ONIP-1 / Bloc 1 / Maitriser les concepts de base de Python pour la Science

Séance 2

Exercice 1 / Listes

Notions : *Data Types and Display - Lists*

1. Générer une liste ***L_temps*** de 101 points régulièrement répartis entre 0 et 1 s.
2. Afficher la taille de cette liste.
3. Afficher le dernier élément de la liste.
4. Quelle est la période d'échantillonnage ?

```python
N = 101
T = 1

L_temps = [i * (T / (N-1)) for i in range(N)]
print(L_temps)

print(f'size of the list = {len(L_temps)}')

# Display the last element
print(L_temps[-1])

print(f'Sampling period = {L_temps[1] - L_temps[0]} s')
```

Exercice 2 / Temps d'exécution et fonction

Notions : *Functions* | *Execution Time*

1. Créer une fonction ***get_list*** qui génère une liste de ***N*** nombres réels régulièrement répartis entre une valeur ***start*** et une valeur ***stop***. Les éléments ***N***, ***start*** et ***stop*** seront des paramètres de cette fonction. Le paramètre ***start*** aura une valeur par défaut de 0.
2. Mesurer le temps d'exécution de la création d'une liste ***L_test*** de 10.000 points répartis entre 0 et 1.

```python
import timeit

def get_list(number, stop, start=0):
    T = stop - start
    return [i * (T / (number-1)) for i in range(number)]


execution_time = timeit.timeit(lambda: get_list(10000, 1),
        globals=globals(), number=1)
print(f"Execution time: {execution_time} seconds")
```

Temps estimé sur un processeur Intel i5 avec 8Go de RAM sous Windows 10 : 1.6 ms

Exercice 3 / Génération d'un vecteur avec Numpy

Notions : *Functions* | *Numpy Basics - Using Vectors* | *Execution Time*

1. Importer la bibliothèque **Numpy**.
2. A l'aide de la bibliothèque **Numpy**, générer un vecteur ***V_test*** de 10.000 nombres réels répartis entre 0 et 1.
3. Afficher la taille de ce vecteur.
4. Afficher le dernier élément de ce vecteur.
5. Mesurer le temps d'exécution de la génération d'un vecteur de 10.000 nombres réels répartis entre 0 et 1.
6. Comparer à la mesure de l'exercice 2. Que concluez-vous ?

```
import numpy as np
import timeit

def get_vector(number, stop, start=0):
    return np.linspace(start, stop, number)

V_test = get_vector(10000, 1)

print(f'size of the array = {V_test.shape}')

print(f'last element = {V_test[-1]}')


execution_time = timeit.timeit(lambda: get_vector(10000, 1),
        globals=globals(), number=100)
print(f"Execution time: {execution_time} seconds")
```

