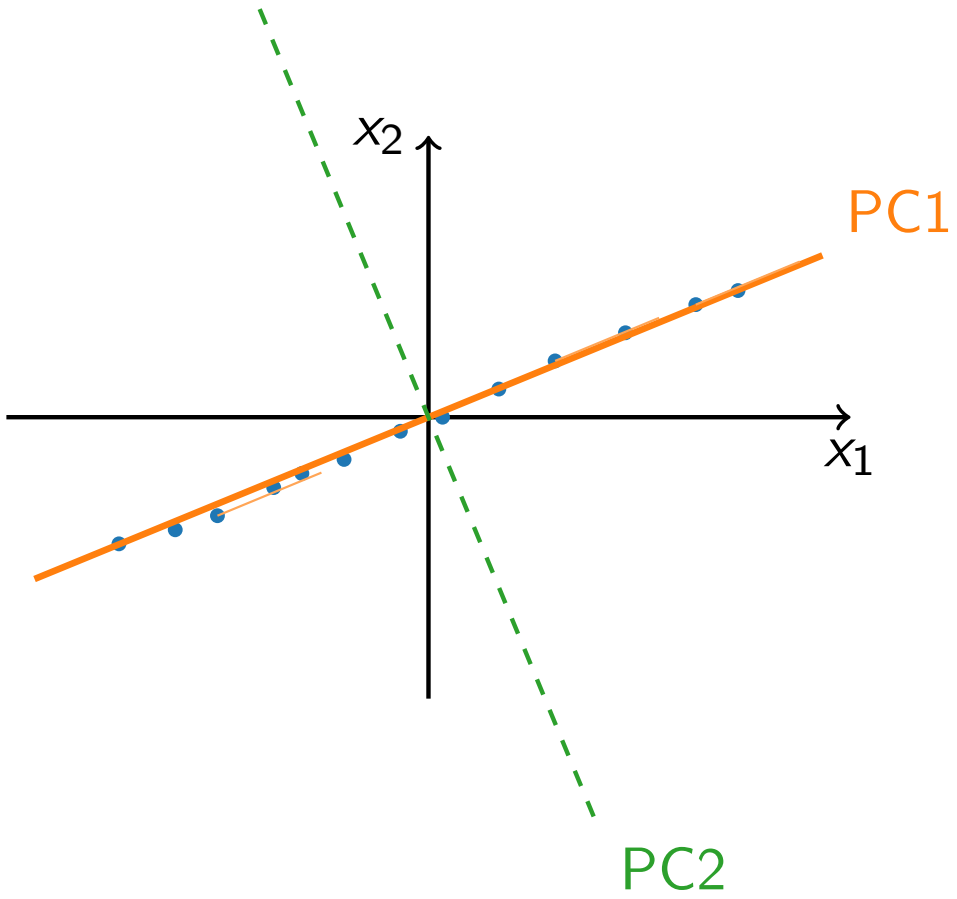
- Finds new coordinates aligned with variability in the data.
- Orders them by importance (variance explained).
- Allows truncation (keeping only first k PCs).

Applications:

- Visualization of high-dimensional data (reduce to 2D or 3D).
- Noise reduction (discard low-variance components).
- Preprocessing for regression or clustering (avoid multicollinearity).
- Computer vision: eigenfaces for face recognition.
- Recommender systems: latent factors from user–item matrices.

Key idea: PCA = data compression without (much) loss of information.

TikZ Illustration: PCA Projection (2D)



Variance Explained I

Proposition 4

Let $\lambda_1 \geq \cdots \geq \lambda_d \geq 0$ be the eigenvalues of the covariance matrix. The proportion of variance captured by the first k principal components is

$$\textit{Explained Variance Ratio}(k) = \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{d} \lambda_i}.$$

Interpretation:

- Each eigenvalue λ_i measures how much variance is along direction v_i.
- The ratio tells us how much information is kept when reducing to k dimensions.
- Example: If the first two PCs explain 90% of the variance, then projecting data to 2D keeps almost all the information.

Use in practice:

Variance Explained II

- Helps decide how many PCs to keep (e.g., choose k so that $> 95\%$ of variance is explained).
- Often visualized by a *scree plot*: eigenvalues or cumulative explained variance vs. number of components.

Outline

Definition 13

The **condition number** (in 2-norm) of a matrix A is

$$\kappa_2(A) = \frac{\sigma_{\max}(A)}{\sigma_{\min}(A)},$$

the ratio of the largest to the smallest singular value.

Interpretation:

- If $\kappa(A)$ is small (close to 1), A is *well-conditioned*: small changes in the data produce small changes in the solution.
- If $\kappa(A)$ is large, A is *ill-conditioned*: even tiny data errors or rounding in the computer can lead to large errors in the result.

