Practical Session (M1 Computer Science) — Linear Algebra for Machine Learning

Real Data Applications in Python: Performance & Accuracy

Youssef SALMAN

Academic Year 2025–2026

Objectives

- Manipulate vectors and matrices and connect these operations to ML tasks (similarities, PCA, SVD).
- Compare **accuracy** and **execution time** between homemade implementations and optimized library functions.
- Develop skills in **profiling** (time measurement) and **numerical validation** (errors, stability).

Instructions

For each question:

1. Read the **Reminder** of the method or algorithm.
2. Implement a naive Python version (loops, direct formulas).
3. Compare with the optimized version using `NumPy` or `scikit-learn`.
4. Measure and discuss execution times.

Exercises

Question 1 — Determinant of a Matrix

Reminder: The determinant of an $n \times n$ matrix can be computed recursively using Laplace expansion:

$$\det(A) = \sum_{j=1}^{n} (-1)^{1+j} a_{1j} \det(M_{1j})$$

where M_{1j} is the minor of A obtained by removing row 1 and column j. Complexity grows as $\mathcal{O}(n!)$.

Task: Implement this recursive method in Python and compare with `numpy.linalg.det`.

```
def my_determinant(M):
    # TODO: implement recursive Laplace expansion
    return det
```

Measure execution time for matrices of size $n = 3, 5, 7, 10$ and discuss complexity.

—

Question 2 — Eigenvalues and Eigenvectors

Reminder: The **Power Iteration** algorithm finds the dominant eigenvalue λ of a matrix A:

1. Choose random vector b_0.
2. Iterate $b_{k+1} = \dfrac{Ab_k}{\|Ab_k\|}$.
3. Approximate eigenvalue: $\lambda_k = \dfrac{b_k^T A b_k}{b_k^T b_k}$.

Task: Implement Power Iteration to approximate the largest eigenvalue of a symmetric matrix. Compare results and execution times with `numpy.linalg.eig`.

—

Question 3 — Singular Value Decomposition (SVD)

Reminder: The first singular vector u_1 of a matrix A can be obtained by computing the dominant eigenvector of AA^T. Then $v_1 = A^T u_1 / \|A^T u_1\|$. Iterating gives an SVD-like decomposition.

Task: Implement a naive SVD approach (first singular vector via power iteration). Compare with `numpy.linalg.svd`. Apply both methods to a grayscale image (100×100) and compare reconstruction quality and execution time.

—

Question 4 — Principal Component Analysis (PCA)

Reminder: PCA steps:

1. Center data: $X_c = X - \bar{X}$.
2. Compute covariance: $C = \frac{1}{n-1} X_c^T X_c$.
3. Find eigenvectors/eigenvalues of C.
4. Project data on top k eigenvectors.

Task:

- Implement PCA manually (following above steps).
- Compare with `sklearn.decomposition.PCA`.
- Use the `digits` dataset from scikit-learn and compare execution times and results.

—

Bonus — Cosine Similarity

Reminder: For two vectors $x, y \in \mathbb{R}^n$:

$$\cos(\theta) = \frac{x \cdot y}{\|x\|\|y\|}$$

Task: Implement cosine similarity manually. Compare with `sklearn.metrics.pairwise.cosine_simi`
Test with 10^5 vectors of dimension 300, and compare execution times.