Analyse de Fourier et Filtrage

On se propose ici d'illustrer le cours sur l'analyse de Fourier et le filtrage en simulant des signaux et en les filtrant numériquement.
On partira à chaque fois d'un spectre connu, que nous construirons (fonctions `spectre_...()`). Nous construirons ensuite le signal correspondant (fonctions `signal()`) avant ou après filtrage (fonction `filtre_...`).
Pour les représentations graphiques du spectre d'une part, et du signal d'autre part, on construira deux fonctions `graphe_spectre()` et `graphe_signal()`.

a. *Mise en place de l'algorithme*

Le signal, périodique de période $T = \frac{1}{f}$, est représenté et construit à partir de sa décomposition en série de Fourier en cosinus, qui comportera nécessairement un nombre fini de termes [1]

$$s(t) = A_0 + \sum_{k=1}^{n} A_k \cos(2\pi k f t + \varphi_k)$$

Pour faciliter l'opération de filtrage et utiliser les fonctions de transfert usuelles, on travaille avec les nombres complexes. Le spectre est donc stocké sous la forme d'un *vecteur* (variable `tfs` [2], de type `array` à une dimension) contenant la série des amplitudes complexes [3] des composantes spectrales présentes :

$$A_k \, . \, e^{j\varphi_k} \ , \ k \in [0, n] \, .$$

On doit pouvoir associer à ce vecteur la liste des fréquences utilisées dans ce spectre. Il est préférable de les stocker séparément dans un vecteur `ff`, de sorte à pouvoir combiner librement toutes les composantes spectrales voulues sous la forme plus générale [4]

$$s(t) = A_0 + \sum_{k=1}^{n} A_k \cos(2\pi f_k t + \varphi_k) \, .$$

1. On aura donc forcément une représentation approchée des signaux qui comportent en théorie une infinité d'harmoniques, comme les signaux triangulaires ou rectangulaires.
2. pour *Transformée de Fourier du signal*, ce qu'il est à une constante multiplication près.
3. Ce nombre correspond en fait au double du coefficient C_k utilisé dans la forme complexe $s(t) = \sum_{k=-\infty}^{+\infty} C_k \, \mathrm{e}^{ik\omega t}$.
4. Une autre méthode consisterait à stocker seulement la fréquence fondamentale f du signal et de construire les autres sous la forme kf. Mais cela requiert de connaître f, ce qui n'est pas toujours immédiat.

En stockant aussi la fréquence nulle dans `ff` (composante continue), on obtient deux vecteurs `ff` et `tfs` de même taille $(n+1)$.
Pour un spectre de fonction de transfert

$$\underline{H}(j\omega) = G(\omega) \, . \, e^{j\varphi(\omega)} \ ,$$

l'amplitude complexe de la composante de pulsation ω est multipliée par $\underline{H}(j\omega)$. Cette opération peut se faire de façon vectorisée en utilisant les vecteurs Numpy. Ainsi l'opération de filtrage consistera simplement à multiplier le spectre contenu dans `tfs` par la fonction de transfert avant de reconstruire le signal.
Les fonctions de transfert seront écrite sous forme canonique, en fonction de la pulsation/fréquence réduite :

$$x = \frac{\omega}{\omega_0} = \frac{f}{f_0}$$

et il suffira donc donner des valeurs numériques pertinentes à f_0, à H_0, et à Q le cas échéant.

b. *Filtrage d'un signal à 2 composantes*

Commençons simplement par l'addition de deux signaux asynchrones, de fréquences assez différentes.

$$s(t) = A_1 \cos(2\pi f_1 t + \varphi_1) + A_2 \cos(2\pi f_2 t + \varphi_2) \, .$$