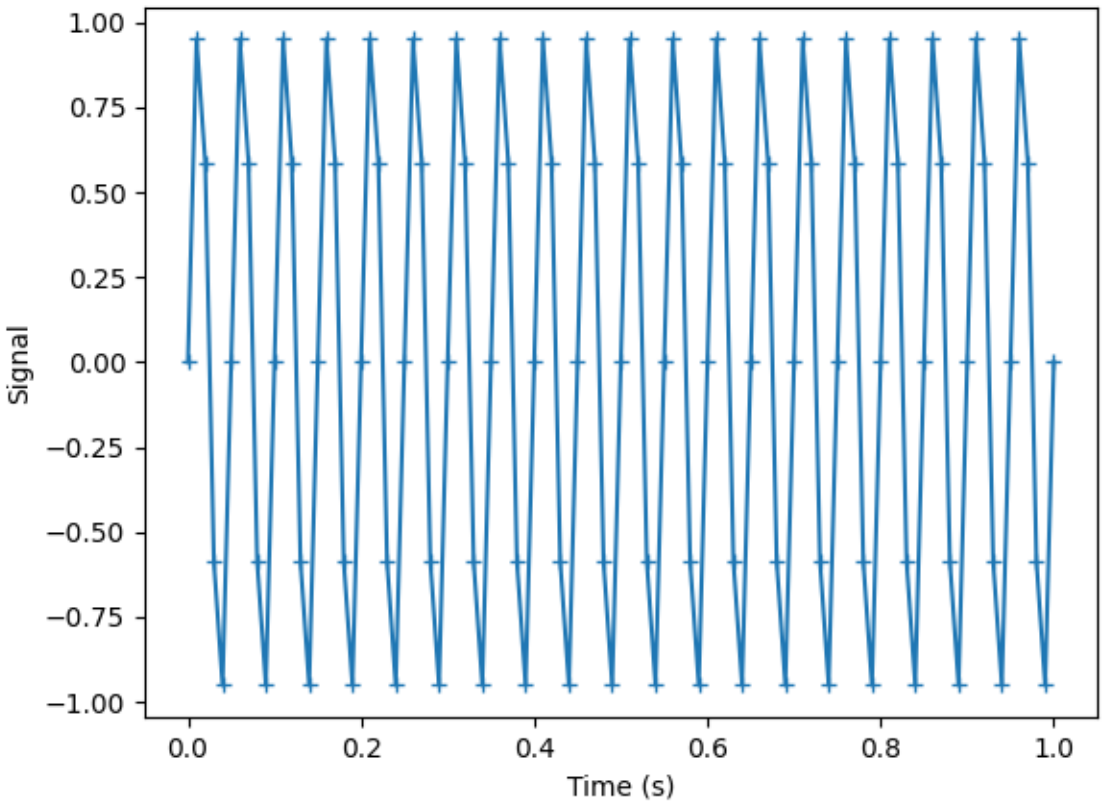
Temps estimé sur un processeur Intel i5 avec 8Go de RAM sous Windows 10 : 60 us

Exercice 4 / Génération de signaux et affichage

Notions : *Functions* | *Numpy Basics - Using Vectors* | *Matplotlib Basics - Scientist figure*

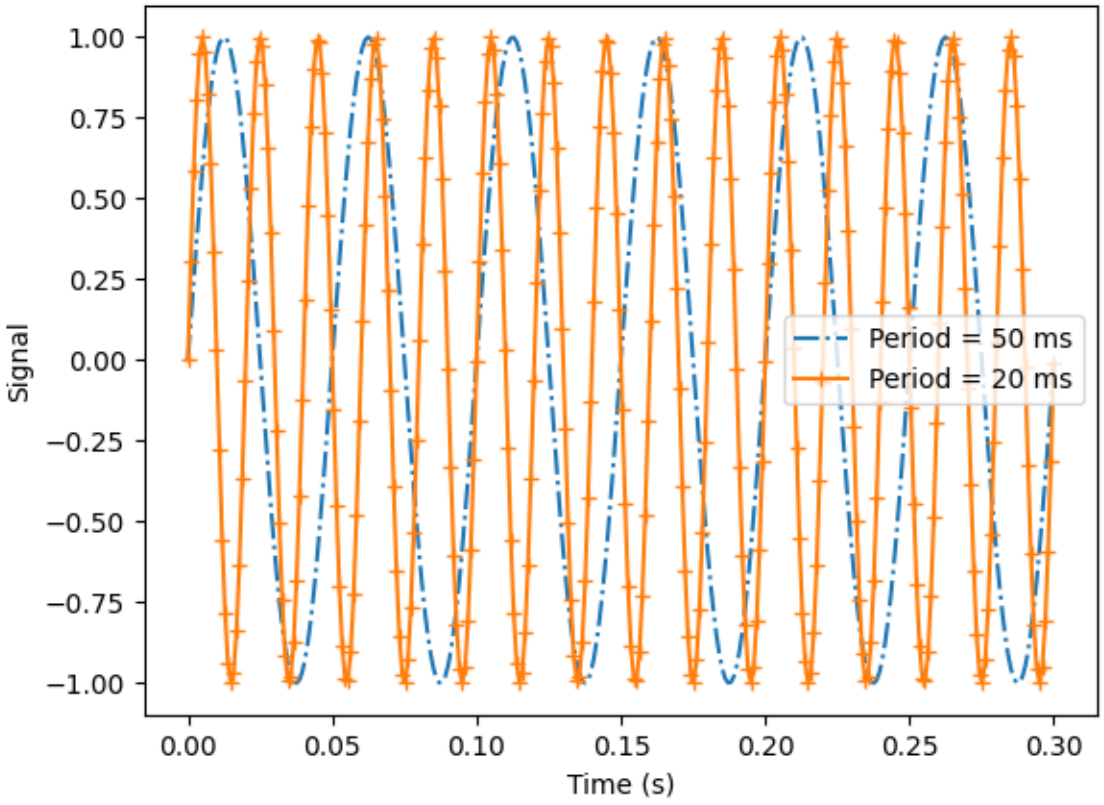
1. Créer un vecteur ***temps*** de 101 points régulièrement répartis entre 0 et 1 s. Quelle est la période d'échantillonnage ?
2. Importer la bibliothèque **matplotlib.pyplot**.
3. Tracer une sinusoïde de période 50 ms en rouge. Ajouter un titre, des axes et une légende au graphique.
4. Que pensez-vous du résultat ? Améliorer le résultat.
5. Tracer sur le même graphique une sinusoïde de période 20 ms en bleu.
6. Faire un zoom pour n'afficher que quelques périodes des deux sinusoïdes.
7. Faire une fonction qui prend comme argument la période de la sinusoïde, tester-la avec une période de 30 ms.

```
1  import numpy as np
2  from matplotlib import pyplot as plt
3  import timeit
4
5  def sine_wave(amplitude, frequency, time):
6      return amplitude * np.sin(2*np.pi*frequency*time)
7
8  temps = np.linspace(0, 1, 101)
9
10 period1 = 50e-3
11 freq1 = 1/period1
12 sine1 = sine_wave(1, freq1, temps)
13
14 plt.figure()
15 plt.plot(temps, sine1, '-+')
16 plt.show()
```



