TP filtre numerique

October 14, 2021

Dans ce TP vous allez mettre en oeuvre une chaine complète de traitement numérique d'un enregistrement audio : un capteur, un CAN, votre ordinateur comme traitement numérique et CNA, et vos écouteurs.

1 Numérisation du signal analogique

Branchez la sortie analogique de votre micro sur la carte d'acquisition et utilisez Latis pro pour commander la carte d'acquisition.

Pour éviter que les étapes de traitement numérique ne prennent trop temps de calcul, vous choisirez un temps d'acquisition de moins de 10s.

Choisissez une période d'échantillonnage qui permette de numériser votre voix correctement. Vous pouvez faire une recherche sur les caractéristiques de la voix pour cela.

Enregistrez votre voix et exportez vos données dans un fichier "data.txt" avec des nombres séparées par des tabulations et dont le séparateur décimal est un point. Après exportation vous pourrez effacer manuellement l'en-tête du fichier texte.

Pour visualiser le résultats nous avons besoin des librairies

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
```

Puis nous pouvons utiliser la fonction np.loadtxt pour importer les données

[ ]:
```
data = np.loadtxt("data.txt")
```

Pour visualiser la façon dont les données sont enregistrée dans data vous pouvez d'abord utiliser la fonction print, puis faire un graphe.

[ ]:
```
print(data)
```

[ ]:
```
t = # récupérer ici la liste des temps
e = #  récupérer ici la liste des tensions
x = # ce que vous voulez tracer en abscisse
y = # ce que vous voulez tracer en ordonnée
plt.plot(x,y,'.', label="ma légende")
plt.ylabel('axe des ordonnées')
plt.xlabel("axe des abscisses")
plt.grid(True)
#plt.xticks([min(x), max(x)],
#       [r'$0$', r'$T_a$'])
#plt.yticks([min(y), 0, max(y)],
```

```
#       [r'$-A$', r'$0$', r'$A$'])

plt.legend(loc='upper right')
plt.title('Titre du graphique', loc = 'center', y = -0.25)
plt.figtext(0,-0.1 ,'Commentaire du graphique',)

#plt.savefig("mafigure.pdf")

plt.show()
```

On peut aussi calculer la représentation en série de Fourier du signal numérisé à l'aide de la fonction np.fft.fft(). Cette fonction retourne la liste des amplitudes complexes de chaques harmoniques de la décomposition en série de Fourier.

Chaques harmoniques sont séparées d'une résolution spectrale δf donc les fréquences correspondantes sont $0, \delta f, 2\delta f, 3\delta f, \ldots$

Calculez puis tracez le spectre du signal d'entrée.

[ ]:
```
N = # récupérer ici le nombre de point
Te = # récupérer ici la période d'échantillonnage
Fe = # calculer la fréquence d'échantillonnage
deltaF = # calculer la résolution spectrale

f = np.arange(N)*deltaF
TF_e = np.fft.fft(e)

x = # ce que vous voulez tracer en abscisse
y = # ce que vous voulez tracer en ordonnée
plt.plot(x,y,'.', label="ma légende")
plt.ylabel('axe des ordonnées')
plt.xlabel("axe des abscisses")
plt.grid(True)
#plt.xticks([min(x), max(x)],
#       [r'$0$', r'$T_a$'])
#plt.yticks([min(y), 0, max(y)],
#       [r'$-A$', r'$0$', r'$A$'])

plt.legend(loc='upper right')
plt.title('Titre du graphique', loc = 'center', y = -0.25)
plt.figtext(0,-0.1 ,'Commentaire du graphique',)

#plt.savefig("mafigure.pdf")

plt.show()
```

Pour mieux visualiser l'échelle en fréquence on peut choisir de la tracer en échelle logarithmique via la commande plt.xscale('log')

[ ]:
```
plt.plot(x,y,'.', label="ma légende")
plt.ylabel('axe des ordonnées')
```

