TP 2 : Signaux périodiques

Ce TP a pour objectif l'étude du traitement linéaire du signal. Des signaux en créneau seront en particulier étudiés.

Nous utiliserons la libraire graphique standard `matplotlib.pyplot`, le script commencera par importer celle-ci à l'aide de la ligne :

```
import matplotlib.pyplot as plt
```

1 Affichage du spectre d'un signal créneau

1.1 Préambule : tracé d'un segment

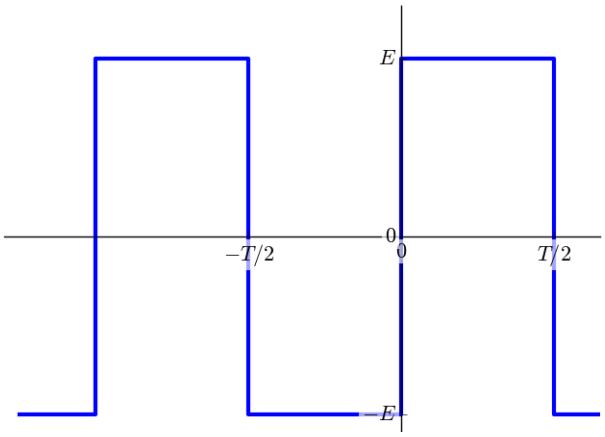
Question 1. En utilisant l'instruction `plt.plot` (qui fait partie du module `matplotlib.pyplot`), tracer un segment allant d'un point de coordonnées (1,2) à un autre point de coordonnées (3,6) dans le plan visible entre les abscisses 0 et 5 et entre les ordonnées 0 et 8.

Question 2. Changer l'épaisseur du trait pour qu'il fasse trois pixels.

1.2 Tracé du spectre d'un créneau symétrique impair

On rappelle la décomposition en série de Fourier d'un signal créneau symétrique f du temps t, de période T, de pulsation ω et d'amplitude E :

$$f(t) = \frac{4E}{\pi}\left(\sin(\omega t) + \frac{1}{3}\sin(3\omega t) + ... + \frac{1}{2p+1}\sin((2p+1)\omega t) + ...\right)$$



Le but est de représenter graphiquement le spectre du créneau sous forme d'un ensemble de barres, en portant en abscisse la pulsation de l'harmonique $(\omega, 3\omega, 5\omega, ...)$ et en ordonnée le coefficient du sinus correspondant $\left(\frac{4E}{\pi}, \frac{4E}{3\pi}, \frac{4E}{5\pi}, ...\right)$

Question 3. Créer une liste de 15 éléments `bnCarre` contenant la suite des valeurs des ordonnées du spectre pour $E = 1$ et $\omega = 1$

Question 4. Représenter graphiquement le spectre ainsi créé. On nommera *pulsation* l'abscisse, et *amplitude spectrale* l'ordonnée. On donnera à cette figure le titre *spectre d'un créneau symétrique.*

Question 5. Comment serait-il modifié si on ajoutait un *offset*, c'est-à-dire une composante continue ou encore une valeur moyenne positive au créneau symétrique ?

2 Reconstitution d'un chronogramme à partir d'un spectre donné

L'objectif de cette partie est de représenter un polynôme trigonométrique défini par son spectre. Dans une première sous-partie, nous traiterons ce problème à partir de deux listes contenant les coefficients de Fourier (a_k) et (b_k) ; dans une seconde partie, nous importerons ces coefficients à partir d'un fichier externe.

2.1 Reconstitution de la fonction

Une suite finie des coefficients $(a_k)_{0\leq k\leq n}$ et $(b_k)_{0\leq k\leq n}$ de Fourier définissent (de manière unique si l'on impose $b_0 = 0$) une fonction g, appelée polynôme trigonométrique, par la formule suivante :

$$g(t) = \sum_{k=0}^{k=n} (a_k \cos(2\pi kt) + b_k \sin(2\pi kt))$$

Question 6. Écrire une fonction `sommepartielle` d'arguments `A` et `B` et `t`, où A et B sont deux listes de même longueur, qui renvoie la valeur en t du polynôme trigonométrique g défini plus haut.