Amélioration : on change le pas d'échantillonnage (plus de points ici)

```
Ts = 0.99e-3
temps2 = np.arange(0, 1, Ts)

period1 = 50e-3
freq1 = 1/period1
sine1 = sine_wave(1, freq1, temps2)

plt.figure()
plt.plot(temps2, sine1, '-.')
plt.xlabel("Time (s)")
plt.ylabel("Signal")
plt.show()
```




```
# Zoom
N = len(temps2) // 4
plt.figure()
plt.plot(temps2[:N], sine1[:N], '-.', label="Period = 50 ms")
plt.plot(temps2[:N], sine2[:N], '-+', label="Period = 20 ms")
plt.show()
```

Exercice 5 / Fonction porte

Notions : *Numpy Basics - Using Vectors* | *Numpy Basics - Conditionals on arrays*

1. Créer une fonction ***porte*** qui renvoie un vecteur de booléens à partir d'un vecteur ***v_in***. Ce vecteur sera **vrai** lorsque les valeurs de ***v_in*** sont comprises entre une valeur ***min*** et une valeur ***max*** et **faux** sinon. Le vecteur ***v_in*** ainsi que les valeurs ***min*** et ***max*** seront passés en paramètre.
2. Tester cette fonction.

```
1  import numpy as np
2
3  def rect_func(v, min, max):
4      if(min > max):
5          max, min = min, max
6      return ((v >= min) & (v <= max))
7
8  v_test = np.arange(1, 11, 1)
9  print(v_test)
10
11 v_rect = rect_func(v_test, 2, 5)
12 print(v_rect)
```

Exercice 6 / Matrices

Notions : *Numpy Basics - Using Vectors* | *Numpy Basics - Using Matrices*

Soit la matrice A suivante :

$$A = \begin{pmatrix} 4 & 6 & -2 & 3 \\ 2 & -1 & 0 & 1 \\ -7 & 0 & 1 & 12 \end{pmatrix}$$

