



Programmation Orientée Objet (Python)

TP3 : Visualisation scientifique - NumPy et Matplotlib

[A.Martin](#)

2026-2027



Visual Studio Code



Objectif

Maîtriser les bases de la visualisation scientifique en Python avec NumPy et Matplotlib.

À la fin de la séance, vous saurez :

- générer et manipuler des données numériques avec NumPy ;
- tracer des courbes, nuages de points, histogrammes et images 2D avec Matplotlib ;
- personnaliser vos figures (titres, unités, légendes, styles, couleurs) ;
- organiser des sous-figures pour comparer plusieurs graphiques ;
- exporter vos figures proprement (résolution, formats).

La séance servira de référence pour vos futurs travaux pratiques, notamment dans le traitement et la visualisation de données scientifiques et techniques.

Durée estimée : 4 heures

1. Introduction

La visualisation de données est une compétence essentielle pour tout ingénieur. Que ce soit pour analyser un signal expérimental, comparer des distributions, ou représenter une image, Python offre des outils puissants. Nous allons utiliser deux bibliothèques fondamentales :

- *NumPy* : pour générer et manipuler efficacement des tableaux de nombres.
- *Matplotlib* : pour tracer et personnaliser des graphiques scientifiques.

2. Exercices

Les scripts Python respecteront le squelette suivant :

```
#!/usr/bin/env python3
# Importez toutes les bibliothèques nécessaires (numpy, matplotlib,
# etc.)

# Définissez éventuellement des fonctions

def main():
    # Vos instructions

if __name__ == "__main__":
    main()
```

En Python, on rencontre souvent en fin de fichier la structure suivante :

```
if __name__ == "__main__":
    main()
```

Cette convention permet d'exécuter la fonction `main()` seulement si le fichier est lancé directement avec la commande « `python fichier.py` », pas si le fichier est importé comme un module (« `import fichier` »). Dans le premier cas, la variable d'environnement interne `__name__` se voit automatiquement associer la valeur `__main__` par l'interpréteur Python.

Travail préparatoire

1. Ouvrez un terminal.
2. Dans votre dossier « `tp_python` », créez le sous-dossier « `tp2_visualization` » et déplacez-vous dedans.

3. Créez l'arborescence suivante :

```
tp2_visualization/
|-- data/
|   |-- out/           # Dossiers de sortie pour vos fichiers (ex. CSV, images)
|-- src/               # Scripts Python à compléter
|-- README.md
|-- requirements.txt    # Bibliothèques Python utilisées dans le TP
```

4. Créez l'environnement virtuel « env_tp2 » et vérifiez la nouvelle arborescence :

```
tp2/
|-- data/
|   |-- out/           # Dossiers de sortie pour vos fichiers (ex. CSV, images)
|-- env_tp2/           # Fichiers de l'environnement virtuel ...
|   |--
|-- src/               # Scripts Python à compléter
|-- README.md
|-- requirements.txt    # Bibliothèques Python utilisées dans le TP
```

5. Activez l'environnement virtuel et installez les bibliothèques « numpy » et « matplotlib » :

```
pip install -r requirements.txt
```

6. Vérifiez leur installation :

```
pip list
```

7. Désactivez ensuite l'environnement virtuel (vous l'activerez depuis VSCode).

Exercice n° 1 : Signal amorti bruité avec Numpy

Créez le script « ex01_signal_numpy.py » dans le dossier « src ». Celui-ci va générer, analyser et filtrer un signal amorti bruité de la manière suivante :

1. Génération de l'axe temporel t sur l'intervalle $[0, 50]$ avec 501 points.
2. Calcul de $f = Ae^{-\alpha t} \cos(\omega t)$ avec les paramètres $A = 5.0$, $\alpha = 0.5$, $\omega = 2\pi$.
3. Création d'un générateur aléatoire déterministe et d'un bruit gaussien d'écart type 0.2 de même forme que t .
4. Calcul de la fonction $f_{bruitee} = f + \text{bruit}$.
5. Calcul et affichage de :
 - la moyenne, l'écart-type, le min et le max de $f_{bruitee}$ (en une ligne lisible),
 - le pourcentage de points tels que $|f_{bruitee}| < 0.1$ (utilisez un masque booléen !).
6. Détection de passages par zéro (sur le signal idéal f) :
 - Trouve les indices i tels que $f[i]$ et $f[i+1]$ ont des signes opposés.
 - Affiche les instants de passage par zéro.