```
plt.xlabel("axe des abscisses")
plt.grid(True)
#plt.xticks([min(x), max(x)],
#       [r'$O$', r'$T_a$'])
#plt.yticks([min(y), 0, max(y)],
#       [r'$-A$', r'$O$', r'$A$'])

plt.legend(loc='upper right')
plt.title('Titre du graphique', loc = 'center', y = -0.25)
plt.figtext(0,-0.1 ,'Commentaire du graphique',)

plt.xscale('log') # commande pour afficher l'axe des x en échelle logarithmique

#plt.savefig("mafigure.pdf")

plt.show()
```

2 Conversion Numérique Analogique

Avant de faire un traitement numérique du signal on peut s'assurer que le signal est correctement numérisé en le ré-écoutant après une conversion numérique analogique.

Branchez vos écouteurs/casques sur l'ordinateur, l'objectif de cette partie sera alors d'écouter votre enregistrement en l'enregistrant dans un format .wav

Pour encoder le fichier nous aurons besoin des librairies suivantes.

[ ]:
```
import wave
import math
import binascii
import winsound
import struct
import os
import scipy.io.wavfile
import scipy
```

La fréquence d'échantillonnage d'un fichier .wav est fixé à 44,1kHz nous devons donc adapter l'enregistrement pour obtenir une suite d'échantillons avec cette fréquence d'échantillonnage.

Pour cela nous allons utiliser la fonction np.interp(twav, t, e) qui pour une liste d'échantillon e pris aux instants de la liste t, donne la liste des échantillons les plus proches pris aux instants twav.

[ ]:
```
Ta = # calculer ici la durée d'acquisition
Fwav = 44100
Twav = # calculer ici la durée entre deux points du signal .wav

Nwav = int(np.floor(Ta*Fwav))

twav = # calculer ici la liste des instants où sont pris les échantillons dans␣
 ↪le fichier .wav
```

```
ewav = np.interp(twav,t,e)
```

On peut s'assurer que l'interpolation est correcte en traçant les échantillons numérisés et les échantillons interpolés sur un même graphique.

[ ]:
```
x = # ce que vous voulez tracer en abscisse
y = # ce que vous voulez tracer en ordonnée
plt.plot(x,y,'o', label="ma légende")

x = # ce que vous voulez tracer en abscisse
y = # ce que vous voulez tracer en ordonnée
plt.plot(x,y,'.', label="ma légende")

plt.ylabel('axe des ordonnées')
plt.xlabel("axe des abscisses")
plt.grid(True)
#plt.xticks([min(x), max(x)],
#       [r'$0$', r'$T_a$'])
#plt.yticks([min(y), 0, max(y)],
#       [r'$-A$', r'$0$', r'$A$'])

plt.legend(loc='upper right')
plt.title('Titre du graphique', loc = 'center', y = -0.25)
plt.figtext(0,-0.1 ,'Commentaire du graphique',)

#plt.savefig("mafigure.pdf")

plt.show()
```

Chaque valeur de tension doit être maintenant convertie en un nombre entier correspondant à un nombre binaire. Le signal sera encodé sur 1 octect donc il faut encoder toutes les valeurs sur un nombre entier allant de 0 à 255.

La liste de valeur ewav doit donc être convertie en une liste de nombres entiers ebinaire allant de 0 à 255.

[ ]:
```
maximum = np.max(abs(ewav))

ebinaire = np.zeros(Nwav, dtype = np.int16)

for i in range(Nwav):
    ebinaire[i] = # calculer le nombre entier entre 0 et 255 correspondant à la␣
 ↪tension e[i]
```

On peut visualiser cette conversion en traçant ebinaire

[ ]:
```
x = # ce que vous voulez tracer en abscisse
y = # ce que vous voulez tracer en ordonnée
plt.plot(x,y,'.', label="ma légende")

plt.ylabel('axe des ordonnées')
```

```
plt.xlabel("axe des abscisses")
plt.grid(True)
#plt.xticks([min(x), max(x)],
#       [r'$0$', r'$T_a$'])
plt.yticks([0, 127, 255],
       [r'$0$', r'$127$', r'$255$'])

plt.legend(loc='upper right')
plt.title('Titre du graphique', loc = 'center', y = -0.25)
plt.figtext(0,-0.1 ,'Commentaire du graphique',)

