Université Sultan Moulay Slimane Faculté Polydisciplinaire – Beni Mellal

Filière : LICENCE D'EXCELLENCE

Spécialité : IOT-R

Module : Python

Compte Rendu de
Ex _ 5 _ TP _ 5 _ Python

Réalisé par :

MOHAMED AMRAOUI

Encadré par :

Pr. MOUNCIF

Année universitaire : 2024-2025

I- Introduction :

Ce rapport présente les travaux pratiques réalisés dans le cadre du module Python, portant sur l'utilisation des bibliothèques NumPy et Matplotlib pour des applications en analyse de données, traitement du signal et simulations probabilistes.

Ce travail visait à :

- ✓ Générer et analyser des données aléatoires via une matrice de covariance,
- ✓ Appliquer la transformée de Fourier pour étudier un signal sinusoïdal
- ✓ Simuler des lancers de dés et visualiser la distribution des résultats.

Ces manipulations illustrent l'importance de Python dans les domaines scientifiques, notamment pour la modélisation et l'analyse statistique.

II- Partie pratique :

1- Matrice de Covariance :

a- Objectif :

Étudier les relations linéaires entre trois variables aléatoires générées.

b- Méthodologie :

- Création d'un tableau de dimensions (100, 3) avec des valeurs aléatoires uniformes (np.random.rand),
- Calcul de la matrice de covariance (np.cov), où chaque élément reflète la variance (diagonale) ou la covariance (hors diagonale) entre deux colonnes.

c- Code Python :

```
import numpy as np
import matplotlib.pyplot as plt
# 1. Tableau de données et matrice de covariance
print("1. Matrice de covariance:")
data = np.random.rand(100, 3)  # 100 lignes, 3 colonnes de valeurs aléatoires
cov_matrix = np.cov(data, rowvar=False)  # Calcul de la matrice de covariance
print(cov_matrix)
```

d- Résultats :

```
1. Matrice de covariance:
[[ 0.09329433 -0.00086451 -0.00369123]
 [-0.00086451  0.08050172  0.00925842]
 [-0.00369123  0.00925842  0.08787895]]
```

e- Interprétation :

- Les valeurs diagonales (~0.09) indiquent une variance similaire pour chaque variable,

- Les covariances proches de zéro suggèrent une faible corrélation linéaire entre les colonnes (attendu pour des données aléatoires).

2- *Transformée de Fourier :*

a- *Objectif :*

Analyser un signal sinusoïdal dans le domaine fréquentiel.

b- *Méthodologie :*

- Génération d'un signal sinusoïdal de fréquence 5 Hz (np.sin),
- Application de la transformée de Fourier (np.fft.fft) pour obtenir son spectre,
- Visualisation de l'amplitude en fonction des fréquences (plt.plot).

c- *Code Python :*

Application de la transformation de Fourier sur un tableau de données sinusoïdales :

