TD 2
Numpy – Signal échantillonné
Transformée de Fourier

Pour l'ensemble des TD suivants, on préfèrera l'utilisation de programmes plutôt que l'éditeur python lui-même.

Exercice 1. Les bases avec numpy

1. Importer le module `numpy` en le renommant `np`. Créer un vecteur `n` contenant des entiers allant de 0 à 99. Vérifier le type et les dimensions de `n`, ainsi que le type des éléments du vecteur.

2. Créer une nouveau vecteur `t` égal à `n`. Modifier `t` afin qu'il contienne des valeurs allant de 0 à $\simeq 1$ sec. par pas de 10 ms. Vérifier que `n` n'a pas été modifié par l'opération. Proposer une commande permettant de créer `t` directement sans passer par `n`.

3. Déterminer l'index de `t` correspondant à la valeur de 50 ms ? On pourra utiliser la méthode `where` de `numpy`

4. Un signal sinusoidal est un signal continu défini par l'équation:

$$x(t) = a\sin(2\pi f_0 t + \phi)$$

où a est l'amplitude, f_0 la fréquence de la sinusoide et ϕ la phase.

Avec des outils numériques, le temps n'est pas continu mais discret. Pour discrétiser un signal continu, il suffit de prendre un échantillon de ce signal continu toutes les T_e secondes. T_e est appelé période d'échantillonnage, la fréquence d'échantillonnage est définie par $F_e = 1/T_e$.

Générer un signal sinusoidal de paramètres $a = 1.5$, $\phi = 0$ et $f_0 = 10$ Hz. Le signal sera échantillonné à la fréquence $F_e = 100$ Hz. On limitera le signal à 1 sec.

5. Pour faire des graphiques, et tracer $x(t)$, on utilisera la famille de fonction `pyplot` du module `matplotlib`

```
import matplotlib.pyplot as plt
plt.plot(t,x)
plt.xlabel('temps (s)')
plt.ylabel('amplitude (UA)')
plt.show()
```

Tracer le signal sinusoidal entre 0 et 1 s. Mesurer la période réelle et la comparer avec la période du signal (T_0). Maintenant, tracer le signal entre 50 ms et 150 ms. Vous devez observer des lignes brisées. Pourquoi ? (on aurait pu aussi utiliser la fonction `plt.stem(t,x)`).

6. Augmenter la fréquence d'échantillonnage et afficher le signal entre 50 ms et 150 ms. Qu'observez-vous ? Intuitivement, nous pouvons penser qu'une bonne restitution du signal nécessite une fréquence d'échantillonnage élevée.

7. Fixer la fréquence d'échantillonnage $F_e = 1000$ Hz. Et compléter le tableau suivant ($T_0 = 1/f_0$):

N^o	F_e	f_0	Période T_0 réelle	Période mesurée
1	1000 Hz	10 Hz		
2	1000 Hz	300 Hz		
3	1000 Hz	500 Hz		
4	1000 Hz	990 Hz		

   Que remarquez-vous ?

   Lorsque $F_e = 2f_0$, l'échantillonnage n'est plus capable de capturer la fréquence de la sinusoïde. (https://www.youtube.com/watch?v=jQDjJRYmeWg)

   Lorsque $F_e < 2f_0$, on parle de **repliement du spectre (aliasing)**. (https://www.youtube.com/watch?v=jHS9JGkEOmA)

8. Afin de lire et d'écrire des fichiers WAV, on peut utiliser les fonctions `wavfile` du module python `scipy`: `from scipy.io import wavfile`
   `wavfile.write(data, filename, rate)` écrit une matrice numpy `data` dans un fichier wav `filename` à la fréquence d'échantillonnage `rate`.

   Ecrire 3 fichiers WAV correspondants aux paramètres des signaux N^{o}2, 3 et 4. Les écouter.

Remarque importante:
Les résultats de ces observations sont fondamentaux pour le traitement du signal. Le théorème de Shannon formalise la condition permettant d'échantillonner correctement un signal.

$$F_e > 2f_{max}$$

La fréquence la plus élevée que l'oreille peut entendre est d'environ 20 kHz. C'est pour cela que la fréquence d'échantillonnage utilisée pour les supports audio numérique est généralement de 44.1 kHz.

