TP numérique

Décomposition en série de Fourier et application aux filtres

Pour tout le tp, on importera les modules *matplotlib.pyplot* et *math* en les renommant respectivement *plt* et *m*.

1 Graphes

Les fonctions et méthodes graphiques sont disponibles dans le module *matplotlib*. Les importer en entrant *import matplotlib.pyplot as plt* : toutes les fonctions et méthodes devront être préfixées par *plt*.
La fonction la plus importante est *plot*, qui prend en arguments deux listes (de même taille) de nombres et permet d'obtenir le graphe. Pour afficher le graphe à l'écran on utilise *show()* et pour le sauvegarder *savefig(nom_fichier)*.

1.1 Subdiviser un intervalle

Recopier la fonction suivante, la tester et écrire sa *docstring*.

```
def points (a,b,n) :
    out = []
    for i in range(n) :
        out.append(a+i*(b-a)/(n-1))
    return out
```

1.2 Représentation graphique

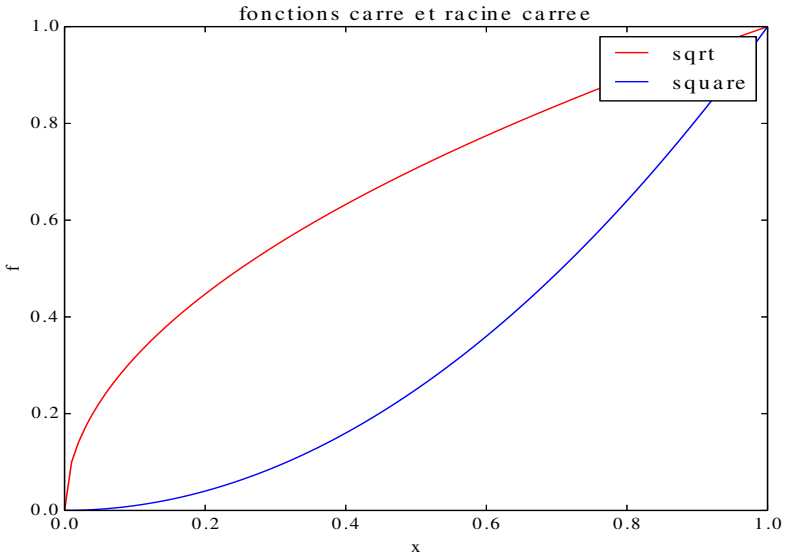
On peut enrichir le graphe en ajoutant un titre, des légendes, en modifiant les couleurs... A titre d'exemple, essayer le script suivant :

```
import matplotlib.pyplot as plt
import math as m

liste_x = points(0,1,100)
liste_y1 = [ m.sqrt(i) for i in liste_x ]
liste_y2 = [ i**2 for i in liste_x ]

plt.plot(liste_x,liste_y1,color ="red",label="sqrt")
plt.plot(liste_x,liste_y2,color="blue",label="square")
plt.legend()
plt.xlabel("x")
plt.ylabel("f")
plt.title("fonctions carre et racine carree")
plt.show()
```

Ce qui devrait donner le graphe :



Pour en savoir plus, entrer *matplotlib gallery* sur un moteur de recherche, on peut voir différents graphes et avoir accès au code source en cliquant dessus.

1.3 Fonction graphe

Ecrire une fonction qui prend en argument une fonction et deux nombres (les bornes de l'intervalle) et affiche le graphe de cette fonction sur l'intervalle spécifié, en subdivisant l'intervalle en 1000 points.

2 Fonction créneaux

Dans cette partie, on génère des fonctions créneaux qui seront utilisées dans la suite.