Why it matters in PCA and ML:

Conditioning and Normalization II

- Covariance matrices with very different scales across features can have huge κ, making eigenvalue/SVD computations unstable.
- Multicollinearity (highly correlated features) $\Rightarrow$ small singular values $\Rightarrow$ large κ.
- This can cause unreliable PCs or regression coefficients.

Note 6

Practical tip:

- *Always center features (mean = 0).*
- *Often scale features (variance = 1) to reduce ill-conditioning.*
- *Use* **SVD-based PCA**, *which is numerically more stable than directly solving the eigenvalue problem.*

Outline

NumPy: Eigenvalues & Eigenvectors

```
import numpy as np

A = np.array([[2., 1., 0.],
              [1., 2., 1.],
              [0., 1., 2.]])

# Symmetric => use eigh (more stable)
w, V = np.linalg.eigh(A) # w: eigenvalues (asc), V: eigenvectors
idx = np.argsort(w)[::-1] # sort descending if desired
w, V = w[idx], V[:, idx]

print("Eigenvalues:", w)
print("Orthonormal eigenvectors (columns of V):\n", V)
```

NumPy: SVD & Best Rank-k Approximation

```
import numpy as np

X = np.random.randn(100, 40) # data matrix (centered features recommended)
U, s, Vt = np.linalg.svd(X, full_matrices=False) # X = U @ np.diag(s) @ Vt

k = 5
Xk = (U[:, :k] * s[:k]) @ Vt[:k, :]
err_fro = np.linalg.norm(X - Xk, 'fro') / np.linalg.norm(X, 'fro')

print(f"Relative Frobenius error (k={k}): {err_fro:.3f}")
```

Scikit-Learn: PCA (Variance Explained)

```
import numpy as np
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# X: shape (n_samples, n_features)
X = np.random.randn(500, 50)

# Center & scale before PCA (common in practice)
Xz = StandardScaler(with_mean=True, with_std=True).fit_transform(X)

pca = PCA(n_components=0.95, svd_solver='full') # keep 95% variance
X_pca = pca.fit_transform(Xz)

print("Selected components:", pca.n_components_)
print("Explained variance ratio (first 10):", pca.explained_variance_ratio_
    [:10])
print("Cumulative:", np.cumsum(pca.explained_variance_ratio_)[:10])
```

Outline

Kernel, Image, Rank: Worked Example

Example 14

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

- Compute rank(A) via row-reduction or SVD.
- Find a basis for ker(A).
- Verify Rank–Nullity.

Note 7

In Python, use `np.linalg.matrix_rank(A)` *and solve* $A\boldsymbol{x} = \mathbf{0}$ *with* `scipy.linalg.null_space`.

PCA on Real Data (Sketch)

Example 15

Take the digits dataset (8x8 images) from `sklearn`. Apply PCA and visualize the first two PCs and reconstructions with $k = 10, 20, 40$.

Note 8

Use `sklearn.decomposition.PCA` and plot cumulative explained variance to select k.

Outline

Conclusion: Mathematical Takeaways

- **Vectors and Matrices:** building blocks for representing data and transformations.
- **Linear Maps:** connect matrices with geometric effects (scaling, rotation, projection).
- **Kernel & Image:** describe information lost and preserved by a transformation.
- **Rank:** measures the number of independent directions; links to invertibility.
- **Eigenvalues/Eigenvectors:** reveal invariant directions and scaling factors.
- **Matrix Decompositions (SVD):** universal tool for stability, compression, and analysis.

Key idea: Linear algebra provides the language and tools to manipulate high-dimensional data efficiently.

Conclusion: Relevance to Machine Learning

- **Regression:** solution formulas involve inverses and matrix factorizations.
- **Dimensionality reduction:** PCA projects data onto principal components to reduce noise and redundancy.
- **Neural networks:** each layer applies linear transformations before nonlinear activations.
- **Data compression:** SVD and PCA approximate large datasets by low-rank structures.
- **Numerical stability:** conditioning and normalization are crucial for reliable algorithms.

Take-home message: Without linear algebra, machine learning cannot scale or even function. *This chapter builds the bridge from abstract math to practical AI.*

Key Takeaways

- rank + nullity $= n$; invertibility $\Leftrightarrow$ full rank.
- Eigen-decomposition diagonalizes symmetric matrices; use `eigh`.
- SVD exists for all matrices; gives optimal low-rank approximations.
- PCA = SVD on centered data; keep components by variance explained.
- Always consider conditioning and scaling for stable computations.