On construit d'abord son spectre, c'est-à-dire ses amplitudes complexes et les fréquences associées, grâce à la fonction suivante [5].

```
def spectre_2_comp(f1,A1,phi1,f2,A2,phi2):
    """Signal constitue d'une superposition d'une composante
    lente et d'une rapide:
    s(t) = A1 cos(2pi*f1*t + phi1) + A2 cos(2pi*f2*t + phi2)
    """
    ff = np.array([f1,f2])
    tfs = np.array([A1*np.exp(1j*phi1) , A2*np.exp(1j*phi2)])
    return ff, tfs
```

Pour construire le signal sur un domaine temporel à choisir (vecteur `tt` des instants) à partir du spectre on utilisera ensuite celle-ci :

```
def signal(tt, ff, tfs):
    """Calcule la fonction s(t) pour les instants contenus
    dans le vecteur tt a partir du spectre donne sous la forme
```

5. On rappelle que sous python, le nombre complexe j tel que $j^2 = -1$ s'écrit `1j`.

```
    du vecteur des frequences ff et du vecteur tfs dont le
    module est l'amplitude de l'harmonique associee a chaque
    frequence (en cosinus) et l'argument sa phase a l'origine :
        tfs[k] = A_k * exp ( i*phi_k ) pour la frequence ff[k].
    """
    n = len(ff) # Nb de composantes spectrales (composante continue comprise)
    ss = np.zeros(len(tt))  # Initialisation de la somme
    for k in range(n): # boucle d'addition des composantes spectrales presentes
        amp_k, phi_k = np.abs(tfs[k]), np.angle(tfs[k]) # amplitude, phase
        ss += amp_k * np.cos( 2*np.pi*ff[k]*tt + phi_k )
    return ss # vecteur des valeurs de s(t) pour les instants t dans tt
```

Les représentations graphiques seront utiles, non seulement pour le spectre (en amplitude à gauche et phase à droite),

```
def graphe_spectre(spectre):
    plt.figure(1) # ouverture de la figure 1
    plt.subplot(1,2,1)  # Graphe Spectre en amplitude
    plt.title("Spectre en amplitude")  # Titre
    plt.xlabel("f(Hz)")  # Légende des x
    plt.ylabel("Amplitude (SI)")  # Légende des y
    plt.bar(spectre[0],np.abs(spectre[1]), width=0.2)  # Graphe spectre en amplitude
    plt.subplot(1,2,2)  # Graphe Spectre en phase
    plt.title("Spectre en phase")  # Titre
    plt.xlabel("f(Hz)")  # Légende des x
    plt.ylabel("rad")  # Légende des y
    plt.bar(spectre[0],np.angle(spectre[1]), width=1)  #  Tracé du spectre en phase
    plt.show()
    return
```

mais surtout pour le signal en fonction du temps.

```
def graphe_signal(tt,ss,style='-k',epaisseur=1,etiquette="s(t)"):
    plt.figure(2) # ouverture de la figure 1
    plt.title("Signal")  # Titre
    plt.xlabel("t (s)")  # Légende des x
    plt.ylabel("s(t)")  # Légende des y
    plt.plot(tt, ss, style, linewidth=epaisseur, label=etiquette)  # Graphe temporel
    plt.grid()
    plt.legend()
    #plt.show()
    return
```

Ci-dessus on a commenté l'instruction `plt.show()` pour pouvoir superposer plusieurs courbes sur le même graphe avant de l'afficher.

Construisons un premier filtre de type passe-bas du premier ordre, dont la fonction de transfert canonique s'écrit