Contraintes

- Contentez-vous d'afficher les statistiques et les valeurs seulement.

Exercice n° 2 : Affichage d'une courbe avec Matplotlib

Créez le script « ex02_affichage_signal.py » dans le dossier « src ». Celui-ci doit :

1. Tracer la fonction $f_{bruitee}$ précédente sur l'intervalle $[0, 50]$.
2. Ajouter un titre et une grille à la figure.

Exercice n° 3 : Comparer des distributions avec leurs histogrammes

Créez le script « ex03_histogrammes.py » dans le dossier « src ». Celui-ci doit :

1. Générer deux jeux de données aléatoires avec des distributions normales (méthode `random.normal()` de Numpy).
2. Tracer les histogrammes sur le même graphique avec de la transparence.
3. Ajouter une légende pour distinguer les distributions.

Exercice n° 4 : Créer un nuage de points avec couleurs variables

Créez le script « `ex04_nuage_couleur.py` » dans le dossier « `src` ». Celui-ci doit :

1. Générer des données aléatoires x , y et z .
2. Tracer le nuage de points de y en fonction de x et changer la couleur des points suivants les valeurs de z .

Exercice n° 5 : Afficher plusieurs images dans des sous-figures

Créez le script « `ex05_sous_figures.py` » dans le dossier « `src` ». Celui-ci doit :

1. Charger quatre images différentes et les afficher dans une grille 2×2 (vous utiliserez des images de votre choix).
2. Ajouter des titres pour chaque image.
3. Masquer les axes pour une meilleure présentation.

Exercice n° 6 : Tracer des courbes avec des styles différents

Créez le script « `ex06_courbes_et_styles.py` » dans le dossier « `src` ». Celui-ci doit :

1. Tracer les fonctions $\sin(x)$, $\sin(2x)$ et $\sin(3x)$ sur le même graphique avec des styles différents (vous utiliserez différentes couleurs, styles de lignes et marqueurs).
2. Ajouter une légende pour identifier chaque courbe.

Exercice n° 7 : Visualisation d'une simulation physique

L'évolution de température et de pression dans un réacteur est simulée avec les équations suivantes :

$$T = 300 + 10 \sin(0.1t)$$

et

$$P = 100 + 5 \cos(0.1t)$$

Nous souhaitons visualiser les variations de T (°C) et P (en *kPa*) sur une échelle de temps t allant de 0 à 100 secondes avec 500 acquisitions.

Créez le script « `ex07_sim_reacteur.py` » dans le dossier « `src` ». Celui-ci doit :

1. Tracer les courbes T et P sur deux sous-figures partageant le même axe de temps.
2. Personnaliser le graphique avec des titres, légendes, et grilles.

Exercice n° 8 : Ajout d'annotations sur une figure

Créez le script « `ex08_annotations.py` » dans le dossier « `src` ». Celui-ci doit :

1. Tracer la courbe représentant la fonction sigmoïde donnée par la formule :

$$f(x) = \frac{1}{(1 + e^{-x})}$$

pour des valeurs de x allant de -10 à 10.

2. Ajouter une annotation au point d'inflexion (càd. le point où la sigmoïde change de concavité).
3. Utiliser une flèche pour pointer vers ce point sur la courbe.

Modifiez le script de manière à sauvegarder la figure dans le dossier « data/out » aux formats « PNG » et « PDF », et avec une résolution de 300 DPI.

Exercice n° 9 : Multi-capteurs synthétiques

Créez le script « ex09_extraction_multi_capteurs.py » dans le dossier « src ». Celui-ci doit :

1. Générer t sur l'intervalle $[0, 300]$ avec 601 points.
2. Simuler :

$$T = 20 + 2 \sin(2\pi t/120) + b_1$$

$$P = 101 + 0.8 \cos(2\pi t/90) + 0.02T + b_2$$

$$H = 50 + 5 \sin(2\pi t/150 + \varphi) - 0.1T + b_3$$

où b_1, b_2, b_3 représentent des bruits qui suivent une loi normale (écart-type raisonnable) et φ est fixé.

