HMEF104 — Electromagnétisme

TP4 - Capacités numériques

1 Objectifs du TP

- Savoir lire, écrire, et comprendre un code rédigé en Python pour répondre à un problème physique.
- Matériel :
  - Codes Python `signaux_*.py` (si besoin)

2 Environnement de travail

Il est mis à disposition des enseignants et étudiants du secondaire l'environnement `EduPython`, qui est téléchargeable sous Windows, mais également accessible sous les distributions Linux et MacOS (voir l'aide sur le site). Il est constitué d'un interpréteur Python 3.4 et d'un éditeur intégré (PyScripter). Des informations complémentaires peuvent être trouvées sur le site dédié :

`http://edupython.tuxfamily.org/`

Il rassemble notamment les librairies usuelles en Python :

- `numpy` pour la gestion des tableaux,
- `matplotlib` pour le tracé de courbes,
- `scipy` pour des traitements avancés de tableaux (intégration numérique, transformée de Fourier, signaux aléatoires, optimisation, etc.)

Sinon, pour programmer en Python, il est recommandé d'utiliser l'environnement Spyder, accessible depuis le package Anaconda d'installation de Python, qui dispose d'une aide intégrée : **pour coder en Python, l'aide en ligne est votre meilleure alliée** afin de connaître la manière dont fonctionnent les différentes commandes ! Pour les novices en Python, une bonne introduction se trouve à l'adresse :

`http://python.lycee.free.fr/`

3 Étude de signaux par la programmation en Python

3.1 Extraits des programmes officiels

- **En classe de Première de la filière générale :** Représenter un signal périodique et illustrer l'influence de ses caractéristiques (période, amplitude) sur sa représentation.
- **En classe de Terminale de la filière générale :** Représenter, à l'aide d'un langage de programmation, la somme de deux signaux sinusoïdaux périodiques synchrones en faisant varier la phase à l'origine de l'un des deux.
- **En classe de Terminale de la filière STL :** Utiliser un langage de programmation ou un tableur pour visualiser une somme de signaux sinusoïdaux de fréquences multiples de celle du fondamental.
- **En classe de Terminale de la filière STI2D :** Utiliser un outil numérique pour relever le spectre d'amplitude d'un signal sonore périodique.

3.2 Implémentation

3.2.1 Première étude de signaux sinusoïdaux

1. Définir une fonction $f(t, A, T, u) = A\cos(2\pi t/T + u)$ la sinusoïde d'amplitude A, de période T et de phase à l'origine u.

Solution: Voici un code minimal `signaux_1.py` (la première ligne permet d'utiliser des caractères spéciaux, comme les accents).

```
# -*- coding: utf-8 -*-

import math as m # librairie pour les fonctions mathématiques

def f(t,A,T,u): # fonction sinusoïdale
    return A*m.cos(2*m.pi*t/T+u)
```

2. Tracer la sinusoïde ainsi définie, en faisant varier l'amplitude A, la période T et la phase à l'origine u. On la tracera sur un intervalle de temps correspondant à cinq périodes, et on ajustera l'intervalle selon les deux axes. On donnera un titre aux axes, ainsi qu'au graphique où on précisera les valeurs de A, T et u choisies.