2.1 Créneaux sur [0,2π]

Initialiser une variable T à 2π (on pourra par la suite modifier cette valeur). Ecrire une fonction *creneaux* qui prend un nombre t en argument et renvoie :

$$\begin{cases} 1 \ si \ 0 \leq t \leq \frac{T}{2} \\ -1 \ si \ \frac{T}{2} < t \leq T \end{cases}$$

2.2 Créneaux périodiques

Modifier la fonction précédente de manière à ce qu'elle soit périodique, de période T. Visualiser le graphe sur l'intervalle [-10,10]. On *pourra* ajouter avant *plot* l'instruction *axis([xmin,xmax,ymin,ymax])* (en mettant des valeurs adéquates pour xmin,xmax,ymin et ymax) afin de mieux visualiser les créneaux.

2.3 Fonction *gen_ creneaux* qui génère des fonctions créneaux

Ecrire une fonction *gen_ creneaux* qui prend en argument une période T et renvoie une fonction créneaux de période T. C'est donc une fonction qui renvoie une fonction !

3 Fonction de transfert et diagramme de bode

3.1 Définition d'une fonction de transfert

Ecrire une fonction *passe_ bas* qui prend un nombre f_c (qui représente la fréquence de coupure) en argument et renvoie une fonction correspondant à la fonction de transfert d'un filtre passe bas du premier ordre de fréquence de

coupure f_c. On rappelle que Python possède un type complexe et que la notation est la même qu'en électricité (par exemple on peut entrer 2+3j, attention j sera considéré comme la variable j et non comme un nombre complexe, il faut entrer 1j à la place). On rappelle également que la fonction de transfert d'un filtre passe bas du premier ordre de fréquence de coupure f_c est :

$$\underline{H} = \frac{1}{1 + j\left(\frac{f}{f_c}\right)}$$

3.2 Gain et déphasage

Ecrire une fonction *gain* qui prend en entrée une fonction *transfert* et renvoie une fonction définie par :

$$f \rightarrow |transfert(f)|$$

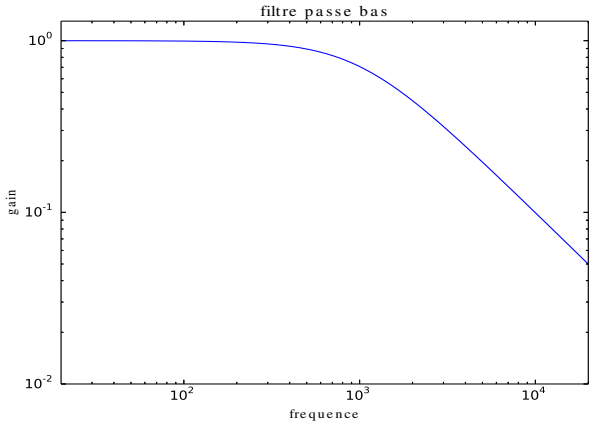
Ecrire de même une fonction *dephasage* qui renvoie une fonction définie par :

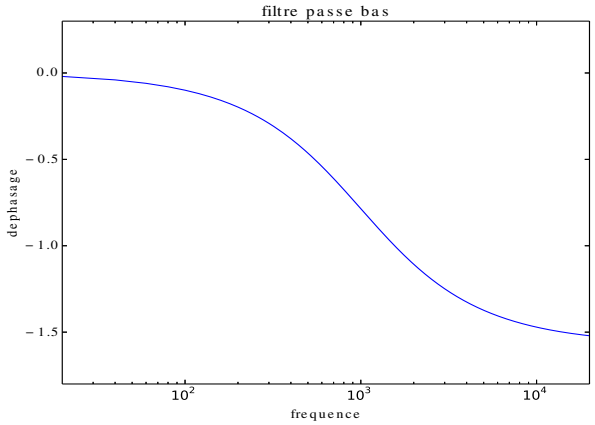
$$f \rightarrow \arg(transfert(f))$$

Attention, f est ici la variable et non une fonction, elle représente la fréquence.
On peut obtenir le module d'un nombre complexe avec la fonction *abs* (c'est une fonction prédéfinie de Python) et l'argument avec la fonction *phase* (dans le module *cmath*).