1. Créer une matrice ***A*** avec NUMPY (*np.array*) et l'afficher.
2. Afficher le type de ***A*** et le type d'un des éléments de ***A*** . Afficher aussi la taille de la matrice.
3. Afficher la seconde colonne puis la troisième ligne.
4. Afficher la moyenne de tous les éléments, la moyenne par colonne, la moyenne par ligne.
5. Utiliser l'affichage formaté pour afficher : "la moyenne de tous les éléments vaut ... "
6. Modifier la matrice ***A*** pour que ses 2 premières lignes soient multipliées par 2 puis que sa dernière colonne soit divisée par 3. Quel est alors le type des éléments de ***A*** ?
7. Créer une seconde matrice ***B*** de 3 lignes par 4 colonnes ne contenant que des 2. Remplacer la première colonne de ***B*** par des 0.
8. Que représente ***A*B*** ? Comment faire un produit matriciel entre ***A*** et ***B*** ?
9. Créer une matrice ***CC*** qui ne conserve que les éléments supérieurs à 5 de la matrice ***A*B*** et force les autres à 0.

```
import numpy as np

matrix_A = np.array([[4, 6, -2, 3], [2, -1, 0, 1], [-7, 0, 1, 12]])

print(f'Type of A : {type(matrix_A)}')
print(f'Type of an element of A : {matrix_A.dtype}')
print(f'Shape of the matrix : {matrix_A.shape}')
```

Type of A : <class 'numpy.ndarray'>

Type of an element of A : int32

Shape of the matrix : (3, 4)

```
print(f'Second column = {matrix_A[:, 1]}')
print(f'Third row = {matrix_A[2, :]}')

print(f'Mean of all the elements = {matrix_A.mean()}')
print(f'Mean in column = {matrix_A.mean(axis=0)}')
print(f'Mean in row = {matrix_A.mean(axis=1)}')
```

```
# Modifiing A
matrix_A[0:2,:] = matrix_A[0:2,:]*2
matrix_A[:,-1] = matrix_A[:,-1]/3.0

print(matrix_A)
print(f'Type of an element of A : {matrix_A.dtype}')
```

Type of an element of A : int32

Il est possible de forcer le calcul en flottant en modifiant le type de la matrice A (utilisation de la fonction *astype* de Numpy).

```
# Creating B
matrix_B = np.ones((3,4))*2
matrix_B[:,0]=0

matrix_C = matrix_A * matrix_B
print(matrix_C)
matrix_D = matrix_A.dot(matrix_B.T)
print(matrix_D)
```

```
# Creating CC
matrix_CC = (matrix_C > 5)*matrix_C
print(matrix_CC)
```

Exercice BONUS / Résolution d'une équation du 2nd degré

Faire une fonction qui calcule les solutions d'une équation du second degré.

Travail à réaliser (suite)