Question 7. Créer des listes `ae` et `be`, composées de 30 éléments, et définies comme suit :

```
ae = [0., 0., 0., ...,0.]
be = [0., 4/np.py, 0., 4/(3*np.py), 0., 4/(5*np.py), 0., ..., 0., 4/(29*np.py)]
```

Question 8. Tracer le polynôme trigonométrique associé aux listes `ae` et `be`.

2.2 Traitement d'un fichier externe

Le fichier `fourier.txt` contient sur les lignes successives les amplitudes spectrales en cosinus et en sinus, séparées par un point virgule, (colonne de droite) de la série de Fourier d'un polynôme trigonométrique. La première ligne correspond ainsi à la composante continue (fréquence nulle), la deuxième ligne à la fréquence fondamentale, les lignes suivantes aux harmoniques successives.

Question 9. Placer ce fichier sur votre répertoire de travail Python.

Ouvrir ce fichier en lecture, et récupérer son contenu (une chaîne de caractères) dans une variable `coeffstxt`.

Que fait la ligne de commande `lignes = coeffstxt.split('\n')` ?

Exploiter alors cette variable `ligne` pour créer deux listes `a` et `b` contenant les valeurs flottantes des amplitudes spectrales en cosinus et en sinus.

Question 10. À l'aide des questions précédentes, tracer le chronogramme correspondant aux listes `a` et `b` ainsi créées.

3 Réponse spectrale et temporelle d'un filtre linéaire

3.1 Nombres complexes

L'annexe rappelle les outils de base pour la manipulation des nombres complexes en Python.

Il a été vu en TD de physique un algorithme simple d'extraction de l'argument d'un complexe z :

1. Si la partie réelle est nulle :
   (a) Si la partie imaginaire est strictement positive, l'argument vaut $\frac{\pi}{2}$.
   (b) Si la partie imaginaire est strictement négative, l'argument vaut $-\frac{\pi}{2}$.
   (c) Si la partie imaginaire est nulle, l'argument n'est pas défini.
2. Si la partie réelle est strictement positive, l'argument vaut $\arctan\left(\frac{Im(z)}{Re(z)}\right)$.

3. Si la partie réelle est strictement négative :
   (a) Si la partie imaginaire est positive ou nulle, l'argument vaut $\arctan\left(\frac{Im(z)}{Re(z)}\right) + \pi$.
   (b) Si la partie imaginaire est strictement négative, l'argument vaut $\arctan\left(\frac{Im(z)}{Re(z)}\right) - \pi$.

Question 11. Définir une fonction `module` prenant en argument un nombre complexe z quelconque et retournant son module. Il existe par ailleurs une fonction toute faite dans Python (voir annexe).

Question 12. Définir une fonction `argument` prenant en argument un nombre complexe z quelconque et retournant son argument.

Question 13. Tester ces fonctions sur les trois nombres complexes : -3, $1+2j$, $7j$

3.2 Fonction de transfert

On prend l'exemple d'un filtre linéaire passe bas du premier ordre de gain statique unitaire.

Question 14. Rappeler la forme canonique de sa fonction de transfert en notant ω_C sa pulsation de coupure, choisie à la valeur numérique 1000 rad/s.

Question 15. Définir une fonction complexe `H` de la variable `omega` pour la fonction de transfert.

Question 16. Définir deux fonction `gdB` et `Phi` de la variable `omega` pour le gain en décibel et le déphasage de la sortie en fonction de l'entrée.

Question 17. Afficher enfin, sur quatre décades à partir de la pulsation 10 rad/s, les diagrammes de Bode en gain et en phase du filtre, l'un en dessous de l'autre avec une échelle de pulsations logarithmique.