3.3 Diagrammes de Bode

Pour la fonction de transfert *passe_bas* définie précédemment, tracer le graphe du gain et du déphasage en fonction de la fréquence. Les fonctions *semilogx* et *loglog* (de *matplotlib*), à entrer avant de tracer le graphe, permettent d'obtenir le diagramme de Bode.





4 Décomposition en série de Fourier (rappels)

Une fonction périodique de période T peut être décomposée en une somme de fonctions sinusoidales de périodes $\frac{T}{n}$ avec n entier (strictement positif). Cette décomposition s'écrit de la manière suivante :

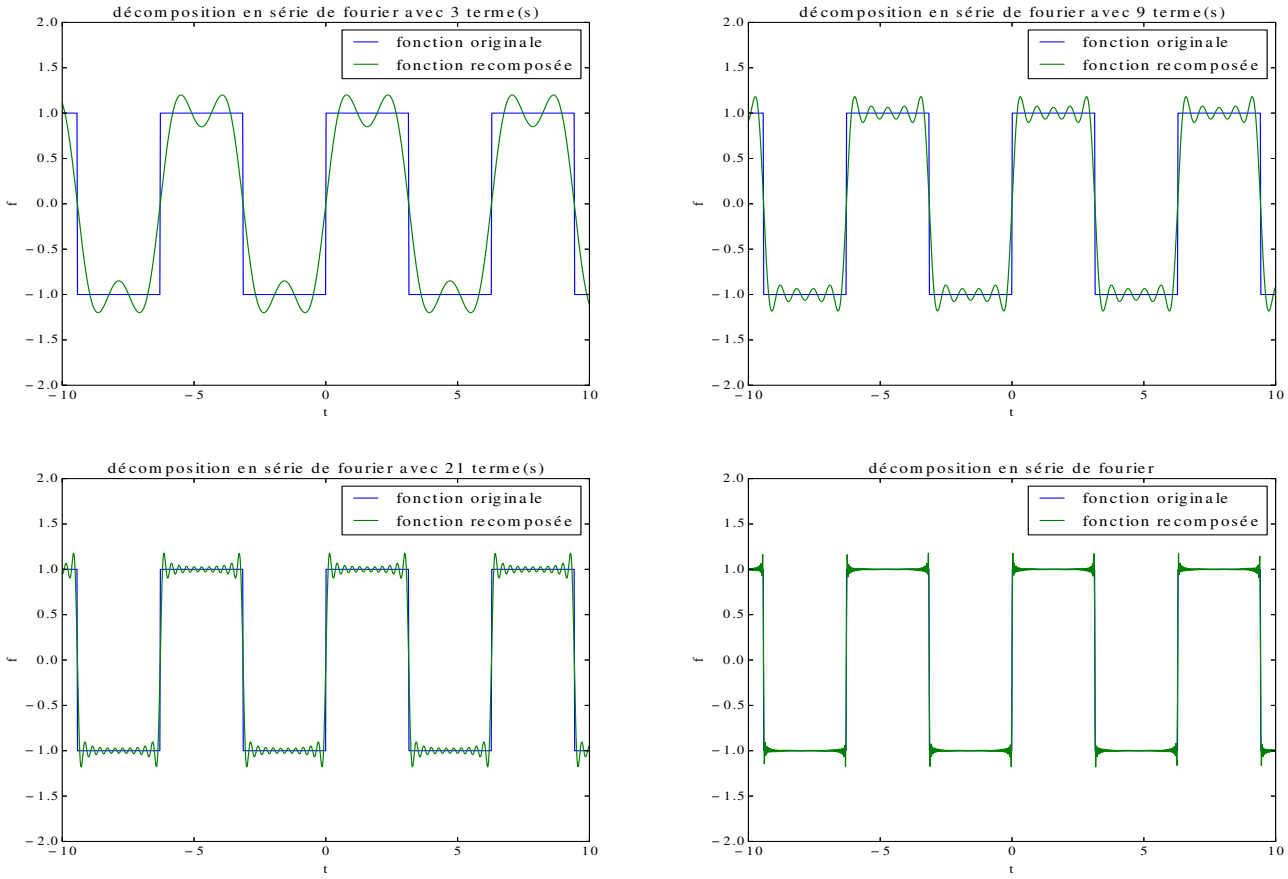
$$f(t) = < f > + \sum_{n=1}^{\infty} a_n \cos(n\omega t) + b_n \sin(n\omega t)$$