Séance 3

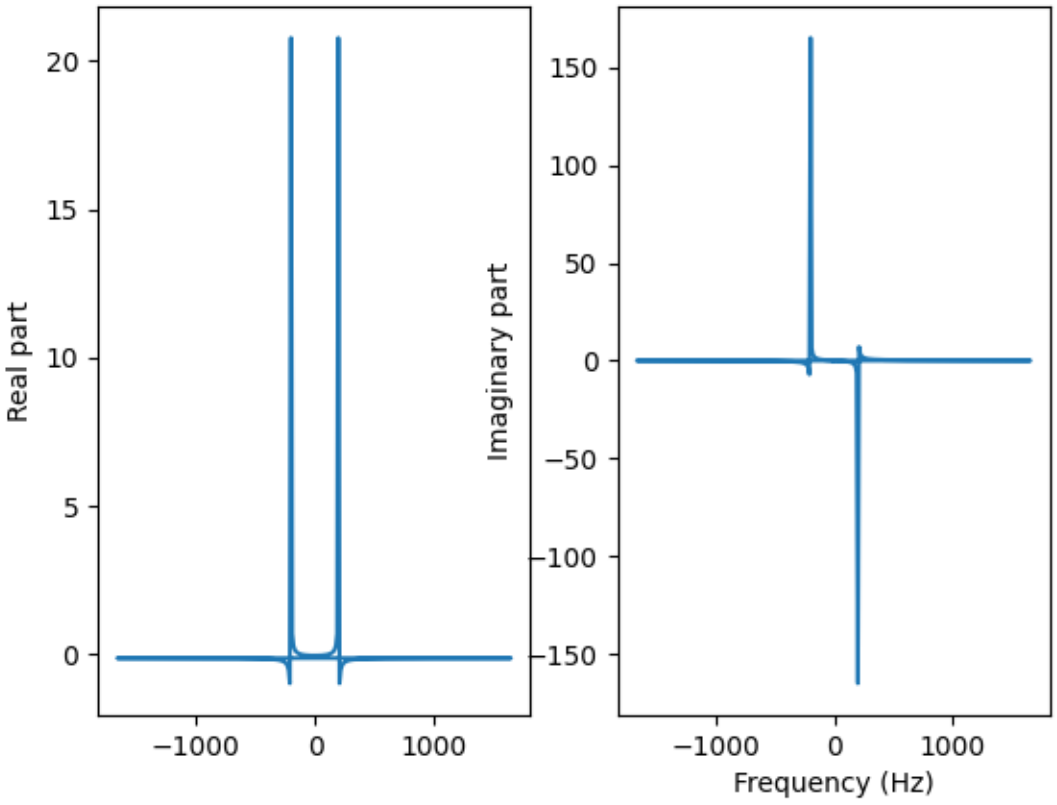
Exercice 1 / Transformée de Fourier

Notions : *Functions* | *Numpy Basics - Using Vectors* | *Matplotlib Basics - Scientist figure* | *FFT - Calculate the FFT of a signal*

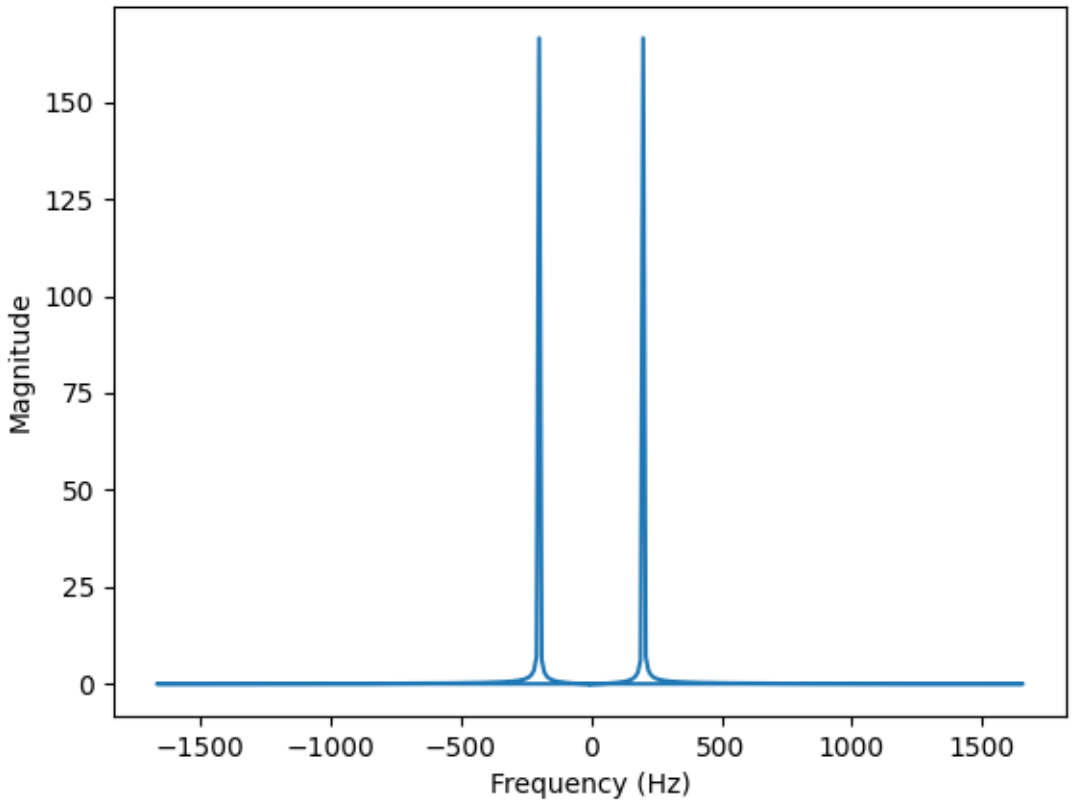
1. Créer une fonction permettant de générer une fonction sinusoïdale dont l'amplitude, la fréquence et le vecteur temps seront passés en paramètre.
2. Générer un signal sinusoïdal de fréquence 200 Hz avec une période d'échantillonnage de 300 μs, sur un intervalle de temps de 0.1 s.
3. Afficher ce signal.
4. Calculer la transformée de Fourier (TF) discrète du signal précédent, en utilisant l'algorithme FFT.
5. Afficher le résultat précédent sur un graphique (parties réelle et imaginaire). Ajouter un titre, des axes et une légende. Préciser le rôle de la fonction *np.fft.fftshift()*.
6. Reconstruire l'axe des fréquences pour compléter le graphique précédent.
7. Afficher à présent le module de la TF discrète du signal précédent. Est-ce le résultat voulu ?