$$\underline{H} = \frac{H_0}{1+jx} = \frac{H_0}{1+jf/f_0}\,.$$

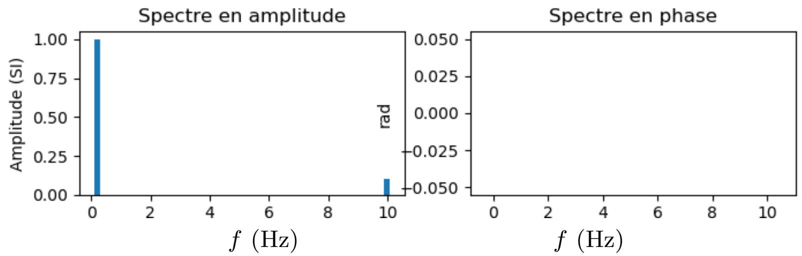
La fonction reçoit en argument les valeurs des paramètres canoniques f_0 et H_0, ainsi que le vecteur des fréquences utilisées `ff`, et renvoie un vecteur de même taille avec les valeurs associées de la fonction de transfert (calcul vectorisé) :

```
def FT_pbas_1erO(f0,H0,ff):
    xx = ff / f0 # fréquence réduite
    return H0 / ( 1 + 1j * xx ) # forme canonique
```

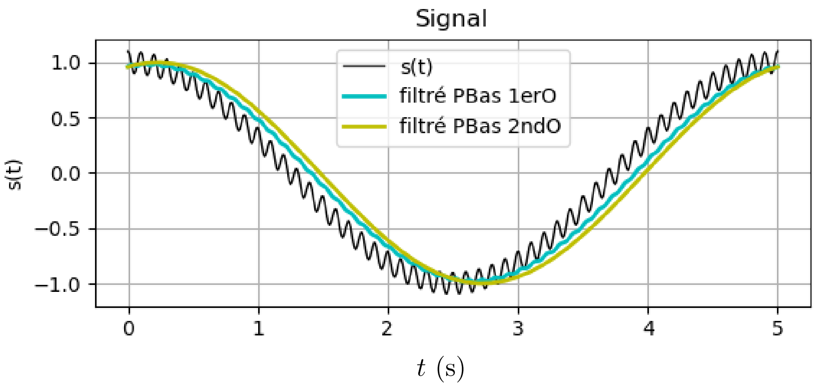
Les autres fonctions de filtres sont regroupées en annexe.

La suite d'instructions ci-dessous permet de construire et tracer le spectre puis le signal (fréquences 0,2 Hz et 10 Hz respectivement), accompagné de sa version filtrée par un passe-bas d'ordre 1 puis d'ordre 2 ($f_0 = 1\,\mathrm{Hz}$, et $Q = \frac{1}{\sqrt{2}}$).

```
# spectre
spectre = spectre_2_comp(0.2,1.,0.,10.,0.1,0.)
graphe_spectre(spectre)
# signal
instants = np.linspace(0,5,500)
ff, tfs = spectre
s = signal(instants,ff,tfs)
graphe_signal(instants,s)
# signal filtre
sf = signal(instants,ff,tfs*FT_pbas_1erO(1,1.,ff))
graphe_signal(instants,sf,'-c',2,"filtre PBas 1erO")
sf = signal(instants,ff,tfs*FT_pbas_2ndO(1,1/np.sqrt(2),1.,ff))
graphe_signal(instants,sf,'-y',2,"filtre PBas 2ndO")
```



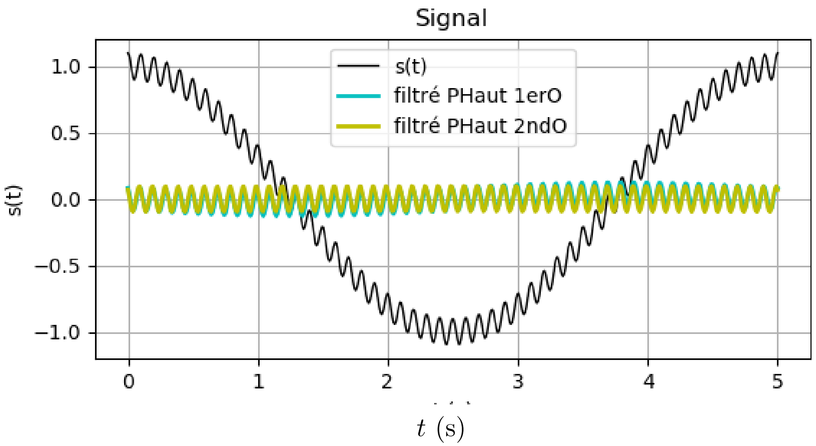
Le spectre ne contient que deux composantes, avec des phases à l'origine nulles.