La valeur moyenne et les coefficients se calculent de la manière suivante :

$$< f >= \frac{1}{T}\int_0^T f(t)dt \qquad a_n = \frac{2}{T}\int_0^T f(t)\cos(n\omega t)\,dt \qquad b_n = \frac{2}{T}\int_0^T f(t)\sin(n\omega t)\,dt$$

Les notations ω et T représentent respectivement la pulsation et la période de f. La valeur moyenne est aussi appelée la composante continue de f, le premier terme de la série est le premier harmonique que l'on qualifie de *fondamental* car il a la même périodicité que f, les termes suivants sont les harmoniques (le n$^{\text{ième}}$ terme est le n$^{\text{ième}}$ harmonique).

L'objectif est de calculer les coefficients de Fourier a_n et b_n d'une fonction donnée (on commence par une fonctions simples à définir, des créneaux) puis de «recomposer »la fonction en sommant un certain nombre de termes. Plus le nombre de termes sommés est élevé, plus la somme se rapproche de la fonction originale, ce que l'on cherche à mettre en évidence.



5 Coefficients de Fourier

5.1 La fonction *quad*

De nombreuses possibilités de calcul numérique sont offertes par le module *scipy*, on va utiliser la fonction *quad* qui prend en arguments une fonction et deux nombres (les bornes de l'intégrale) et renvoie une liste contenant la valeur approchée de l'intégrale ainsi qu'une majoration de l'erreur.

Importer le module : *import scipy.integrate as si* (la fonction *quad* devra être préfixée par *si*).
Utiliser *quad* pour calculer une valeur approchée des intégrales suivantes et commenter :

$$\int_0^{2\pi} \cos(t)dt \quad et \quad \int_0^{2\pi} \cos^2(t)dt$$

5.2 Premiers calculs

Utiliser *quad* pour calculer une valeur approchée de la valeur moyenne et du coefficient b_1 pour la fonction *creneaux* (la valeur moyenne est nulle et $b_1 = \frac{4}{\pi}$). La fonction donnée comme argument à *quad* est le produit de la fonction *creneaux* et d'un sinus, mais Python ne peut multiplier directement des fonctions, il faut donc définir soi même (avec *def* ou *lambda*) la fonction proposée à *quad*.

5.3 Calcul des coefficients

Ecrire une fonctions *moyenne* qui prend en argument une fonction f et une période T et renvoie la valeur moyenne de f sur une période (le plus simple est de prendre l'intervalle [0,T]). Ecrire deux fonctions *coeff_ a* et *coeff_ b* qui prennent en argument une fonction f, une période T et un entier n et renvoient respectivement les coefficients de Fourier a_n et b_n de cette fonction.
Faire quelques tests, en particulier la fonction *creneaux* que l'on a définie est impaire, tous les coefficients a_n sont donc nuls.

6 Sommation

6.1 Principe

On veut maintenant calculer la somme des N premiers termes de la décomposition en série de Fourier, soit :

$$< f > + \sum_{n=1}^{N} a_n \cos(n\omega t) + b_n \sin(n\omega t)$$

On veut en particulier vérifier que plus N est grand et plus cette somme se rapproche de la fonction originale.

6.2 Ecriture de la fonction

Ecrire une fonction *fourier* qui prend en arguments une fonction f, une période T et un nombre N et renvoie une fonction égale à la somme des N premiers termes de la décomposition en série de Fourier de la fonction f.

6.3 Résultats

Tester la fonction *fourier* pour une fonction créneaux, pour différents nombres de termes, visualiser les résultats et commenter.