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

def sine_wave(amplitude, frequency, time):
    return amplitude * np.sin(2*np.pi*frequency*time)

T_end = 0.1
Ts = 300e-6
freq = 200

time = np.arange(0, T_end, Ts)

sine = sine_wave(1, freq, time)

plt.figure()
plt.plot(time, sine, '-*')
plt.xlabel("Time (s)")
plt.ylabel("Signal")
plt.show()
```

```
# Process FFT
tf_sine = np.fft.fftshift(np.fft.fft(sine))
freq = np.fft.fftshift(np.fft.fftfreq(len(tf_sine), Ts))

plt.figure()
plt.subplot(1,2,1)
plt.plot(freq, np.real(tf_sine))
plt.ylabel("Real part")

plt.subplot(1,2,2)
plt.plot(freq, np.imag(tf_sine))
plt.ylabel("Imaginary part")
plt.xlabel("Frequency (Hz)")
plt.show()
```



```
1 # Display Magnitude
2 tf_sine_mag = np.abs(tf_sine)
3 plt.figure()
4 plt.plot(freq, tf_sine_mag)
5 plt.ylabel("Magnitude")
6 plt.xlabel("Frequency (Hz)")
7 plt.show()
```

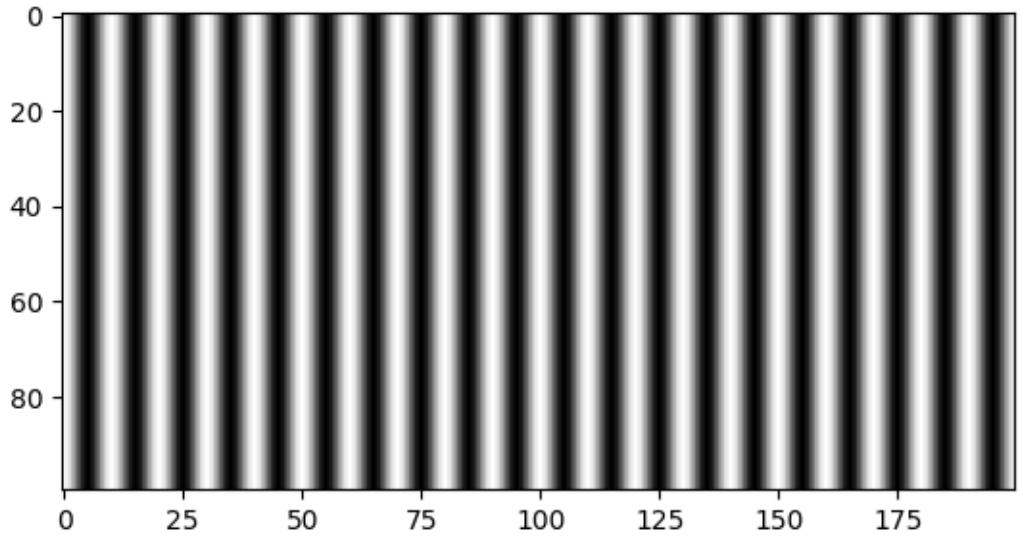


Exercice 2 / Génération de matrices 2D et utilisation d'un *meshgrid*

Notions : *Meshgrid and matrix calculation* | *Execution Time*

1. Générer une matrice à deux dimensions de taille 200 par 100, à l'aide de la fonction *np.ones()*.
2. A l'aide d'une double boucle, générer une matrice à 2 dimensions de largeur 200 et hauteur 100 (x compris entre 0 et 199 avec un pas de 1 et y compris entre 0 et 99 avec un pas de 1) dont les valeurs décrivent une fonction sinusoïdale de période 10 dans le sens vertical.
3. Visualiser le résultat à l'aide de la fonction *plt.imshow()*.
4. Mesurer le **temps d'exécution** de cette fonction.
5. Générer cette même matrice à l'aide d'un objet de type ***meshgrid*** (bibliothèque **Numpy**).
6. Visualiser le résultat à l'aide de la fonction *plt.imshow()*.
7. Mesurer le temps d'exécution de cette génération et comparer ce temps à celui obtenu précédemment. Qu'en concluez-vous ?

