Spring 2026

Programmation 2

TP 10 — *Lisez attentivement l'ensemble du document.* — *Nabil Mustafa*

Consignes

1. *L'objectif est d'apprendre la programmation en Python.* ***Inutile de se précipiter !*** *Prenez votre temps et terminez correctement tous les TP précédents avant d'aborder celui-ci.*

2. *Pour réussir,* ***lisez le texte attentivement et lentement****. Ne parcourez pas les consignes en diagonale. Lisez chaque phrase une fois, puis une deuxième. Si un passage reste flou après plusieurs lectures, c'est le moment idéal pour poser une question au professeur. C'est ainsi que l'on apprend efficacement.*

3. *Avant de commencer,* ***mettez votre téléphone en mode « ne pas déranger » et rangez-le dans votre sac****. Chaque notification consultée brise votre concentration et vous oblige à reprendre le fil de votre raisonnement, ce qui allonge considérablement le temps nécessaire pour terminer l'exercice et favorise les erreurs.*

4. *Adoptez une approche progressive : lisez une instruction, puis exécutez-la immédiatement. Évitez de lire tout le TP d'un coup.* ***Avancez étape par étape pour rester concentré sur l'action immédiate et ne pas vous sentir submergé par un long texte.***

5. *Souvenez-vous : les étudiants qui réussissent le mieux ne sont pas ceux qui « finissent en premier », mais ceux qui* ***lisent avec soin, posent des questions lorsqu'ils bloquent et restent concentrés*** *jusqu'au bout.*

Votre objectif aujourd'hui n'est pas d'aller vite, mais de comprendre profondément.

Tâche 0: Exercises

(a) Connectez-vous à `LINUX` avec votre compte utilisateur.

(b) Téléchargez le Jupyter Notebook fichier `Lec10-numpy.ipynb`, disponible sur le site web du module.

(c) Pour le lire, ouvrez un `terminal` dans le *même dossier* que le fichier `Lec10-numpy.ipynb`, et exécuter le command :

```
jupyter-notebook Lec10-numpy.ipynb
```

(d) Pour exécuter un code Python dans le Jupyter Notebook, cliquez sur le code et appuyez sur "SHIFT + ENTER".

(e) **Lisez toutes les informations** dans cette Jupyter Notebook.

(f) **Faites tous les exercices** dans cette Jupyter Notebook.

Rappelle: linear algebra + numpy

Concept	Mathématique	Python
Vecteur 2D	$\mathbf{v} = \begin{pmatrix} x \\ y \end{pmatrix}$ représente un point ou une flèche depuis l'origine $(0, 0)$.	`import numpy as np` `v = np.array([x, y])`

Opérations	$\mathbf{v}_1 + \mathbf{v}_2$ (addition)
	$c \cdot \mathbf{v}_1$ (multiplication scalaire)
	$\mathbf{v}_1 \cdot \mathbf{v}_2$ (produit scalaire)
	$\|\mathbf{v}\|$ (longueur)
		`v1 + v2`
		`c * v1`
		`np.dot(v1, v2)`
		`np.linalg.norm(v)`

Normalisation	Pour rendre la norme d'un vecteur égale à 1: $\mathbf{u} = \frac{\mathbf{v}}{\|\mathbf{v}\|}$	`u = v / np.linalg.norm(v)`
Projection	Ombre de $\mathbf{a}$ sur $\mathbf{b}$ $\text{proj}_{\mathbf{b}}(\mathbf{a}) = \left(\mathbf{a} \cdot \frac{\mathbf{b}}{\|\mathbf{b}\|}\right) \frac{\mathbf{b}}{\|\mathbf{b}\|}$	`bp = b / np.linalg.norm(b)` `proj = (np.dot(a, bp)) * bp`
Orthogonalisation	Partie perpendiculaire à $\mathbf{b}$ $\mathbf{a}_\perp = \mathbf{a} - \text{proj}_{\mathbf{b}}(\mathbf{a})$	`a_perp = a - proj`
Réflexion	Reflection autour de la droite engendrée par $\mathbf{b}$ $\text{reflection}(\mathbf{a}) = 2\ \text{proj}_{\mathbf{b}}(\mathbf{a}) - \mathbf{a}$	`refl = 2*proj - a`

