

TP : Représentation, Numérisation et Transformée de Fourier

EL BENANY Mohamed Mahmoud

1 TP1 : Représentation des signaux sous Python

1.1 Théorie

Les signaux peuvent être représentés de différentes manières :

- **Signal sinusoïdal** : $x(t) = A \sin(2\pi ft + \phi)$, où A est l'amplitude, f la fréquence, et ϕ la phase.
- **Signal amorti** : $x(t) = Ae^{-\alpha t} \sin(2\pi ft + \phi)$, où α est le coefficient d'amortissement.
- **Signal Porte** : Un signal rectangulaire défini par une amplitude constante sur un intervalle de temps donné.
- **Signal sonore** : Un signal complexe composé de plusieurs fréquences, souvent représenté par un fichier audio.

1.2 Pratique

Voici un exemple de code Python pour représenter ces signaux :

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 # Param tres
5 fs = 1000 # Fr quence d' chantillonage en Hz
6 t = np.linspace(0, 1, fs, endpoint=False) # Temps de 0 1
   seconde
7
8 # Signal sinuso dal
9 f_sin = 5 # Fr quence en Hz
10 x_sin = np.sin(2 * np.pi * f_sin * t)
11
12 # Signal amorti
13 alpha = 5 # Coefficient d'amortissement
14 x_amorti = np.exp(-alpha * t) * np.sin(2 * np.pi * f_sin * t)
15
16 # Signal Porte
17 x_porte = np.where((t >= 0.2) & (t <= 0.8), 1, 0) # Porte entre
   0.2 et 0.8 secondes
18
```

```

19 # Affichage des signaux
20 plt.figure(figsize=(12, 8))
21
22 plt.subplot(3, 1, 1)
23 plt.plot(t, x_sin)
24 plt.title("Signal sinuso dal")
25 plt.xlabel("Temps (s)")
26 plt.ylabel("Amplitude")
27
28 plt.subplot(3, 1, 2)
29 plt.plot(t, x_amorti)
30 plt.title("Signal amorti")
31 plt.xlabel("Temps (s)")
32 plt.ylabel("Amplitude")
33
34 plt.subplot(3, 1, 3)
35 plt.plot(t, x_porte)
36 plt.title("Signal Porte")
37 plt.xlabel("Temps (s)")
38 plt.ylabel("Amplitude")
39
40 plt.tight_layout()
41 plt.show()

```

2 TP2 : Numérisation des signaux

2.1 Théorie

La numérisation des signaux implique :

- **Repliement spectral (Aliasing)** : Si la fréquence d'échantillonnage f_s est inférieure à deux fois la fréquence maximale du signal, un repliement spectral se produit.
- **Représentation temps-fréquence** : La transformée de Fourier à court terme (STFT) permet de visualiser l'évolution des fréquences dans le temps.
- **Interpolation** : Reconstruction d'un signal à partir d'échantillons discrets.

2.2 Pratique

Voici un exemple de code Python pour illustrer ces concepts :

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 from scipy.signal import stft
4 from scipy.interpolate import interp1d
5
6 # Signal sinuso dal avec aliasing
7 fs = 50 # Fr quence d' chantillonnage en Hz
8 t = np.linspace(0, 1, fs, endpoint=False)
9 f_signal = 60 # Fr quence du signal en Hz

```

```

10 x_aliasing = np.sin(2 * np.pi * f_signal * t)
11
12 # Repr sentation temps-fr quence (STFT)
13 f, t_stft, Zxx = stft(x_aliasing, fs, nperseg=20)
14
15 # Interpolation
16 t_interp = np.linspace(0, 1, 1000) # Temps interpol
17 f_interp = interpfd(t, x_aliasing, kind='cubic') # Interpolation
    cubique
18 x_interp = f_interp(t_interp)
19
20 # Affichage des r sultats
21 plt.figure(figsize=(12, 8))
22
23 plt.subplot(3, 1, 1)
24 plt.plot(t, x_aliasing, 'o-')
25 plt.title("Signal avec aliasing")
26 plt.xlabel("Temps (s)")
27 plt.ylabel("Amplitude")
28
29 plt.subplot(3, 1, 2)
30 plt.pcolormesh(t_stft, f, np.abs(Zxx), shading='gouraud')
31 plt.title("Repr sentation temps-fr quence (STFT)")
32 plt.xlabel("Temps (s)")
33 plt.ylabel("Fr quence (Hz)")
34
35 plt.subplot(3, 1, 3)
36 plt.plot(t_interp, x_interp)
37 plt.title("Signal interpol ")
38 plt.xlabel("Temps (s)")
39 plt.ylabel("Amplitude")
40
41 plt.tight_layout()
42 plt.show()

```

3 TP3 : Applications sur la Transformée de Fourier

3.1 Théorie

La Transformée de Fourier permet de décomposer un signal en ses composantes fréquentielles. Elle est utilisée pour :

- Filtrage fréquentiel (passe-bas, passe-haut).
- Analyse spectrale (identification des fréquences dominantes).
- Reconstruction de signaux à partir de leur spectre.

3.2 Pratique

Voici un exemple de code Python pour appliquer la Transformée de Fourier :

```

1 import numpy as np

```

```

2 import matplotlib.pyplot as plt
3
4 # Signal composé de plusieurs sinusoides
5 fs = 1000 # Fréquence d'échantillonnage en Hz
6 t = np.linspace(0, 1, fs, endpoint=False)
7 f1, f2, f3 = 50, 120, 300 # Fréquences en Hz
8 x = 0.7 * np.sin(2 * np.pi * f1 * t) + 0.5 * np.sin(2 * np.pi * f2
9     * t) + 0.3 * np.sin(2 * np.pi * f3 * t)
10
11 # Transformée de Fourier
12 X = np.fft.fft(x)
13 freqs = np.fft.fftfreq(len(t), 1/fs)
14
15 # Filtrage passe-bas
16 f_cutoff = 100 # Fréquence de coupure en Hz
17 X_filtre = X.copy()
18 X_filtre[np.abs(freqs) > f_cutoff] = 0 # Supprime les fréquences
19     > 100 Hz
20
21 # Transformée de Fourier inverse
22 x_filtre = np.fft.ifft(X_filtre).real
23
24 # Affichage des résultats
25 plt.figure(figsize=(12, 8))
26
27 plt.subplot(3, 1, 1)
28 plt.plot(t, x)
29 plt.title("Signal original")
30 plt.xlabel("Temps (s)")
31 plt.ylabel("Amplitude")
32
33 plt.subplot(3, 1, 2)
34 plt.plot(freqs[:len(freqs)//2], np.abs(X[:len(freqs)//2]))
35 plt.title("Spectre du signal")
36 plt.xlabel("Fréquence (Hz)")
37 plt.ylabel("Amplitude")
38
39 plt.subplot(3, 1, 3)
40 plt.plot(t, x_filtre)
41 plt.title("Signal filtré (passe-bas 100 Hz)")
42 plt.xlabel("Temps (s)")
43 plt.ylabel("Amplitude")
44
45 plt.tight_layout()
46 plt.show()

```

4 Conclusion

Ces TP permettent aux étudiants de :

- Représenter des signaux simples et complexes sous Python.
- Comprendre la numérisation des signaux, y compris les effets d'aliasing et l'interpolation.
- Appliquer la Transformée de Fourier pour analyser et filtrer des signaux.