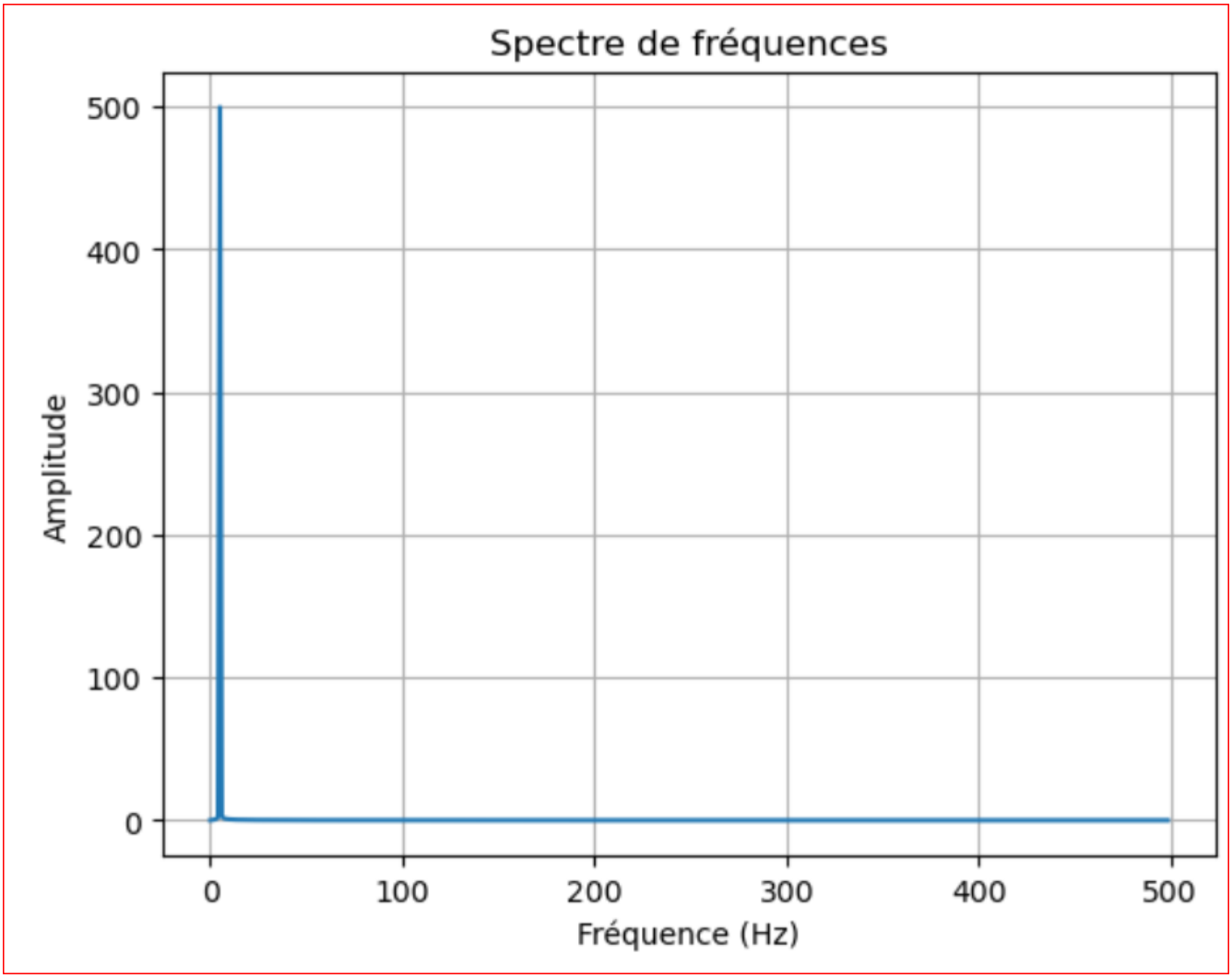
```
# 2. Transformation de Fourier sur des données sinusoïdales
print("\n2. Transformation de Fourier:")
# Création d'un signal sinusoïdal
t = np.linspace(0, 1, 1000)
freq = 5  # Fréquence du signal
signal = np.sin(2 * np.pi * freq * t)
# Transformation de Fourier
fourier = np.fft.fft(signal)
frequencies = np.fft.fftfreq(len(signal), d=t[1]-t[0])
```

Affichage du spectre de fréquences :

```
plt.figure()
plt.plot(frequencies[:len(frequencies)//2], np.abs(fourier[:len(fourier)//2]))
plt.title("Spectre de fréquences")
plt.xlabel("Fréquence (Hz)")
plt.ylabel("Amplitude")
plt.grid()
plt.show()
```

d- *Résultats :*

- ***Spectre obtenu :*** Un pic à 5 Hz (cf. figure), confirmant la fréquence du signal généré,
- ***Bruit nul :*** L'amplitude est nulle ailleurs, car le signal est parfaitement périodique.



