

Exercise #03

IT University of Copenhagen (ITU)
Data Mining KSD (DAMIN)
(Autumn 2025)

09 September 2025

Introduction This exercise list will provide you with hands-on experience in applying basic linear algebra concepts using Python. The tasks are designed to introduce you to vectors, matrices, and fundamental operations that are foundational to data mining and machine learning. Each exercise is approachable for beginners and offers a practical understanding of key concepts. The learning objectives for this exercise encompass:

- Understand and apply basic operations with vectors and matrices.
- Implement linear transformations and explore their effects.
- Utilize Python libraries like NumPy for efficient mathematical computations.
- Solve simple linear systems of equations.

Exercise 03.01. Vector Operations (15-20 minutes) – Write a Python program that performs basic operations on two vectors. The program should (at least) compute the sum, difference, dot product, and cross product of two vectors. Use Python lists or NumPy arrays to represent the vectors.

- **Instructions:**

- Create two vectors as Python lists or NumPy arrays, for example:

$$v_1 = [1, 2, 3]$$

$$v_2 = [4, 5, 6]$$

- Compute and display the result of the following operations:

- * Vector addition ($v_1 + v_2$)
 - * Vector subtraction ($v_1 - v_2$)
 - * Dot product ($v_1 \cdot v_2$)
 - * Cross product ($v_1 \times v_2$)

- **Example Output:**

- If $v_1 = [1, 2, 3]$ and $v_2 = [4, 5, 6]$, the program should output:

```
v1 + v2 = [5, 7, 9]
v1 - v2 = [-3, -3, -3]
v1 . v2 = 32
v1 x v2 = [-3, 6, -3]
```

Exercise 03.02. *Matrix Operations* (20-25 minutes) – Develop a Python program to perform basic operations on matrices. The program should create two matrices and compute their sum, difference, product, and transpose. Use Python lists or NumPy arrays to represent the matrices.

- **Instructions:**

- Create two matrices, for example:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \quad B = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

- Perform and display the following operations:

- * Matrix addition ($A + B$)
- * Matrix subtraction ($A - B$)
- * Matrix multiplication ($A \times B$)
- * Transpose of matrices A and B (A^T and B^T)

- **Example Output:**

- If $A = [[1, 2], [3, 4]]$ and $B = [[5, 6], [7, 8]]$, the program should output:

```
A + B = [[6, 8], [10, 12]]
A - B = [[-4, -4], [-4, -4]]
A * B = [[19, 22], [43, 50]]
A^T = [[1, 3], [2, 4]]
B^T = [[5, 7], [6, 8]]
```

Exercise 03.03. *Two-Dimensional Linear Transformation* (20-30 minutes) – Write a Python program that applies a linear transformation to a set of 2D points. Define a 2×2 transformation matrix (e.g., rotation or scaling matrix) and apply it to a list of 2D points. Display the original and transformed points. – *Homework Exercise*

- **Instructions:**

- Define a 2×2 transformation matrix T (e.g., rotation or scaling matrix).

- Create a list of 2D points, e.g., $points = [(1, 1), (2, 2), (3, 3)]$.
- Apply the transformation to each point.
- Display the original and transformed points.

- **Example Output:**

- If $T = [[0, -1], [1, 0]]$ (90-degree rotation), the program should output:

Original points: $[(1, 1), (2, 2), (3, 3)]$

Transformed points: $[(-1, 1), (-2, 2), (-3, 3)]$

Exercise 03.04. Three-Dimensional Linear Transformations (30-35 minutes) – In this exercise, you will explore the concept of linear transformations in three dimensions. You will implement various 3D transformations using matrices and visualize their effects on a set of points in three-dimensional space. You will define and apply different linear transformations, such as scaling, rotation, and shearing, to a set of 3D points. The exercise will help you understand how linear transformations can manipulate objects in three-dimensional space.

– *Homework Exercise*

- **Instructions:**

- Create a Set of 3D Points:

- * Define a set of points in 3D space that form a simple object, such as a cube or pyramid. Represent these points as a NumPy array. For example:

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

- Define Transformation Matrices:

- * Create transformation matrices for different 3D transformations, such as scaling, rotation, and shearing. For example:

$$T_{scale} = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix} \quad T_{rotate} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad T_{shear} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Apply Transformations:

- * Apply each transformation to the set of points by multiplying the transformation matrix with the point coordinates.
- * Display the new set of transformed points.
- Interpret the Results:
 - * Discuss the effects of each transformation on the shape and orientation of the object in 3D space. Explain how different transformations can be combined to achieve complex effects.
 - * Write a brief summary (as comments in the code) explaining the findings.

- **Example Output:**

- After applying a scaling transformation, the program might output:

```
Original points:
```

```
[[0 0 0]
 [1 0 0]
 [1 1 0]
 [0 1 0]
 [0 0 1]
 [1 0 1]
 [1 1 1]
 [0 1 1]]
```

```
Transformed points (Scaling by 2):
```

```
[[0 0 0]
 [2 0 0]
 [2 2 0]
 [0 2 0]
 [0 0 2]
 [2 0 2]
 [2 2 2]
 [0 2 2]]
```

Exercise 03.05. Solving a System of Linear Equations (25-30 minutes) – Develop a Python program to solve a system of linear equations using matrices. Represent the system in matrix form $Ax = b$ and use NumPy to solve for the vector x . Display the solution and test the program with different systems of equations.

- **Instructions:**

- Consider the system of equations:

$$\begin{cases} 2x + 3y = 5 \\ 4x + y = 6 \end{cases}$$

- Represent this system in matrix form $Ax = b$.
- Use NumPy to solve for x (the vector $[x, y]$).
- Display the solution.
- Try different systems of equations to test the program.