Solution: On utilise le code écrit à la question précédente pour le compléter par la définition de tableaux d'abscisses et d'ordonnées puis le tracé de courbes (`signaux_2.py`).

```
# -*- coding: utf-8 -*-

import numpy as np # librairie pour les tableaux
import matplotlib.pyplot as plt # librairie pour les figures
import math as m # librairie pour les fonctions mathématiques

def f(t,A,T,u): # fonction sinusoïdale
    return A*m.cos(2*m.pi*t/T+u)

T1=10 # période à modifier
A1=4. # amplitude à modifier
u1=0. # phase à l'origine à modifier
n=200 # nombre de points dans les tableaux à modifier
x1=np.linspace(0,5*T1,n) # définition des abscisses entre 0 et 5*T
y1=np.zeros(n) # définition des ordonnées (pour l'instant à 0)
for i in range(n):
    y1[i]=f(x1[i],A1,T1,u1)
fig,ax=plt.subplots() # création d'une figure
ax.plot(x1,y1,'-or') # tracé
ax.set_xlabel('t (s)') # titre de l'axe des abscisses
ax.set_ylabel ('Signal (u.a.)') # titre de l'axe des ordonnées
ax.set_xlim(0,5*T1) # limites des axes des abscisses
ax.set_ylim(-A1,A1) # limites des axes des ordonnées
ax.set_title('Sinusoïde pour A={} (u.a.), T={}s, u={} rad'.format(A1,T1,u1)) # titre
    de la figure avec les paramètres
fig.show() # montre la figure
```

3. Représenter sur le même graphique deux sinusoïdes de périodes T_1 et T_2, d'amplitude A_1 et A_2 et de phases à l'origine u_1 et u_2. On indiquera une légende avec les paramètres de chacune des sinusoïdes qui devra être placée au-dessus ou en-dessous des courbes sans dépasser du cadre de la figure, ni se superposer aux courbes.

Solution: On s'inspire du code de la question précédente (`signaux_3.py`).

```
# -*- coding: utf-8 -*-

import numpy as np # librairie pour les tableaux
import matplotlib.pyplot as plt # librairie pour les figures
import math as m # librairie pour les fonctions mathématiques

def f(t,A,T,u): # fonction sinusoïdale
    return A*m.cos(2*m.pi*t/T+u)

T1=10 # période du premier signal à modifier
A1=4. # amplitude du premier signal à modifier
u1=0. # phase à l'origine du premier signal à modifier
T2=2 # période du second signal à modifier
A2=1. # amplitude du second signal à modifier
u2=m.pi/4. # phase à l'origine du second signal à modifier
n=200 # nombre de points dans les tableaux à modifier
x=np.linspace(0,5*max(T1,T2),n) # définition des abscisses
y1=np.zeros(n) # définition des ordonnées pour le premier signal (pour l'instant à 0)
y2=np.zeros(n) # définition des ordonnées pour le second signal (pour l'instant à 0)
for i in range(n):
    y1[i]=f(x[i],A1,T1,u1)
    y2[i]=f(x[i],A2,T2,u2)
fig,ax=plt.subplots() # création d'une figure
ax.plot(x,y1,'-or',label='Signal 1 avec A={:.2f} (u.a.), T={:.2f}s, u={:.2f} rad'.
    format(A1,T1,u1)) # tracé du premier signal (les valeurs sont données avec deux chiffres
    après la virgule)
ax.plot(x,y2,'-pg',label='Signal 2 avec A={:.2f} (u.a.), T={:.2f}s, u={:.2f} rad'.
    format(A2,T2,u2)) # tracé du second signal (les valeurs sont données avec deux chiffres
    après la virgule)
ax.set_xlabel('t (s)') # titre de l'axe des abscisses
ax.set_ylabel ('Signaux (u.a.)') # titre de l'axe des ordonnées
ax.set_xlim(0,5*max(T1,T2)) # limites des axes des abscisses
ax.set_ylim(-max(A1,A2),max(A1,A2)+2) # limites des axes des ordonnées (le +2 sert à laisser
    de la place au-dessus pour mettre la légende)
ax.set_title('Deux sinusoïdes') # titre de la figure
ax.legend(loc=9) # légende (loc=9 correspond à la position de la légende en haut au centre)
fig.show() # montre la figure
```

4. Représenter sur le même graphique deux sinusoïdes de même période T, d'amplitudes $A_1 = 1$ u.a. et $A_2 = A$, et de phases à l'origine $u_1 = 0\,\text{rad}$ et $u_2 = u$, ainsi que la somme des deux signaux. Faire varier A et u. Là encore, on mettra une légende et on indiquera un titre au graphique.