3- *Simulation de lancers de dés et histogramme :*

a- *Objectif :*

Étudier la distribution de la somme de deux dés (loi discrète).

b- *Méthodologie :*

- Simulation de 1000 lancers de deux dés (np.random.randint),
- Calcul des sommes et génération d'un histogramme (plt.hist).

c- *Code Python :*

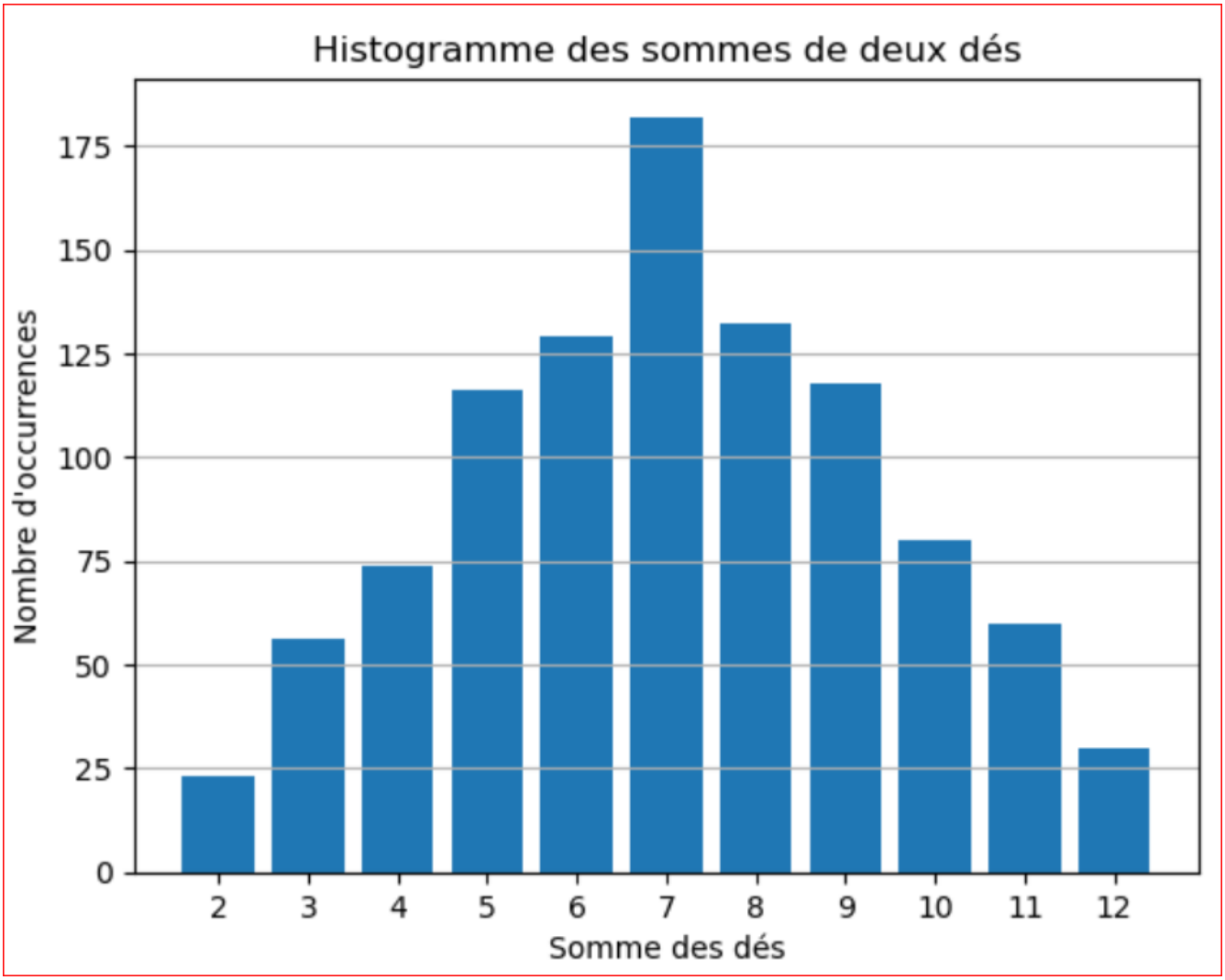
```
print("\n3. Histogramme des sommes de dés:")
# Simulation de 1000 lancers de deux dés
dice1 = np.random.randint(1, 7, 1000)
dice2 = np.random.randint(1, 7, 1000)
sums = dice1 + dice2
```

Affichage de l'histogramme :

```
plt.figure()
plt.hist(sums, bins=range(2, 14), align='left', rwidth=0.8)
plt.title("Histogramme des sommes de deux dés")
plt.xlabel("Somme des dés")
plt.ylabel("Nombre d'occurrences")
plt.xticks(range(2, 13))
plt.grid(axis='y')
plt.show()
```

d- *Résultats :*

- ***Histogramme :*** Distribution en forme de triangle centrée sur 7 (cf. figure).
- ***Probabilités :***
  - ✓ Somme la plus probable : 7 (16.7% théorique),
  - ✓ Sommes extrêmes (2 et 12) moins fréquentes (2.8% chacune).