Sous-espace	L'ensemble des points $c \cdot \mathbf{v}_1$ (pour $c \in \mathbb{R}$) forme une **droite** passant par l'origine.
	L'ensemble des points $c\mathbf{v}_1 + d\mathbf{v}_2$ (pour $c, d \in \mathbb{R}$) forme un **plan** (tout l'espace 2D) si les vecteurs sont indépendants (non colinéaires). S'ils sont colinéaires, cela donne seulement une droite.
		`# Droite`
		`t = np.linspace(-5,5,100)`
		`points_droite = np.array([c*v1 for c in t])`
		
		`# Plan`
		`points_plan = []`
		`for c in np.linspace(-5,5,10):`
		`  for d in np.linspace(-5,5,10):`
		`  points_plan.append(c*v1 + d*v2)`
		`points_plan = np.array(points_plan)`

Rappelle: Dessins

Les fonctions ci-dessous ont déjà été écrites pour vous. **Ne les modifiez pas.** Utilisez-les simplement dans votre code.

draw_vector2D() : Dessine une flèche (vecteur)

```
draw_vector2D(vector=v1, color='red', label='v1')
draw_vector2D(origin=some_point, vector=v2, color='blue')
```

Paramètres :

- `origin` : point de départ (par défaut = [0,0])
- `vector` : le vecteur à dessiner
- `color` : couleur de la flèche
- `label` : texte affiché à côté (optionnel)

draw_point2D() : Dessine un point

```
draw_point2D(point=my_point, color='green', label='P')
```

draw_line2D() : Dessine une droite infinie (définie par sa normale)

```
draw_line2D(normal=np.array([a,b]), color='purple')
```

La droite est l'ensemble des points $\mathbf{x}$ tels que $\mathbf{normal} \cdot \mathbf{x} = 0$.

plt_run() : Affiche le graphique

```
plt_run(title='Mes vecteurs')
```

Appelez cette fonction **une seule fois** après tous vos `draw_*` pour afficher le résultat. Fermez la fenêtre en appuyant sur la touche `'q'`.

Exemple complet

```
import numpy as np
import matplotlib.pyplot as plt

v1 = np.array([1, 5])
v2 = np.array([4, -5])

draw_vector2D(vector=v1, color='red', label='v1')
draw_vector2D(vector=v2, color='green', label='v2')
draw_point2D(point=v1+v2, color='blue', label='somme')

plt_run(title='Exemple')
```

Tâche 1: Exécuter Python

(a) Connectez-vous à `LINUX` avec votre compte utilisateur.

(b) **Téléchargez** le fichier `vectors2D.py`.

(c) **Éditer** ce fichier avec l'éditeur de votre choix. Par exemple, ouvrez un `terminal` dans le même dossier que le fichier `vectors2D.py`, et exécuter le command:

```
kate vectors2D.py
```

(d) **Exécuter** le code avec `Python`: ouvrez un `terminal` dans le *même dossier* que le fichier `vectors2D.py`, et exécuter le command :

```
python3 vectors2D.py
```

Tâche 2: Defining and drawing vectors in 2D

(a) Définissez trois variables de vector `numpy` appelées `v1`, `v2` et `v3` telles que

$$\texttt{v1} \leftarrow \text{le vecteur } \begin{pmatrix} 1 \\ 5 \end{pmatrix}$$
$$\texttt{v2} \leftarrow \text{le vecteur } \begin{pmatrix} 4 \\ -5 \end{pmatrix}$$
$$\texttt{v3} \leftarrow \text{le vecteur } \begin{pmatrix} -6 \\ -1.3 \end{pmatrix}$$

(b) Normalisez le vecteur `v2` et placez ce vecteur normalisé dans la variable `numpy q`.