Solution: Là encore, on s'inspire de la question précédente (`signaux_4.py`).

```
# -*- coding: utf-8 -*-

import numpy as np # librairie pour les tableaux
import matplotlib.pyplot as plt # librairie pour les figures
import math as m # librairie pour les fonctions mathématiques

def f(t,A,T,u): # fonction sinusoïdale
    return A*m.cos(2*m.pi*t/T+u)

T1=1. # période du premier signal à modifier
A1=1. # amplitude du premier signal fixée à 1 u.a.
u1=0. # phase à l'origine du premier signal fixée à 0 rad
T2=T1 # période du second signal égale à celle du premier
A2=1.2 # amplitude du second signal à modifier
u2=m.pi/2. # phase à l'origine du second signal à modifier
n=200 # nombre de points dans les tableaux à modifier
x=np.linspace(0,5*max(T1,T2),n) # définition des abscisses
```

```
y1=np.zeros(n) # définition des ordonnées pour le premier signal (pour l'instant à 0)
y2=np.zeros(n) # définition des ordonnées pour le second signal (pour l'instant à 0)
for i in range(n):
    y1[i]=f(x[i],A1,T1,u1)
    y2[i]=f(x[i],A2,T2,u2)
fig,ax=plt.subplots() # création d'une figure
ax.plot(x,y1,'-or',label='Signal 1 avec A={:.2f} (u.a.), T={:.2f}s, u={:.2f} rad'.
    format(A1,T1,u1)) # tracé du premier signal (les valeurs sont données avec deux chiffres
    après la virgule)
ax.plot(x,y2,'-pg',label='Signal 2 avec A={:.2f} (u.a.), T={:.2f}s, u={:.2f} rad'.
    format(A2,T2,u2)) # tracé du second signal (les valeurs sont données avec deux chiffres
    après la virgule)
ax.plot(x,y1+y2,'-sb',label='Somme des deux sinusoïdes') # tracé de la somme
ax.set_xlabel('t (s)') # titre de l'axe des abscisses
ax.set_ylabel ('Signaux (u.a.)') # titre de l'axe des ordonnées
ax.set_xlim(0,5*max(T1,T2)) # limites des axes des abscisses
ax.set_ylim(-(A1+A2),A1+A2+1) # limites des axes des ordonnées (le +1 sert à laisser de la
    place au-dessus pour mettre la légende)
ax.set_title('Deux sinusoïdes et leur somme') # titre de la figure
ax.legend(loc=9) # légende (loc=9 correspond à la position de la légende en haut au centre)
fig.show() # montre la figure
```

3.2.2 Analyse spectrale d'un signal

1. Représenter un signal sinusoïdal fondamental de fréquence ν, d'amplitude $A = 1$ u.a. et de phase à l'origine u auquel on ajoute $m - 1$ signaux harmoniques de fréquence 2ν, 3ν, etc., et d'amplitudes A_k $(k = 2, .., p)$ et de phases à l'origine u_k qu'on fera varier. Là encore on mettra un titre au graphique. On étudiera les cas suivants et on commentera, notamment, l'évolution avec p :
   - $A_k = \begin{cases} 0 \text{ si } k \text{ est pair} \\ \dfrac{A}{k} \text{ si } k \text{ est impair} \end{cases}$, $u_k = \dfrac{\pi}{2}$, $u = \pi/2$, et $p = 1, 3, 5, 9, 25, 100, 200$.
   - $A_k = \begin{cases} 0 \text{ si } k \text{ est pair} \\ \dfrac{A}{k^2} \text{ si } k \text{ est impair} \end{cases}$, $u_k = 0$, $u = 0$, et $p = 1, 3, 5, 9, 25, 100, 200$.