On constate qu'avec les paramètres choisi pour les filtres, le filtrage du premier ordre n'est pas parfait contrairement à celui du second ordre qui conduit à un lissage complet.
Si au contraire on applique un filtrage passe-haut à ce même signal, du premier ordre puis du second,

```
sf = signal(instants,ff,tfs*FT_phaut_1erO(5.,1.,ff))
graphe_signal(instants,sf,'-c',2,"filtre PHaut 1erO")
sf = signal(instants,ff,tfs*FT_phaut_2ndO(5,1/np.sqrt(2),1.,ff))
graphe_signal(instants,sf,'-y',2,"filtre PHaut 2ndO")
```

on obtient alors les allures suivantes :



On observe une légère oscillation lente résiduelle suite au filtrage d'ordre 1, qui a totalement disparu avec le filtre d'ordre 2.

c. *Caractère intégrateur d'un filtre passe-bas d'ordre 1*

On illustre ici le cas d'un signal créneau pair, d'amplitude S_m et de fréquence f et de valeur moyenne A_0 :

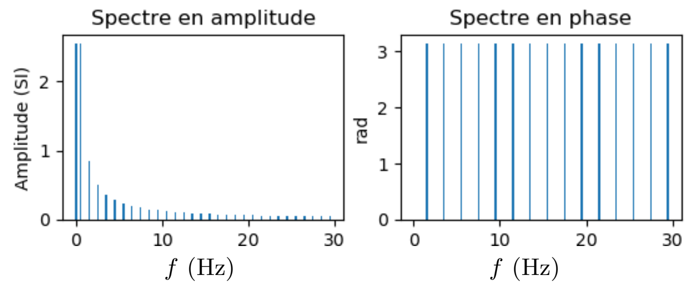
$$s(t) = A_0 + \frac{4S_m}{\pi} \sum_{n=0}^{\infty} \frac{1}{2n+1} \cos(2\pi(2n+1)ft)$$

approximé avec p harmoniques.

```
def spectre_creneau(f,A0,Sm,p):
    """Signal creneau pair (ou impair) de frequence f, de
    valeur moyenne A0, d'amplitude Sm, tronque a l'harmonique
    de rang n = 2*p-1 (il y a donc p harmoniques non nulles).
    """
    n = 2*p-1 # rang de la derniere harmonique, n+1 composantes avec la moyenne
    ff = np.arange(n+1) * f # serie de Fourier (multiples de la fondamentale)
    tfs = np.zeros(n+1, dtype=complex) # initialisation (type complexe!)
    tfs[0] = A0 # composante continue = valeur moyenne
    for i in range(p):
        k = 2*i+1 # rang de l'harmonique presente
        tfs[k] = (-1)**i / k # forme en cosinus, Paire
        #tfs[k] = 1 / k * np.exp(1j * (-np.pi/2)) # forme en sinus, Impaire
    tfs = tfs * 4*Sm/np.pi
    return ff, tfs
```

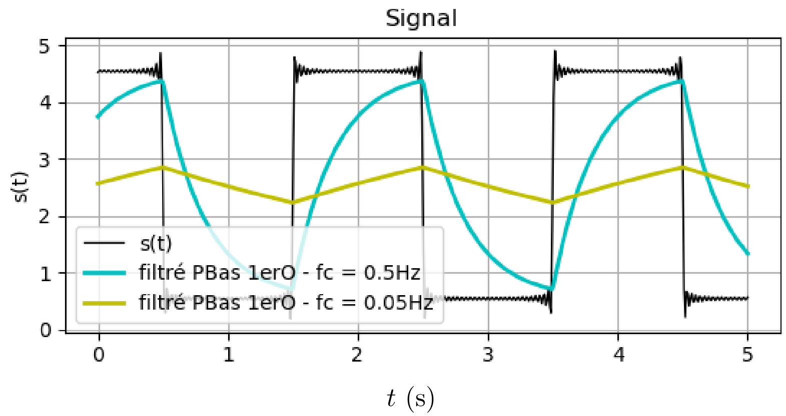
On construit le spectre d'un signal à 0,5 Hz avec $p = 30$ harmoniques et une moyenne non nulle,

```
spectre = spectre_creneau(0.5,1.,2.,30)
graphe_spectre(spectre)
```