(c) Dessinez ces vecteurs en utilisant `matplotlib`, avec

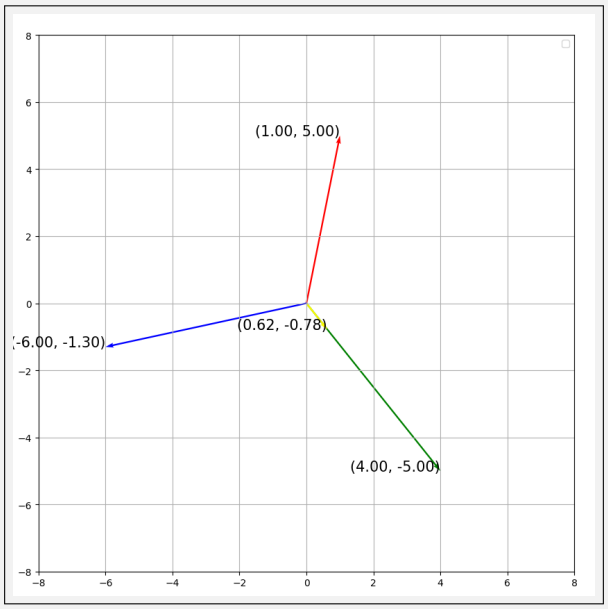
`v1` en couleur red
`v2` en couleur green
`v3` en couleur blue
`q` en couleur yellow

```
#ci-dessous: tache 2

    #ECRIVEZ VOTRE CODE ICI

#ci-dessus: tache 2
```

Si vous le faites correctement, l'exécution du programme affichera :



Pour passer à la tâche suivante, vous devez fermer la fenêtre en appuyant sur la lettre 'q'.

Tâche 3: Subspaces in 2D

(a) Pour le vecteur `v1`, dessinez tous les points possibles (dans green)

```
c · v1
```

où c prend les valeurs $-3, -2.9, -2.8, \ldots, 2.9, 3$.

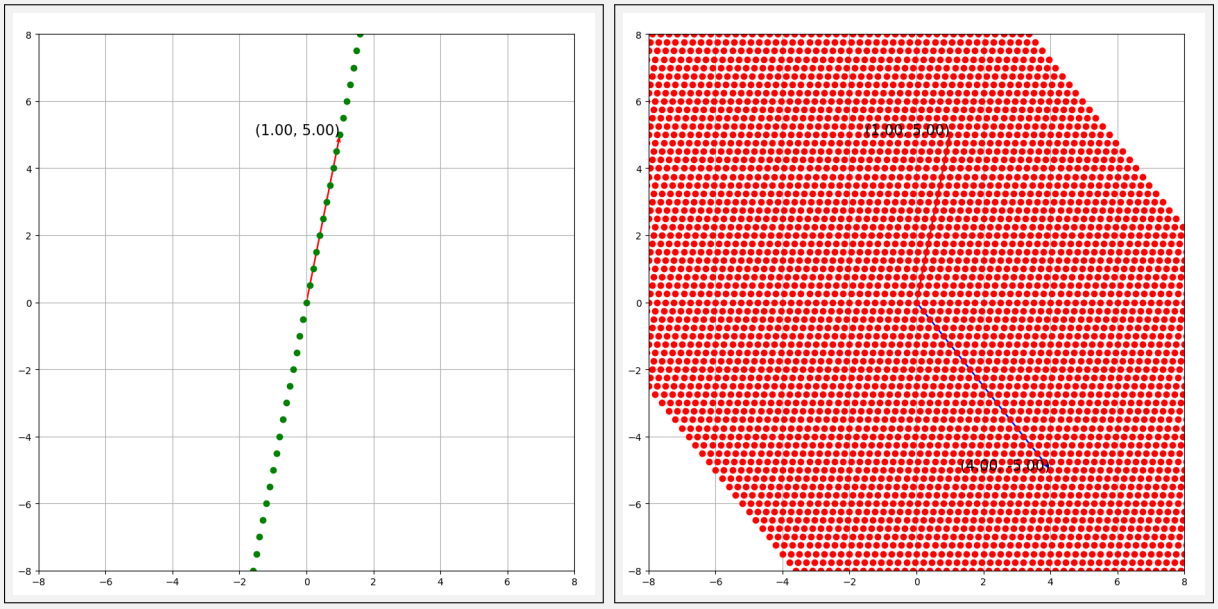
Autrement dit, c prend des valeurs entre -3 et 3 à une distance de 0.1.