```
1  import timeit
2  import numpy as np
3  from matplotlib import pyplot as plt
4
5  def trame_sine(XX, YY, step, alpha=0):
6      alpha_rad = np.radians(alpha)
7      return 0.5*(1 + np.cos(2*np.pi*XX/step*np.cos(alpha_rad)
8          + 2*np.pi*YY/step*np.sin(alpha_rad)))
9
10 def gen_trame(width ,height, step, angle=0):
11     x = np.arange(0,width)
12     y = np.arange(0,height)
13     trame = np.zeros((height, width))
14
15     for i in range(len(y)):
16         for j in range(len(x)):
17             trame[i,j] = trame_sine(x[j], y[i], step=step, alpha = angle)
18     return trame
19
20 trame = gen_trame(200, 100, 10, 0)
21
22 plt.figure()
23 plt.imshow(trame, cmap='gray')
24 plt.show()
```




```python
# Meshgrid
def gen_trame_plus(width ,height, step, angle=0):
    x = np.arange(0,width)
    y = np.arange(0,height)
    XX, YY = np.meshgrid(x, y)
    return trame_sine(XX, YY, step=step, alpha = angle)

trame = gen_trame(200, 100, 10, 0)

plt.figure()
plt.imshow(trame, cmap='gray')
plt.show()