7 Spectre

La décomposition en série de Fourier peut aussi s'écrire sous la forme :

$$f(t) = < f > + \sum_{n=1}^{\infty} c_n \cos(n\omega t + \varphi_n)$$

Les coefficients c_n et ϕ_n sont liés à a_n et b_n introduits précédemment par :

$$c_n = \sqrt{a_n^2 + b_n^2} \quad \text{et} \quad \tan(\varphi_n) = -\frac{b_n}{a_n}$$

7.1 Calcul des coefficients c_n et ϕ_n

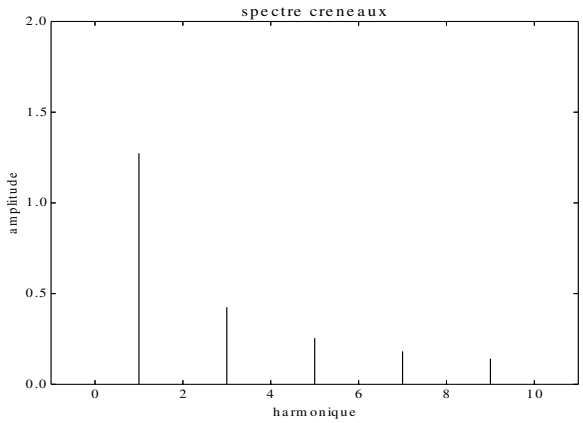
Créer deux fonctions *coeff_ c* et *coeff_ phi* qui prennent en argument une fonction, un nombre f et un nombre n et renvoient respectivement les coefficients de Fourier c_n et ϕ_n de cette fonction. Il faut bien sur faire appel aux fonctions *coeff_ a* et *coeff_ b*, attention à bien distinguer deux cas pour le calcul de ϕ_n.

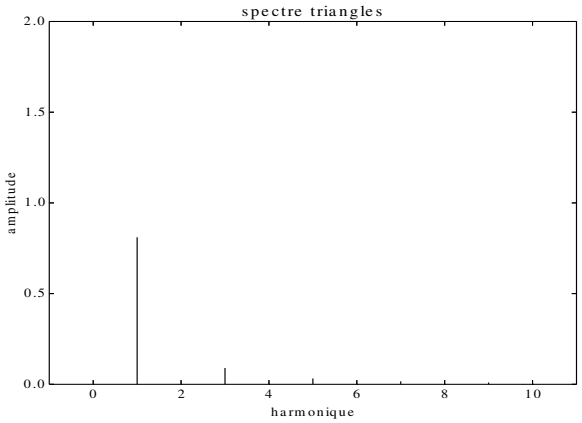
7.2 Essais

Ecrire une fonction *fourier2* qui fait la même chose que *fourier*, mais en utilisant les coefficients c_n et ϕ_n et vérifier que les résultats sont les mêmes que précédemment.

7.3 Spectre

On peut tracer un spectre avec la fonction *vlines* de matplotlib. Cette fonction prend à priori 3 listes de même longueur en arguments : la première contient les abcisses auxquelles les lignes doivent être tracées, la seconde les bornes inférieures des barres verticales et la troisième les bornes supérieures. Si on veut mettre la même valeur pour toutes les bornes inférieures, on peut remplacer la liste par un nombre (de même pour les bornes supérieures).
Obtenir le spectre des fonctions *creneaux* et *triangles* pour les dix premiers harmoniques.





8 Simulation de l'action d'un filtre sur un signal

8.1 Principe

Etant donné une fonction de transfert complexe $\underline{H}$ un signal d'entrée périodique de pulsation ω_s de la forme :

$$e = < e > + \sum_{n=1}^{\infty} c_n \cos(n\omega_s t + \varphi_n)$$

Le signal de sortie (la réponse du filtre, représenté par sa fonction de transfert, au signal d'entrée) s'écrit :

$$s = |H(0)| < e > + \sum_{n=1}^{\infty} |H(n\omega_s)|\, c_n \cos\left(n\omega_s t + \varphi_n + \arg(H(n\omega_s))\right)$$

C'est à dire que, pour chaque harmonique, l'amplitude est multipliée par le gain et la phase est décalée du déphasage (il s'agit bien sur du gain et du déphasage à la pulsation de l'harmonique).