Exercise 03.06. *Eigenvalues and Eigenvectors* (20-25 minutes) – Write a Python program to compute the eigenvalues and eigenvectors of a 2×2 matrix. Use NumPy to perform the calculations and display the results. – *Homework Exercise*

• **Instructions:**

- Create a 2×2 matrix C , for example:

$$C = \begin{bmatrix} 4 & 1 \\ 2 & 3 \end{bmatrix}$$

- Use NumPy to compute the eigenvalues and eigenvectors.
- Display the results.
- Try different matrices to test the program.

• **Example Output:**

```
Eigenvalues: [5.,  2.]
Eigenvectors: [[1,  1], [-0.5, 1]]
Normalized: [[0.70710678, -0.4472136], [0.70710678,  0.89442719]]
```

Exercise 03.07. *Grade Evaluation – Bases, Projections and Approximations.* (20-45min)

– Review and implement algorithms to identify the most relevant components of vectors in spaces of interest.

- *Introductions.* For this exercise, we focus on some mathematical qualities of vectors and linear transformation to flex the muscles of linear algebra. Formal definitions for *vector spaces* can be found at **Wikipedia**, or you can ask your friendly (albeit dumb and sometimes lying) LLM model of the day. In the common and less interesting non-mathematical syllabus however, what really matters is the following:

1. You can add 2 (or more) vectors and still get a vector: $v = a + b$.
2. You can multiply a vector by a number and still get a vector: $v = \alpha w$

Notation!: Here, the Greek letters are numbers, and Latin, vectors.

- *Independence* A given vector v is *independent* of the vectors a , b and c , if you can't find numbers μ , ν , ω such that:

$$v = \mu a + \nu b + \omega c.$$

If you think of vectors as columns. It is the same: does linear system generated by the equation above has a solution? If there is one (or more), the collection v, a, b, c of vectors is said to be *linearly dependent*. Cool right? There are several nice results about this theme. Again look it up if you want. The most relevant one is: *If you're working in D dimensions, a collection of independent vectors has at most D vectors.* In simpler terms, in $2D$ if you have two vectors that are independent, you can't have a third. In $3D$ if 3 vectors are independent you can't have a fourth and so on. So if you have a collection of 3 independent vectors in $3D$ you have a *base for the space*.

- *Problem 1:* Write a program that look at a collection (a list) of vectors of length D and find all possible bases if any, considering the exceptions:

- “There are less than D independent vectors in the set.”
- “The list of vectors contains vectors of different dimensions.”

- *Example Output:*

- for a list $L = [[1, 2, 3, 4], [5, 6, 7, 8], [6, 8, 10, 12], [4, 4, 4, 4], [3, 2, 1, 0]]$, the program should output:

"There are less than D independent vectors in the set."

- *Projections over Vectors:* By the use of the *dot product*, we can decompose a vector into orthogonal and parallel components, with respect to another vector. The equations as quite simple. If $\|a\|$ is the length of a , we denote:

$$\hat{a} = \frac{a}{\|a\|},$$

as a 's direction. The parallel component of b over a is:

$$b_{\parallel} = (\hat{a} \cdot b)\hat{a}.$$

Its counterpart is the orthogonal component $b_{\perp} = b - b_{\parallel}$.

- *Problem 2:* Write a program that calculates the projection of a vector over another, and returns the parallel component of the projected vector.
- *Example Output:*

- for $v = [1, 2, 3, 4]$ and $w = [0, 0, 1, 0]$, the program should output:

"[0,0,3,0]"

- **Projections over Subspaces:** In the 3-dimensional case, if one has independent vectors, such as $a = [1, 1, 0]$ and $b = [1, 0, 2]$ those two can span a 2-dimensional *subspace* inside the 3D space. This will be all possible vectors that can be written in the form:

$$v = \mu a + \nu b.$$

and denote it $W = \text{Span}\{a, b\}$ and say that a and b spans the subspace. The vector $c = [0, 0, -1] = a - b$ is in the W , but $d = [0, 1, 0]$ is not, because there are no solutions to $d = \mu a + \nu b$ (can you check this with problem 1's solution?). This idea extends to all dimensions. For example, in 4-dimension, 1 vector spans a 1D subspace. 2 independent vectors spans a 2D subspace, and 3 independent vectors a 3D. Observe that two different sets of spanning vectors can generate the same subspace.

We are now interested in *projections* over subspaces. That is, *given a vector v and a subspace W , what is the closest vector v_W to the original v* . This is the most complicated step, but there is a list of steps.

1. Given the subspace W of dimension n , i.e., spanned by n independent vectors, find a *orthonormal basis* $\{e_1, e_2, \dots, e_n\}$ for W .
2. For a given vector v , make the decomposition

$$v = v_W + v_{\perp W}$$

$$\text{with } v_{\parallel} = (v \cdot e_1)e_1 + \dots + (v \cdot e_n)e_n$$

3. v_W is the desired approximation, and $v_{\perp W}$ is the remainder.

An orthonormal basis, is a basis in which each vector has length 1 and the dot product of each pair of distinct basis vector is zero. To find a orthonormal basis from the given basis, one uses the Gram-Shmidt process. If one starts with an orthonormal basis, there is no need for the first step.

- *Problem 3:* Write a program that calculates the projection of a vector over a subspace generated by a list of given orthonormal vectors, and returns the parallel component of the projected vector

- *Example Output:*

- For the spanning set $L = [[\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0, 0], [\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}}, 0, 0], [\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}, 0, \frac{1}{\sqrt{3}}]]$ and $v = [2, 3, 4, 5]$, the program should output:

"[2,3,0,5]"