On utilisera la fonction `subplot` du module `matplotlib.pyplot` pour superposer les graphiques, et la fonction `semilogx` du module `matplotlib.pyplot` pour réaliser des échelles logarithmiques.

3.3 Réponse du filtre à un créneau

Le but est de tracer sur un intervalle de deux périodes, un créneau (de pulsation nommée `omegazero` réglable) et, en correspondance, la réponse temporelle du filtre passe-bas à ce créneau.

On rappelle la méthode de cours à mettre en œuvre pour trouver la réponse d'un filtre linéaire à un signal d'entrée périodique : on somme les réponses du filtre de chaque composante spectrale du signal en tenant compte du fait que le filtre affecte à la fois son amplitude et sa phase.

Question 18. Introduire la valeur choisie de `omegazero` (1000 rad/s par exemple), et calculer sa période

Question 19. En tenant compte des 97 premiers termes du développement de Fourier du signal d'entrée créneau, définir une fonction du temps qui soit la réponse temporelle du filtre.

Question 20. La tracer avec le créneau ; essayer avec une pulsation du créneau de 1000 rad/s, puis 10000 rad/s.

Question 21. Commenter ce dernier cas, et justifier le caractère intégrateur du filtre.

4 Annexe : fonctions et méthodes utiles

4.1 Fonctions graphiques

`plt.plot(X, Y, color = 'r', linewidth = 2)` : relie les points successifs définis par (X[i],Y[i]) en rouge avec une ligne d'épaisseur 2. Les deux derniers arguments sont optionnels.

`plt.show()` : affiche la figure créée

`plt.xlabel(s)` : donne le titre contenu dans la chaîne de caractères s aux abscisses.

`plt.ylabel(s)` : donne le titre contenu dans la chaîne de caractères s aux ordonnées.

`plt.title(s)` : donne le titre contenu dans la chaîne de caractères s à la figure.

`plt.subplot(1)` : crée un sous-graphique avec comme numéro 1.

`plt.semilogx(X,Y)` : comme plt.plot(X,Y) mais avec une échelle logarithmique.

`plt.grid()` : trace une grille sur le graphique.

`plt.close()` : ferme la figure.

4.2 Nombres complexes

Les nombres complexes sont définis en Python sous la forme `a+bj`. Par exemple `a=1+2j` et `b=0+3j` sont des déclarations de nombres complexes. Attention, on écrit aussi `1j` et pas `j`.

`type(a)` renvoie alors : `<class 'complex'>`

Un nombre complexe possède des *attributs* `real` et `imag` : ils correspondent à la partie réelle et à la partie imaginaire du nombre.
On peut les récupérer en Python à l'aide de `a.real` et `a.imag` (sans parenthèses, ce sont des attributs et non des méthodes ; on verra plus tard dans l'année la distinction, à l'occasion du TP sur la programmation orientée objet).

Il est possible de calculer directement le module d'un complexe a à l'aide de `abs(a)`

4.3 Fichiers externes : lecture

On peut ouvrir un fichier externe placé dans le répertoire de travail à l'aide de l'instruction `f = open(nom,'r')`, où `nom` est une chaîne de caractères contenant le nom du fichier à ouvrir.

`s = f.read()` permet de stocker dans la chaîne de caractères `s` le contenu du fichier `f`.

On peut aussi, sans utiliser `read`, directement travailler sur `f = open(nom,'r')`, qui trie automatiquement le fichier par lignes.

`f.close()` permet de fermer le fichier ouvert, lorsque le travail sur le fichier f est terminé.

4.4 Méthode split pour les chaînes de caractères

La méthode `split` permet de récupérer une liste de sous-chaînes de caractères à partir d'une chaîne de caractères donnée, en séparant cette chaîne selon un séparateur passé comme argument à `split`.

Par exemple, si `s = "jaune; vert; rouge; bleu"` alors l'instruction `L = s.split(";")`
crée la liste `L =["jaune", "vert", "rouge", "bleu"]`.