Solution: Par exemple, pour le premier cas, on donne un code minimal (`signaux_5.py`).

```
# -*- coding: utf-8 -*-

import numpy as np # librairie pour les tableaux
import matplotlib.pyplot as plt # librairie pour les figures
import math as m # librairie pour les fonctions mathématiques

def f(t,A,f,u): # fonction sinusoïdale
    return A*m.cos(2*m.pi*f*t+u)

nu=1. # fréquence du fondamental à modifier
A=1. # amplitude du fondamental fixée à 1 u.a.
u=m.pi/2. # phase à l'origine du fondamental fixée à 0 rad
p=200 # nombre d'harmoniques
Ah=np.zeros(p) # tableau contenant les amplitudes de chaque harmonique (pour l'instant à 0)
Ah[0]=A # on place l'amplitude du fondamental dans le tableau
for j in range(1,p):
    if (j%2==0): # on ne garde que les indices pairs (qui correspondent aux harmoniques impairs
    car les tableaux sont indexés à partir de 0 en Python), le % désigne le reste dans la division
    euclidienne
        Ah[j]=1./(j+1) # amplitude pour les termes impairs (par exemple pour j=2, cela
    correspond au troisième harmonique d'amplitude 1/3)
n=2000 # nombre de points dans les tableaux à modifier
```

```
x=np.linspace(0,5./nu,n) # définition des abscisses
y=np.zeros(n) # définition des ordonnées (pour l'instant à 0)
for i in range(n):
    for j in range(p):
        y[i]+=f(x[i],Ah[j],(j+1)*nu,m.pi/2.)
fig,ax=plt.subplots() # création d'une figure
ax.plot(x,y) # tracé du signal
ax.set_xlabel('t (s)') # titre de l'axe des abscisses
ax.set_ylabel ('Signal (u.a.)') # titre de l'axe des ordonnées
ax.set_xlim(0,5./nu) # limites des axes des abscisses
ax.set_ylim(np.amin(y),np.amax(y)) # limites des axes des ordonnées
ax.set_title('Somme du fondamental et des harmoniques') # titre de la figure
fig.show() # montre la figure
```

On obtient un créneau quand p augmente (c'est le principe de la décomposition en série de Fourier). Le second cas correspond à un signal triangulaire : la convergence est plus rapide avec p que dans le cas du créneau.

2. Tracer le signal $f(t) = \cos(2\pi\nu t) + 0.5\cos(5\pi\nu t) + 0.4\cos(3\pi\nu t)$ avec Python (où $\nu = 10\,\text{Hz}$). Ce signal est-il périodique ? Si oui, donner sa période.

Solution: Ce signal est en effet périodique de période $T = 0{,}2\,\text{s}$. Voici un code minimal pour le tracer (`signaux_6.py`).

```
# -*- coding: utf-8 -*-

import numpy as np # librairie pour les tableaux
import matplotlib.pyplot as plt # librairie pour les figures

nu=10. # fréquence du fondamental à modifier
n=200 # nombre de points dans les tableaux à modifier
x=np.linspace(0,5./nu,n) # tableau des abscisses
y=np.cos(2*np.pi*nu*x)+0.5*np.cos(5*np.pi*nu*x)+0.4*np.cos(3*np.pi*nu*x) #tableau des
    ordonnées
fig,ax=plt.subplots() # création d'une figure
ax.plot(x,y,'-og') # tracé
ax.set_xlabel('t (s)') # titre de l'axe des abscisses
ax.set_ylabel ('Signal (u.a.)') # titre de l'axe des ordonnées
ax.set_xlim(0,5./nu) # limites des axes des abscisses
ax.set_ylim(np.amin(y),np.amax(y)) # limites des axes des ordonnées
ax.set_title(r'Tracé du signal $f(t)$ avec $\nu=1$ Hz') # titre de la figure
fig.show() # montre la figure
```

3. Calculer puis tracer le spectre du signal $f(t)$ pour $\nu = 10\,\text{Hz}$. Pour cela, on définira un tableau d'abscisses `x` contenant n points entre 0 et t_f et un tableau d'ordonnées `y` de même taille, et on utilisera les commandes pour :
   - le calcul du spectre du signal `y` : `numpy.abs(numpy.fft.rfft(y))`,
   - le calcul des fréquences du spectre : `numpy.fft.rfftfreq(n,dt)` (où `dt` désigne le pas d'échantillonnage du signal).