# Execution time comparison
execution_time = timeit.timeit(lambda: gen_trame(200, 100, 10, 0),
    globals=globals(), number=1)
print(f"Execution time: {execution_time} seconds")
execution_time = timeit.timeit(lambda: gen_trame_plus(200, 100, 10, 0),
    globals=globals(), number=1)
print(f"Execution time: {execution_time} seconds")
```

Temps estimé sur un processeur Intel i5 avec 8Go de RAM sous Windows 10 :

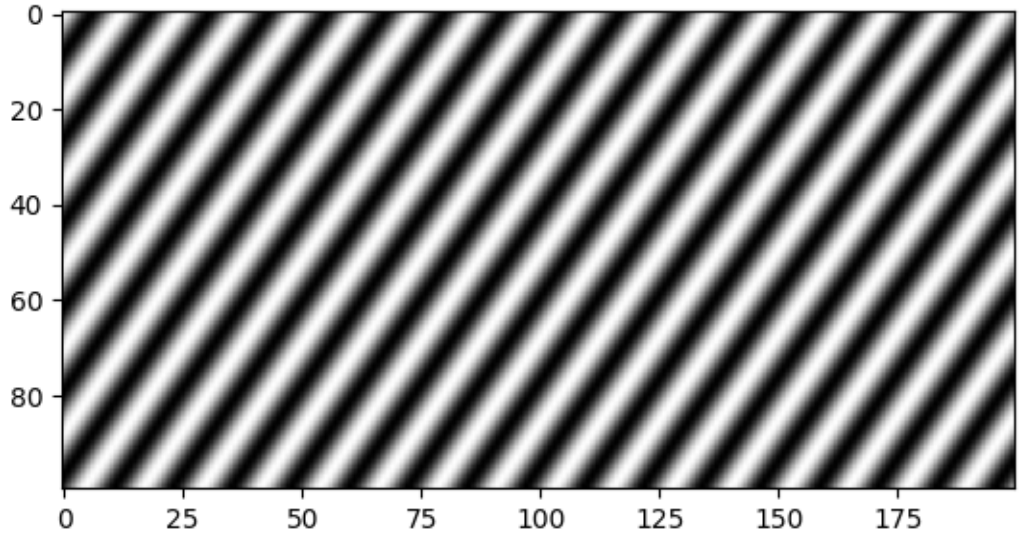
- Version boucle : 230 ms
- Version Meshgrid : 2 ms

Exercice 3 / Génération de matrices 2D - Tramage

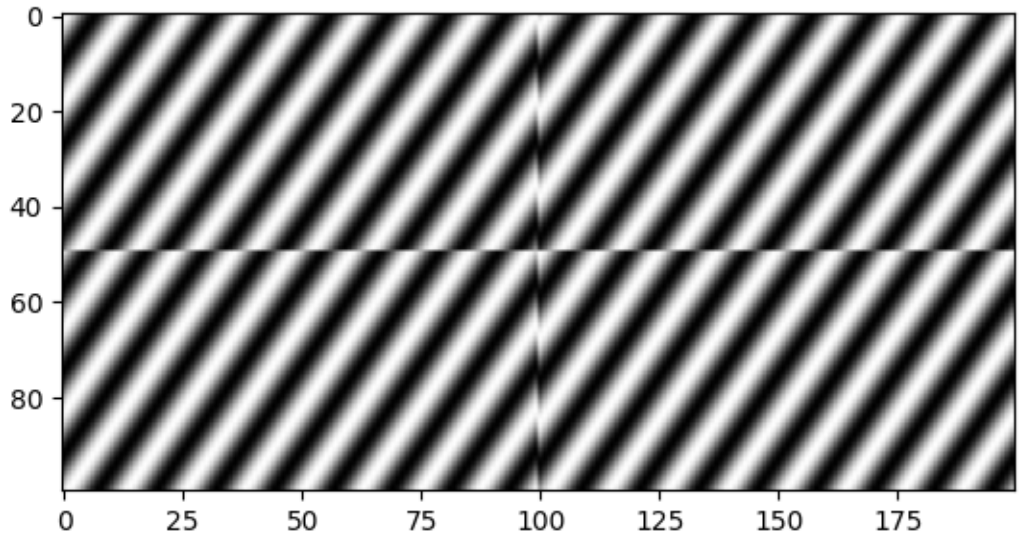
Notions : *Functions* | *Numpy Basics - Using Vectors* | *Meshgrid and matrix calculation* | FFT2D

1. Réaliser une fonction qui génère une trame en 2 dimensions sinusoïdale de taille ***M*** par ***N***, d'un pas spatial de ***step***, selon un angle ***alpha***.
2. Tester cette fonction et générer une trame de 200 par 100 selon un pas de 10.1 et un angle de 35 deg .
3. Utiliser la fonction ***np.fft.fftshift*** sur la matrice générée.
4. Faire la transformée de Fourier en 2D de la trame précédente et afficher le résultat.

```python
import numpy as np
from matplotlib import pyplot as plt

def trame_sine(XX, YY, step, alpha=0):
    alpha_rad = np.radians(alpha)
    return 0.5*(1 + np.cos(2*np.pi*XX/step*np.cos(alpha_rad)
    + 2*np.pi*YY/step*np.sin(alpha_rad)))

# Meshgrid
def gen_trame_plus(width ,height, step, angle=0):
    x = np.arange(0,width)
    y = np.arange(0,height)
    XX, YY = np.meshgrid(x, y)
    return trame_sine(XX, YY, step=step, alpha=angle)

trame = gen_trame_plus(200, 100, 10.1, 35)

plt.figure()
plt.imshow(trame, cmap='gray')
plt.show()
```



```
# Shift
trame_shifted = np.fft.fftshift(trame)

plt.figure()
plt.imshow(trame_shifted, cmap='gray')
plt.show()
```



```
# Process FFT and display
fft_trame = np.fft.fft2(trame)
fft_trame_shifted = np.fft.fftshift(fft_trame)
mag_fft_trame_shifted = np.abs(fft_trame_shifted)

plt.figure()
plt.imshow(np.log(1+mag_fft_trame_shifted), cmap='gray')
plt.show()
```