4- *Conclusion :*

a- *Mise en évidence :*

Ce compte rendu a permis d'explorer trois aspects fondamentaux de la programmation scientifique avec Python, mettant en lumière la puissance des bibliothèques **NumPy** et **Matplotlib** pour résoudre des problèmes concrets en analyse de données, traitement du signal et simulations probabilistes.

b- Résultats obtenus :

Matrice de Covariance :
La génération d'un tableau de données aléatoires et le calcul de sa matrice de covariance ont démontré comment quantifier les relations entre variables. Les résultats ont confirmé l'indépendance attendue entre des variables aléatoires uniformes, avec des covariances proches de zéro.

Transformée de Fourier :
L'analyse d'un signal sinusoïdal à l'aide de la transformée de Fourier a permis d'identifier clairement sa fréquence fondamentale (5 Hz) dans le domaine spectral. Cette manipulation illustre l'utilité de cet outil pour l'étude des signaux périodiques.

Simulation de Lancers de Dés :
La simulation de lancers de dés et la visualisation de l'histogramme des sommes ont validé empiriquement les probabilités théoriques, montrant une distribution symétrique centrée sur la valeur 7, conforme aux attentes statistiques.

Maîtrise des Outils : Ce TP a renforcé la compréhension de NumPy pour les calculs numériques et de Matplotlib pour la visualisation.

Applications Pratiques : Les exercices couvrent des applications variées, de l'analyse statistique au traitement du signal, en passant par les simulations aléatoires.

Approche Scientifique : La confrontation entre résultats théoriques et simulations a souligné l'importance de la validation expérimentale en science des données.

c- Perspectives :

Pour approfondir ces concepts, les pistes suivantes pourraient être explorées :

Analyse de Données Complexes : Utiliser des jeux de données réels pour des analyses multivariées plus poussées.

Signaux Bruités *:* Étudier l'impact du bruit sur la transformée de Fourier et les méthodes de filtrage.

Simulations Avancées : Étendre les simulations à des systèmes plus complexes, comme les marches aléatoires ou les processus stochastiques.

Annexes :

```
import numpy as np
import matplotlib.pyplot as plt
# 1. Tableau de données et matrice de covariance
print("1. Matrice de covariance:")
data = np.random.rand(100, 3)  # 100 lignes, 3 colonnes de valeurs aléatoires
cov_matrix = np.cov(data, rowvar=False)  # Calcul de la matrice de covariance
print(cov_matrix)
# 2. Transformation de Fourier sur des données sinusoïdales
print("\n2. Transformation de Fourier:")
# Création d'un signal sinusoïdal
t = np.linspace(0, 1, 1000)
freq = 5  # Fréquence du signal
signal = np.sin(2 * np.pi * freq * t)
# Transformation de Fourier
fourier = np.fft.fft(signal)
frequencies = np.fft.fftfreq(len(signal), d=t[1]-t[0])
# Affichage du spectre de fréquences
plt.figure()
plt.plot(frequencies[:len(frequencies)//2], np.abs(fourier[:len(fourier)//2]))
plt.title("Spectre de fréquences")
plt.xlabel("Fréquence (Hz)")
plt.ylabel("Amplitude")
plt.grid()
plt.show()
# 3. Simulation de lancers de dés et histogramme
print("\n3. Histogramme des sommes de dés:")
# Simulation de 1000 lancers de deux dés
dice1 = np.random.randint(1, 7, 1000)
dice2 = np.random.randint(1, 7, 1000)
sums = dice1 + dice2
# Affichage de l'histogramme
plt.figure()
plt.hist(sums, bins=range(2, 14), align='left', rwidth=0.8)
plt.title("Histogramme des sommes de deux dés")
plt.xlabel("Somme des dés")
plt.ylabel("Nombre d'occurrences")
plt.xticks(range(2, 13))
plt.grid(axis='y')
plt.show()
```