   On pourra, en particulier, étudier les cas suivants :
   - $t_\text{f} = 0{,}2\,\text{s}$ et $n = 20$,
   - $t_\text{f} = 0{,}3\,\text{s}$ et $n = 20$,
   - $t_\text{f} = 1\,\text{s}$ et $n = 45$,
   - $t_\text{f} = 1\,\text{s}$ et $n = 60$.

   Commenter les rôles de la durée d'acquisition t_f et du nombre de points n (ou de manière équivalente de la fréquence d'échantillonnage), en comparant au spectre attendu.

Solution: Voici un code minimal pour calculer le spectre du signal $f(t)$ (`signaux_7.py`).

```
# -*- coding: utf-8 -*-

import numpy as np # librairie pour les tableaux
import matplotlib.pyplot as plt # librairie pour les figures

nu=10. # fréquence du fondamental à modifier
tf=1. # durée de l'intervalle à modifier
n=60 # nombre de points dans les tableaux à modifier
x=np.linspace(0,tf,n) # tableau des abscisses
y=np.cos(2*np.pi*nu*x)+0.5*np.cos(5*np.pi*nu*x)+0.4*np.cos(3*np.pi*nu*x) #tableau des
    ordonnées
ys=np.abs(np.fft.rfft(y)) # spectre
fs=np.fft.rfftfreq(n,tf/n) # fréquences correspondantes
fig,ax=plt.subplots() # création d'une figure
ax.plot(fs,ys,'-og') # tracé
ax.set_xlabel('f (Hz)') # titre de l'axe des abscisses
ax.set_ylabel ('Spectre (u.a.)') # titre de l'axe des ordonnées
ax.set_title(r'Tracé du spectre de $f(t)$ avec n={} et $t_f$={:.2f}'.format(n,tf))
    # titre de la
    figure
fig.show() # montre la figure
```

Le spectre attendu est la superposition de trois pics aux fréquences 10 Hz, 15 Hz et 25 Hz. On analyse maintenant les quatre situations données par l'énoncé.

Dans le premier cas, on n'observe que deux pics à 10 Hz et 25 Hz. Cela vient d'un problème de résolution du spectre : l'écart en fréquences entre deux points est trop important. Cet écart entre points doit être diminué si on veut effectivement observer les trois pics.

Dans le second cas, on observe bien trois pics, aux fréquences 10 Hz, 16,7 Hz et 26,7 Hz. Entre la situation précédente et celle-ci, seule la durée d'acquisition a changé. On en conclut donc que la durée d'acquisition contrôle la résolution du spectre : **pour avoir le spectre le plus résolu possible, il faut faire une acquisition la plus longue possible.** La fréquence minimale observable dans un spectre (égale à sa résolution) est **l'inverse de la durée d'acquisition**. Précédemment, cette résolution valait $\delta f = 1/t_\mathrm{f} = 5\,\mathrm{Hz}$, alors que maintenant elle vaut $\delta f = 3{,}3\,\mathrm{Hz}$, ce qui permet la résolution des trois pics. Par contre, les valeurs de fréquences sont décalées par rapport au spectre attendu. Ce décalage vient là encore du fait que la résolution dans le domaine spectrale n'est pas assez grande : il faudrait encore augmenter t_f pour diminuer δf.