3. Tracer une figure avec trois sous-figures partageant l'axe du temps, ayant des unités correctes, et avec titres et légendes.
4. Sauvegarder la figure dans le dossier « data/out » avec un format « PNG » et une résolution de 600 DPI.

Exercice n° 10 : Matrice de corrélation et carte thermique

En GEII, l'analyse de corrélation entre différentes variables est fréquente. Cet exercice propose de visualiser une matrice de corrélation sous forme d'une carte thermique (heatmap).

Créez le script « ex10_carte_thermique.py » dans le dossier « src ». Celui-ci doit :

1. Générer quatre séries de longueur $N = 500$ avec NumPy. Inspirez-vous de l'exemple ci-dessous :

```
import numpy as np

rng = np.random.default_rng(17) # générateur déterministe
t = np.linspace(0, 100, 500)
temp = 20 + 2 * np.sin(2*np.pi*t/50) + rng.normal(0, 0.5, size=t.shape)
pres = 101 + 0.5 * np.cos(2*np.pi*t/40) + 0.05 * temp + rng.normal(0, 0.2, size=t.shape)
humid = 50 + 5 * np.sin(2*np.pi*t/70 + np.pi/6) - 0.1 * temp + rng.normal(0, 0.5, size=t.shape)
vent = 10 + 0.3 * pres + rng.normal(0, 1.0, size=t.shape)
```

Ici, les séries sont artificiellement corrélées (par exemple, la pression dépend un peu de la température, le vent dépend de la pression).

2. Calculer la matrice de corrélation entre ces séries de données. Pour ce faire, vous grouperez l'ensemble des données à l'aide de la fonction « `vstack()` » de Numpy, puis déterminerez la matrice de corrélation à l'aide de la fonction « `corrcoef()` » de Numpy.
1. Représenter cette matrice de corrélation sous forme d'une carte thermique avec une barre de couleur pour indiquer l'intensité de la corrélation (vous utiliserez la gamme de couleurs « `coolwarm` »).
2. Quelles sont les caractéristiques les plus corrélées ?

Exercice n° 11 : Ajout de formes sur une figure

Créez le script « ex11_formes.py » dans le dossier « src ». Celui-ci doit :

1. Créer une figure avec deux zones de tracé superposées (deux lignes, une colonne) de manière à y afficher deux courbes différentes.
2. Sur la première zone de tracé, afficher la courbe de la fonction sinus ($y = \sin(x)$).
3. Sur la deuxième zone de tracé, afficher la courbe de la fonction cosinus ($y = \cos(x)$).
4. Sur la courbe sinus, entourer avec un rectangle la zone autour du point où $x = \pi/2$.

5. Sur la courbe cosinus, ajouter un cercle centré autour du point où $x = \pi$.
6. Ajouter un titre pour chaque graphique, des étiquettes pour les axes, ainsi qu'une légende pour les courbes tracées.

Résumé

Au cours de cette séance, vous avez appris à :

- créer différents types de graphiques ;
- afficher des images ;
- personnaliser vos graphiques pour les rendre plus clairs et informatifs ;
- créer des figures multiples et organiser vos graphiques avec des sous-figures ;
- enregistrer vos figures pour une utilisation ultérieure.

Ce TP est une ressource que vous pourrez consulter à nouveau lors de vos futurs travaux pratiques. N'hésitez pas à revenir dessus lorsqu'il vous faudra visualiser des données et présenter des résultats.

Ressources

- Documentation officielle de Matplotlib : <https://matplotlib.org/stable/contents.html>
- Tutoriels Matplotlib : <https://matplotlib.org/stable/tutorials/index.html>
- Galerie d'exemples : <https://matplotlib.org/stable/gallery/index.html>
- Guide sur les gammes de couleurs : <https://matplotlib.org/stable/tutorials/colors/colormaps.html>
- NumPy : <https://numpy.org/doc/>
- Scikit-image : <https://scikit-image.org/>