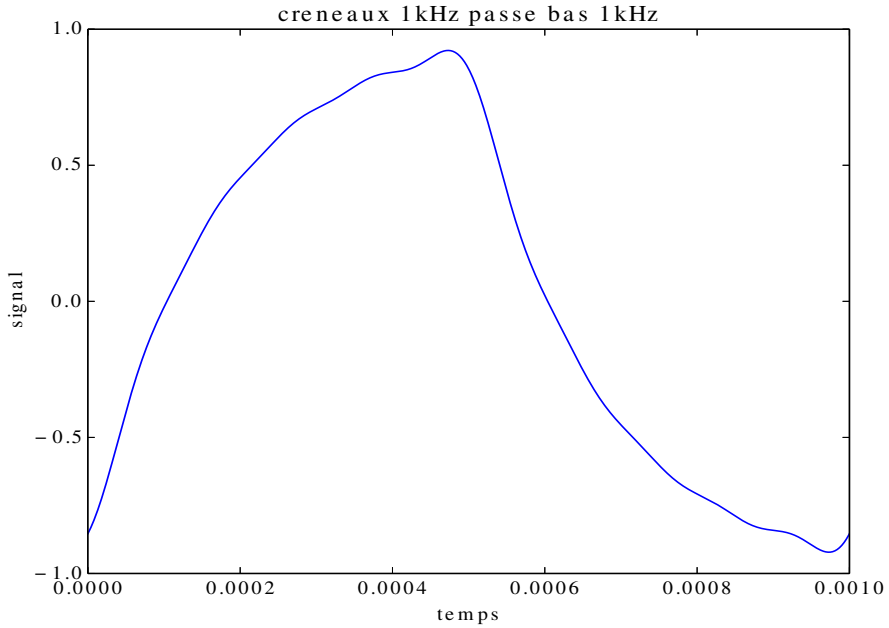
Remarque : Pour bien les différencier, les fréquence et pulsation de coupure du filtre sont notées ω_c et f_c alors que les fréquence et pulsation du signal sont notées ω_s et f_s

8.2 Calcul du signal de sortie

Ecrire une fonction *simulation* qui prend comme arguments :
— une fonction *signal* représentant le signal d'entrée
— un nombre f représentant la fréquence du signal
— un nombre N indiquant le nombre d'harmoniques prises en compte
— une fonction *transfert* pour la fonction de transfert choisie
Cette fonction doit renvoyer une fonction qui correspond au signal de sortie en se basant sur la formule donnée au paragraphe précédent, elle doit utiliser les fonctions *moyenne*, *coeff_c*, *coeff_phi*, *gain* et *dephasage*.

Attention : Les formules relatives aux séries de Fourier sont écrites avec des pulsations, mais on utilise ici des fréquences, $s = |H(0)| < e > + \sum_{n=1}^{\infty} |H(n\omega_s)|\, c_n \cos\left(n\omega_s t + \varphi_n + \arg(H(n\omega_s))\right)$ va donc s'écrire $s = |H(0)| < e > + \sum_{n=1}^{\infty} |H(nf_s)|\, c_n \cos\left(2\pi n f_s t + \varphi_n + \arg(H(nf_s))\right)$ où f_s est la fréquence du signal.

A titre d'essai, essayer *graphe(simulation(creneaux(1/1000),1000,20,passe_ bas(1000)),0,1/1000)*, on devrait observer :



8.3 Résultats et interprétation

En conservant une fonction de transfert passe bas du premier ordre avec une fréquence de coupure de 1kHz, faire des simulations pour des signaux créneaux de 10Hz , 1kHz et 100kHz avec 20 harmoniques et interpréter les résultats.