Dans le troisième cas, on observe bien trois pics, mais aux fréquences 10 Hz, 15 Hz et 19 Hz. Le dernier pic n'est donc pas obtenu à la bonne fréquence. Pour comprendre cela, il est utile de comparer à la fréquence d'échantillonnage $f_\mathrm{e} = n/t_\mathrm{f} = 45\,\mathrm{Hz}$. On voit donc que **le critère de Shannon n'est pas vérifié : la fréquence d'échantillonnage devrait être au moins égale au double de la fréquence maximale présente dans le spectre. Dans le cas contraire, on observe un repliement spectral : le pic à la fréquence $f \geq f_\mathrm{e}/2$ se retrouve dans le spectre à la fréquence $kf_\mathrm{e} - f$ (où l'entier k est tel que -$f_\mathrm{e}/2 \leq kf_\mathrm{e} - f \leq f_\mathrm{e}/2$).** C'est ce qu'on observe ici : le pic attendu à 25 Hz se retrouve à la fréquence $45 - 25 = 20\,\mathrm{Hz}$.

Dans le dernier cas, on observe bien trois pics aux bonnes fréquences : la résolution spectrale $\delta f = 1/t_\mathrm{f} = 1\,\mathrm{Hz}$ est suffisante, et le critère de Shannon est maintenant respecté car $f_\mathrm{e} = 60\,\mathrm{Hz}$ (de sorte que $f_\mathrm{e}/2 \geq 25\,\mathrm{Hz}$). On retiendra donc que **pour éviter le repliement spectral, il faut augmenter la fréquence d'échantillonnage, et donc le nombre de points d'acquisition à durée d'acquisition similaire.**

Conclusion : La durée d'acquisition fixe la résolution spectrale, et le nombre de points la fréquence maximale mesurable (la largeur du domaine spectral sondé).

4. Calculer le spectre de la fonction $f(t) = \cos(2\pi\nu t)$ avec $\nu = 1\,\mathrm{Hz}$ en faisant varier t_f. On prendra à chaque fois un nombre n de points de sorte que le critère de Shannon soit vérifié. On tracera deux

spectres sur le même graphique, par exemple pour $t_f = 10\,s$ ou $t_f = 9{,}5\,s$. Commenter.

Solution: On choisit dans les deux cas une fréquence d'échantillonnage bien supérieure à 1 Hz pour vérifier le critère de Shannon (ici 10 Hz). Voici un code minimal pour répondre à la question (`signaux_8.py`).

```
# -*- coding: utf-8 -*-

import numpy as np # librairie pour les tableaux
import matplotlib.pyplot as plt # librairie pour les figures

nu=1. # fréquence du fondamental à modifier
tf1=10. # durée de l'intervalle à modifier
tf2=9.5 # durée de l'intervalle à modifier
n1=100 # nombre de points d'acquisition
n2=95 # nombre de points d'acquisition
x1=np.linspace(0,tf1,n1) # tableau des abscisses
x2=np.linspace(0,tf2,n2) # tableau des abscisses
y1=np.cos(2*np.pi*nu*x1) #tableau des ordonnées
y2=np.cos(2*np.pi*nu*x2) #tableau des ordonnées
ys1=np.abs(np.fft.rfft(y1,n1)) # spectre
ys2=np.abs(np.fft.rfft(y2,n2)) # spectre
fs1=np.fft.rfftfreq(n1,tf1/n1) # fréquences
fs2=np.fft.rfftfreq(n2,tf2/n2) # fréquences
fig,ax=plt.subplots() # création d'une figure
ax.plot(fs1,ys1,'-og',label=r'$t_\mathrm{f}$=' + '{:.2f}s'.format(tf1)) # tracé
ax.plot(fs2,ys2,'-pr',label=r'$t_\mathrm{f}$=' + '{:.2f}s'.format(tf2)) # tracé
ax.set_xlabel('f (Hz)') # titre de l'axe des abscisses
ax.set_ylabel ('Spectre (u.a.)') # titre de l'axe des ordonnées
ax.set_title(r'Tracé des spectres de $f(t)$') # titre de la figure
ax.legend(loc=0) # légende (loc=0 signifie que le code trouve tout seul le meilleur endroit pour
    la placer)
fig.show() # montre la figure
```

On constate que le pic à la fréquence de 1 Hz s'élargit quand $t_f = 9{,}5\,s$. Cela vient du fait qu'on n'a pas enregistré un nombre entier de périodes du signal. Cela résulte en une diminution de la résolution dans le domaine spectral. **On se débrouillera donc toujours pour enregistrer un nombre entier de périodes si cela est possible** (sinon voir la question suivante).