(b) Pour les vecteurs `v1` et `v2`, dessinez tous les points possibles (dans red)

```
c · v1 + d · v2
```

où c prend des valeurs entre -2 et 2 à une distance de 0.05 et d prend des valeurs entre -2 et 2 à une distance de 0.05.

Si vous le faites correctement, l'exécution du programme affichera :



Pour passer à la tâche suivante, vous devez fermer la fenêtre en appuyant sur la lettre 'q'.

Tâche 4: Projections in 2D

(a) Définir un nouveau vecteur `v5`

$$\texttt{v5} \leftarrow \text{le vecteur } \begin{pmatrix} -4 \\ 4 \end{pmatrix}$$

Dessiner `v1` et `v5` en couleur blue.

(b) **Soit `v5_proj_to_v1` le vecteur qui est `v5` projeté dans la direction de `v1`.**

Calculer et dessiner `v5_proj_to_v1` en couleur red.

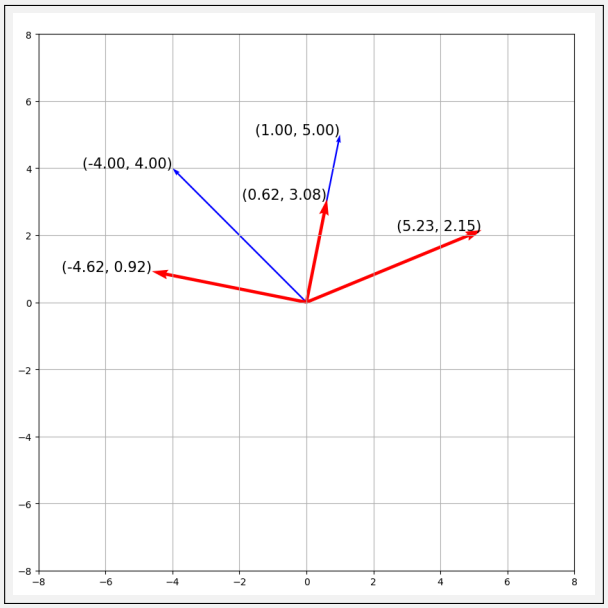
(c) **Soit `v5_orthogonal` ce qui reste de `v5` après que sa composante dans la direction de `v1` ait été supprimée.**

Calculer et dessiner `v5_orthogonal` en couleur red.

(d) Soit `v5_reflected_around_v1` le vecteur qui est **réfléchi** autour du sous-espace engendré par `v1`.

Calculer et dessiner `v5_reflected_around_v1` en couleur red.

Si vous le faites correctement, l'exécution du programme affichera :



Pour passer à la tâche suivante, vous devez fermer la fenêtre en appuyant sur la lettre 'q'.

Tâche 5: Gram-Schmidt in 2D

(a) Définir deux nouveaux vecteurs

$$\texttt{v6} \leftarrow \text{le vecteur } \begin{pmatrix} 6 \\ -2 \end{pmatrix}$$

$$\texttt{v7} \leftarrow \text{le vecteur } \begin{pmatrix} 7 \\ -1 \end{pmatrix}$$

Dessiner `v6` et `v7` en couleur blue.

(b) **Soit `v6_unit` le vecteur unitaire de `v6`.**

Calculer et dessiner `v6_unit` en couleur red.

(c) **Soit `v7_normal` ce qui reste de `v7` après que sa composante dans la direction de `v6` ait été supprimée.**

(d) **Soit `v7_normal_unit` le vecteur unitaire issu de `v7_normal`.**

Calculer et dessiner `v7_normal_unit` en couleur red.

(e) Vérifier que `v7_normal_unit` et `v6_unit` sont orthogonaux.

Si vous le faites correctement, l'exécution du programme affichera :