puis le signal associé, que l'on filtre avec un filtre pas-bas d'ordre 1, d'abord avec une fréquence de coupure à 0,5 Hz puis de 0,05 Hz.

```
instants = np.linspace(0,5,500)
ff, tfs = spectre
s = signal(instants,ff,tfs)
graphe_signal(instants,s)
```



On observe bien des réponses exponentielles à des échelons dans le premier cas, et un signal triangulaire dans le second (intégration du créneau) correspondant à des réponses exponentielles de durée courte par rapport au temps caractéristique du filtre $\tau = \frac{1}{f_c}$.

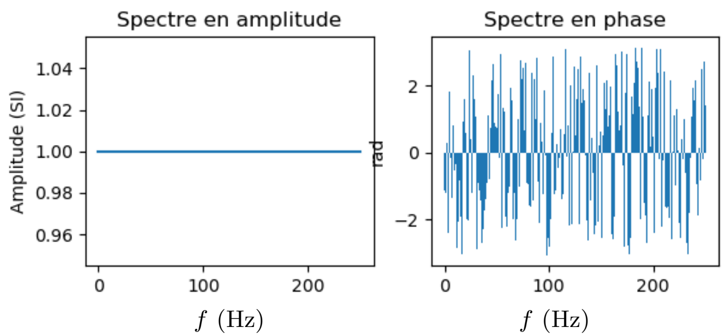
d. *Filtrage d'un bruit coloré*

On termine en illustrant le filtrage d'un bruit blanc [6] de fondamentale $f = 1\,\mathrm{Hz}$, dont on a maximisé le nombre d'harmoniques par rapport au critère de Shannon (pour éviter le *repliement spectral* (c'est-à-dire l'« effet stroboscopique » du au sous-échantillonnage).

```
def spectre_bruit_colore(f,p,Amp,n):
    """Signal aleatoire correspondant a un bruit colore, i.e.:
        A_k = Amp / (k*f)**p
    p = 0 -> bruit blanc
    p = 1 -> bruit rose
    p = 2 -> bruit rouge
    p = -1 -> bruit bleu
    p = -2 -> bruit violet
    ATTENTION: limiter la frequence maximale a f_n=fe/2=N/2T
    au max (T = duree simulation, N = nombre d'echantillons
    temporels) pour respecter le critere de Shannon et eviter
    le repliement spectral.
    Par exemple: f = 1/T et n = int(N/2)
    """
    ff = np.arange(n+1) * f # série de Fourier (multiples de la fondamentale)
    tfs = np.zeros(n+1, dtype=complex) # initialisation (type complexe !)
    tfs[0] = 0 # composante continue = valeur moyenne
    for k in range(n+1):
        phi_k = random.uniform(-np.pi,np.pi) # phase aléatoire
        tfs[k] = 1/k**p * np.exp( 1j*phi_k )
    tfs = tfs * Amp
    return ff, tfs
```

On affiche le spectre [7],

```
instants = np.linspace(0,5,500)
spectre = spectre_bruit_colore(1.,0,1.,int(len(instants)/2))
graphe_spectre(spectre)
ff, tfs = spectre
```