5. Reprendre la question précédente, mais cette fois-ci en multipliant le signal enregistré durant $t_f = 9{,}5\,s$ par :
   - une fenêtre de Hanning : $w(t) = 0.5 - 0.5\cos(2\pi t/t_f)$,
   - une fenêtre de Blackman : $w(t) = 0.42 - 0.5\cos(2\pi t/t_f) + 0.8\cos(4\pi t/t_f)$.

   Quelles propriétés ont ces fonctions ? Comment cela se traduit-il sur les spectres ?

Solution: Les deux fenêtres $w(t)$ proposées sont telles que $w(0) = w(t_f) = 0$. Voici un code minimal pour répondre à la question (`signaux_9.py`).

```
# -*- coding: utf-8 -*-

import numpy as np # librairie pour les tableaux
import matplotlib.pyplot as plt # librairie pour les figures

nu=1. # fréquence du fondamental à modifier
tf1=10. # durée de l'intervalle à modifier
tf2=9.5 # durée de l'intervalle à modifier
n1=100 # nombre de points d'acquisition
n2=95 # nombre de points d'acquisition
x1=np.linspace(0,tf1,n1) # tableau des abscisses
x2=np.linspace(0,tf2,n2) # tableau des abscisses
```

```
y1=np.cos(2*np.pi*nu*x1) #tableau des ordonnées
y2=np.cos(2*np.pi*nu*x2) #tableau des ordonnées
ys1=np.abs(np.fft.rfft(y1,n1)) # spectre
ys2=np.abs(np.fft.rfft(y2,n2)) # spectre
ys3=np.abs(np.fft.rfft(y2*np.hanning(n2),n2)) # spectre
ys4=np.abs(np.fft.rfft(y2*np.blackman(n2),n2)) # spectre
fs1=np.fft.rfftfreq(n1,tf1/n1) # fréquences
fs2=np.fft.rfftfreq(n2,tf2/n2) # fréquences
fig,ax=plt.subplots() # création d'une figure
ax.plot(fs1,ys1,'-og',label=r'$t_\mathrm{f}$=' + '{:.2f}s'.format(tf1)) # tracé
ax.plot(fs2,ys2,'-pr',label=r'$t_\mathrm{f}$=' + '{:.2f}s'.format(tf2)) # tracé
ax.plot(fs2,ys3,'-bs',label=r'$t_\mathrm{f}$=' + '{:.2f}s + Hanning'.format(tf2)) #
    tracé
ax.plot(fs2,ys4,'-y^',label=r'$t_\mathrm{f}$=' + '{:.2f}s + Blackman'.format(tf2))
    #
    tracé
ax.set_xlabel('f (Hz)') # titre de l'axe des abscisses
ax.set_ylabel ('Spectre (u.a.)') # titre de l'axe des ordonnées
ax.set_title(r'Tracé des spectres de $f(t)$') # titre de la figure
ax.legend(loc=0) # légende (loc=0 signifie que le code trouve tout seul le meilleur endroit pour
    la placer)
fig.show() # montre la figure
```

On observe que la largeur du pic diminue avec les fenêtres pour retrouver une largeur proche de celle correspondant au signal enregistré avec $t_\mathrm{f} = 10\,\mathrm{s}$. C'est le principe de l'**apodisation** : on multiplie par une fenêtre qui s'annule aux bords de l'intervalle d'acquisition pour rendre le périodisé du signal enregistré de période t_f continu aux multiples entiers de la période. **On retiendra donc que pour améliorer la résolution d'un spectre, on peut multiplier le signal par une fenêtre d'apodisation qui s'annule aux bords de l'intervalle d'acquisition avant de calculer son spectre.**