#plt.savefig("mafigure.pdf")

plt.show()
```

Les lignes de commmandes suivantes écrivent le fichier .wav sur l'ordinateur

[ ]:
```
niveau = 1 # niveau sonore des hauts-parleur
nbCanal = 2    # stéreo
nbOctet = 1    # taille d'un échantillon : 1 octet = 8 bits

NomFichier = 'son.wav'
Monson = wave.open(NomFichier,'w')
parametres = (nbCanal,nbOctet,Fwav,Nwav,'NONE','not compressed')# tuple
Monson.setparams(parametres)    # création de l'en-tête (44 octets)

for i in range(Nwav):
    # canal gauche
    valG = wave.struct.pack('B',ebinaire[i])
    # canal droit
    valD = valG
    Monson.writeframes(valG + valD) # écriture frame

Monson.close()
```

Vous pouvez alors vous écoutez en lisant directement le fichier son.

3 Filtrage numérique

L'objectif dans cette partie est de calculer numériquement un signal s filtré à partir du signal numérisé e.

Nous avons établit en cours une relation de récurrence qui relie pour un filtre passe-bas RC les échantillons s aux échantillons e. Ecrire une fonction filtre_passe_bas(fc, Te, e) qui prend en argument fc la fréquence de coupure du filtre, Te la période d'échantillonnage de e, et e les échantillons et qui retourne s le signal numérique filtré.

[ ]:
```
def filtre_passe_bas(fc, Te, e):
    omegac = 2*np.pi*fc
    A = # coefficient de la suite de récurrence correspondant à un filtre RC
```

```
    B = # coefficient de la suite de récurrence correspondant à un filtre RC
    s = np.zeros(N)

    for i in range(N-1) :
        s[i+1] = A*s[i]+B*(e[i+1]+e[i]) # suite de récurrence correspondant à␣
↪l'éq. diff. d'un filtre RC

    return s
```

Tester votre filtre en comparant les spectres du signal numérisé, avant et après filtrage.

[ ]:
```
fc = # la fréquence de coupure de votre filtre
s = filtre_passe_bas(fc, Te, e)

TF_s = np.fft.fft(s)

x = # ce que vous voulez tracer en abscisse
y = # ce que vous voulez tracer en ordonnée
plt.plot(x,y,'.', label="ma légende")

x = # ce que vous voulez tracer en abscisse
y = # ce que vous voulez tracer en ordonnée
plt.plot(x,y,'.', label="ma légende")

plt.ylabel('axe des ordonnées')
plt.xlabel("axe des abscisses")
plt.grid(True)
#plt.xticks([min(x), max(x)],
#       [r'$0$', r'$T_a$'])
#plt.yticks([min(y), 0, max(y)],
#       [r'$-A$', r'$0$', r'$A$'])

plt.legend(loc='upper right')
plt.title('Titre du graphique', loc = 'center', y = -0.25)
plt.figtext(0,-0.1 ,'Commentaire du graphique',)
plt.xscale('log')

#plt.savefig("mafigure.pdf")

plt.show()
```

En reprenant la procédure de conversion numérique analogique pour le signal filtré s, vous pouvez maintenant écouter votre voix filtrée.

[ ]:
```

```

D'autres type de filtre sont possibles, vous pouvez concevoir un filtre numérique qui vous fait répéter votre enregistrement 2 fois plus vite. Ecoutez le résultat, que remarquez vous ?

[ ]:
```

```