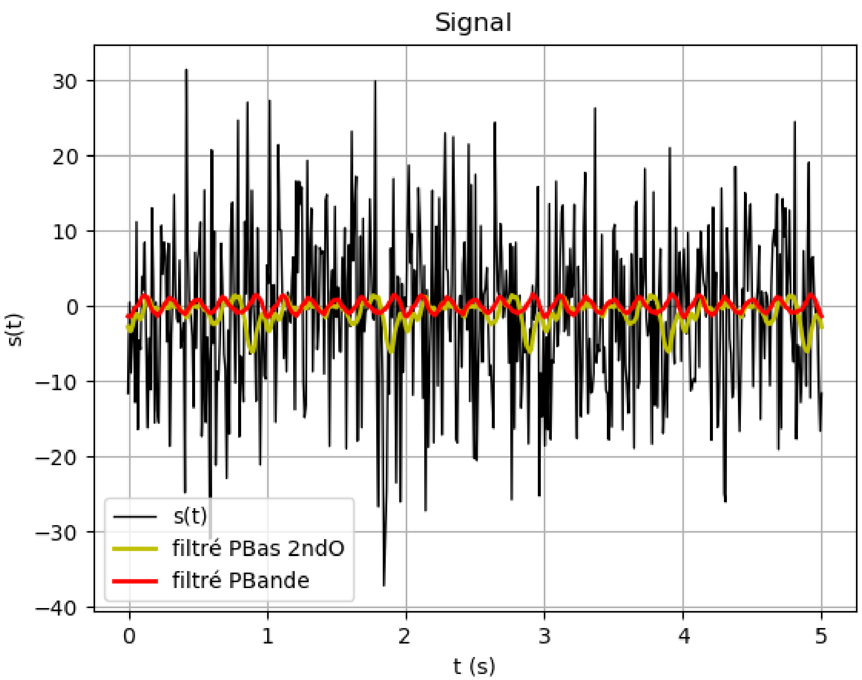
puis on trace le signal et on le filtre d'abord avec un filtre passe-bas d'ordre 2 ($f_c = 5\,\mathrm{Hz}$), puis ensuite avec un passe-bande très sélectif résonnant en $f_0 = 5\,\mathrm{Hz}$ avec $Q = 10$. Ce dernier filtrage assure une bande passante de largeur $\Delta f = \frac{f_0}{Q} = 0,5\,\mathrm{Hz}$, ce qui permet approximativement d'extraire une seule harmonique de ce bruit.

```
# signal
```

6. Un bruit blanc est caractérisé par un spectre en amplitude uniforme.

7. Éviter ici la fonction plt.bar pour ce spectre en amplitude pour l'imprimante...

```
s = signal(instants,ff,tfs)
graphe_signal(instants,s)
# signal filtre
sf = signal(instants,ff,tfs*FT_pbas_2ndO(5,1/np.sqrt(2),1.,ff))
graphe_signal(instants,sf,'-y',2,"filtre PBas 2ndO")
sf = signal(instants,ff,tfs*FT_pbande_2ndO(5,10,1.,ff))
graphe_signal(instants,sf,'-r',2,"filtre PBande")
plt.show()
```



ANNEXE

Ci-dessous on implémente les différents filtres utilisés par le biais de leur forme canonique. Les arguments d'entrée sont les paramètres canoniques et le vecteur des fréquences utilisées. La fonction renvoie le vecteur des valeurs de la fonction de transfert pour ces fréquences.

```
def FT_pbas_1erO(f0,H0,ff):
    xx = ff / f0
    return H0 / ( 1 + 1j * xx )

def FT_pbas_2ndO(f0,Q,H0,ff):
    xx = ff / f0
    return H0 / ( 1 + 1j * xx / Q - xx**2)

def FT_phaut_1erO(f0,H0,ff):
    xx = ff / f0
    return H0 * 1j * xx / ( 1 + 1j * xx )

def FT_phaut_2ndO(f0,Q,H0,ff):
    xx = ff / f0
    return H0 * (-xx**2) / ( 1 + 1j * xx / Q - xx**2)

def FT_pbande_2ndO(f0,Q,H0,ff):
    xx = ff / f0
    return H0 * 1j * xx / Q / ( 1 + 1j * xx / Q - xx**2)
```