Exercice 2. Décomposition en séries de Fourier

Fourier a démontré qu'il était possible de décomposer un signal périodique de fréquence fondamentale f_0 en une somme de plusieurs sinusoïdes de fréquence kf_0. Mathématiquement, n'importe quel signal périodique $s(t)$ peut donc s'exprimer sous la forme:

$$s(t) = a_0 + \sum_{k=1}^{\infty} a_k \cos(2\pi k f_0 t) + \sum_{k=1}^{\infty} b_k \sin(2\pi k f_0 t)$$

Le tableau suivant présente les premières valeurs des coefficients a_k et b_k pour plusieurs signaux périodiques (ces valeurs peuvent s'obtenir assez simplement à l'aide des sommes infinies).

Signal	a_k	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9
Carré	0	1	0	1/3	0	1/5	0	1/7	0	1/9
Triangulaire	0	1	0	-1/9	0	1/25	0	-1/49	0	1/81
Dent de Scie	0	1	-1/2	1/3	-1/4	1/5	-1/6	1/7	-1/8	1/9

1. Avec `numpy` synthétiser un signal carré, triangulaire et en dent de scie avec les fréquences fondamentales suivantes: $f_0 = 10$ Hz et $f_0 = 1000$ Hz. Les signaux seront échantillonnés à la fréquence $F_e = 8$ kHz sur une durée de 1 sec.
   Astuce: on pourra créer une matrice `numpy` contenant 10 colonnes. Chaque colonne correspond à un signal sinusoidal particulier (comme vu dans l'Ex. 1). $s(t)$ sera obtenu en faisant la somme des colonnes.

2. Tracer ces 6 signaux en fonction du temps.

3. Créer des fichiers WAV correspondant aux 6 situations précédentes. Les écouter. Qu'en pensez-vous ?

La décomposition en série de Fourier peut être généralisée au cas des signaux non-périodiques. Dans ce cas, le signal est décomposé en une somme d'exponentielles complexes:

$$x(t) = \int_{-\infty}^{+\infty} X(f) e^{2j\pi ft} df$$

où $X(f)$ correspond à la transformée de Fourier du signal $x(t)$.

Exercice 3. Transformée de Fourier

La transformée de Fourier est très utilisée pour l'analyse des signaux (contenu fréquentiel, spectrogramme, périodigramme, etc.). Cette transformée permet de mettre en valeur des éléments de notre signal difficilement visualisables dans le domaine temporel.

Mathématiquement, la transformée de Fourier d'un signal $x(t)$ s'exprime sous la forme

$$X(f) = \int_{-\infty}^{+\infty} x(t) e^{-2j\pi ft} dt$$

La plupart des langages de programmation intègrent des fonctionnalités pour le calcul de la transformée de Fourier via des algorithmes rapides. Ces algorithmes sont couramment nommés FFT (Fast Fourier Transform). Dans cette exercice, nous allons illustrer l'intérêt de cette transformée.

1. Créer un signal d'1 sec. carré de fréquence fondamentale $f_0 = 120$ Hz et de fréquence d'échantillonnage 8 kHz. Le tracer en fonction du temps.

2. La famille de fonctions `fft` de `numpy` permet de réaliser des opérations suivant les normes de calcul de la DCT (Discrete Cosine Transform). Parmi ces fonctions, la `fft` calcule la transformée de Fourier rapide d'un signal (à 1 dimension).
   ```
   from numpy.fft import fft
   X = fft(x)
   ```
   Calculer la `fft` du signal carré:

3. Observer quelques valeurs de la FFT. Que remarquez-vous ?

4. La FFT d'un signal contenant N échantillons contient elle-aussi N échantillons. Alors que les échantillons temporels de $x(t)$ sont espacés de T_e, les échantillons fréquentiels de $X(f)$ sont espacés de F_e. Quelle est la fréquence maximum ? Déterminer alors un vecteur correspondant aux fréquences.

5. Le module de la FFT est obtenu avec `np.abs(X)`, la puissance `np.abs(X)**2` et sa phase `np.angle(X)`. Tracer le module et la phase de X en fonction des fréquences.

6. Vous observez une série de raies. A quelle fréquence est la première raie ? Dans le cas où le signal est constitué d'une seule sinusoïde (et pas une somme comme dans la décomposition de Fourrier), qu'observerait-on ? Que se passe-t-il si vous modifiez la valeur de la fréquence fondamentale (par exemple à 50 Hz et à 3000 Hz)?