Untitled3

October 10, 2019

1 Librairie

[22]:
```
import wave
import math
import binascii
import winsound
import struct
import os
import numpy as np
import scipy.io.wavfile
import scipy
import matplotlib.pyplot as plt
```

2 Signal d'entrée

2.1 définition des différents paramètres

$T_a >$ temps de réaction humain
$\$f_e \overline{2>\$frquencemaximaleaudible}$
f_a et f_b deux sons composant le signal

[23]:
```
Ta = 2 #durée d'acquisition en seconde du signal
Fe = 44100 #fréquence d'échantillonnage en Hz pour signaux audio
f_a = 220 # fréquence en Hz de l'harmonique a présente dans le signal
f_b = 220*(2**4) # fréquence en Hz de l'harmonique b présente dans le signal
niveau = 1 # niveau sonore des hauts-parleur
nbCanal = 2    # stéreo
nbOctet = 1    # taille d'un échantillon : 1 octet = 8 bits
```

2.2 calculs du nombre d'échantillon, période d'échantillonnage, résolution spectrale

$N = T_a \times F_e$ relation nombre d'échantillon, durée d'acquisition, fréquence d'échantillonnage
$T_e = \frac{1}{f_e}$ définition période d'échantillonnage
$\Delta f = \frac{1}{T_a} = \frac{f_e}{N}$ définition résolution spectrale

[24]:
```
N = int(Ta*Fe) # nombre d'échantillon
Te=1/Fe
```

```
deltaF=Fe/N
```

2.3 calcul trace temporelle et spectre du signal d'entrée

t est une liste de date telle que $t[i] = i \times T_e$

$v_e = A\sin(2\pi f_a t) + A\sin(2\pi f_b t)$

[25]:
```
t=np.linspace(0,Te*(N-1),N)
amp=niveau
ve = np.zeros(N)
for i in range(N) :
    ve[i] = amp*np.sin(2.0*np.pi*f_a*t[i])+amp*np.sin(2.0*np.pi*f_b*t[i])
```

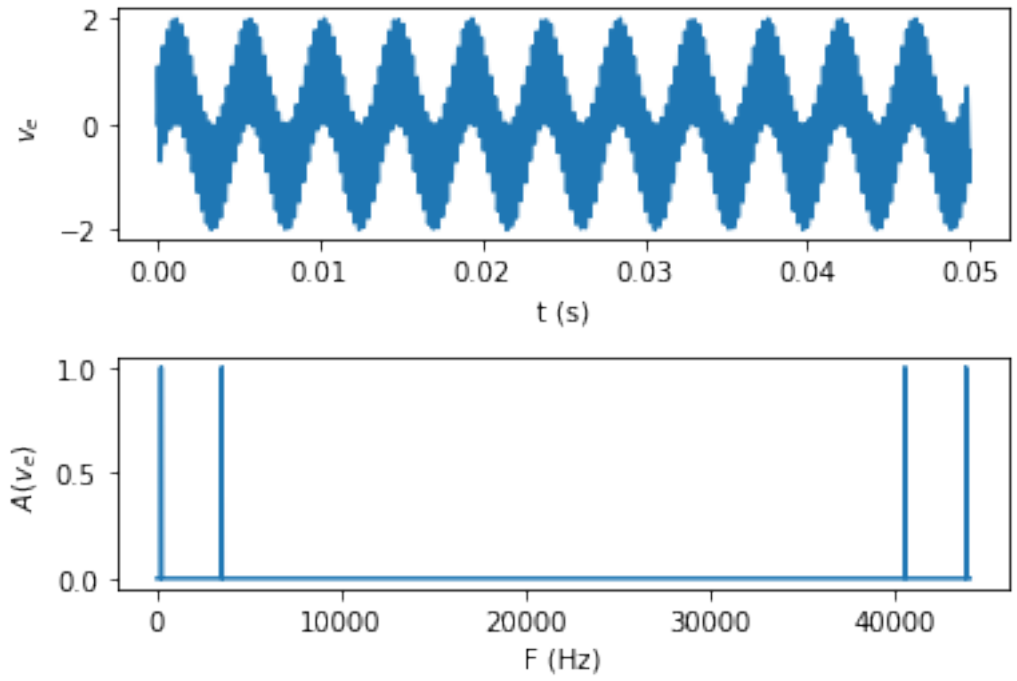
la série de Fourier de v_e se calcule avec l'algorithme FFT

[26]:
```
Se = np.zeros(N)
for i in range(0,N):
    Se[i] = ve[i]
vT, vF = np.arange(N)*Te,np.arange(N)*deltaF
TF_Se = np.fft.fft(Se)
```

affichage du signal temporel entre T_{min} et T_{max}, et entre f_{min} et f_{max}

[27]:
```
Tmin = 0
Tmax = 0.05
fmin = 0
fmax = 44.1*10**3

plt.subplots_adjust(hspace=.5)
plt.subplots_adjust(wspace=.5)
plt.subplot(211)
plt.plot(vT[int(Tmin/Te):int(Tmax/Te)],Se[int(Tmin/Te):int(Tmax/Te)])
plt.xlabel('t (s)')
plt.ylabel('$v_{e}$')
plt.subplot(212)
plt.plot(vF[int(fmin/deltaF):int(fmax/deltaF)],1/N*2*abs(TF_Se[int(fmin/deltaF):
 ↪int(fmax/deltaF)]))
plt.xlabel('F (Hz)')
plt.ylabel('$A(v_{e})$')
```

[27]:
```
Text(0, 0.5, '$A(v_{e})$')
```



2.4 encodage dans un fichier .wav

écriture dans le fichier .wav

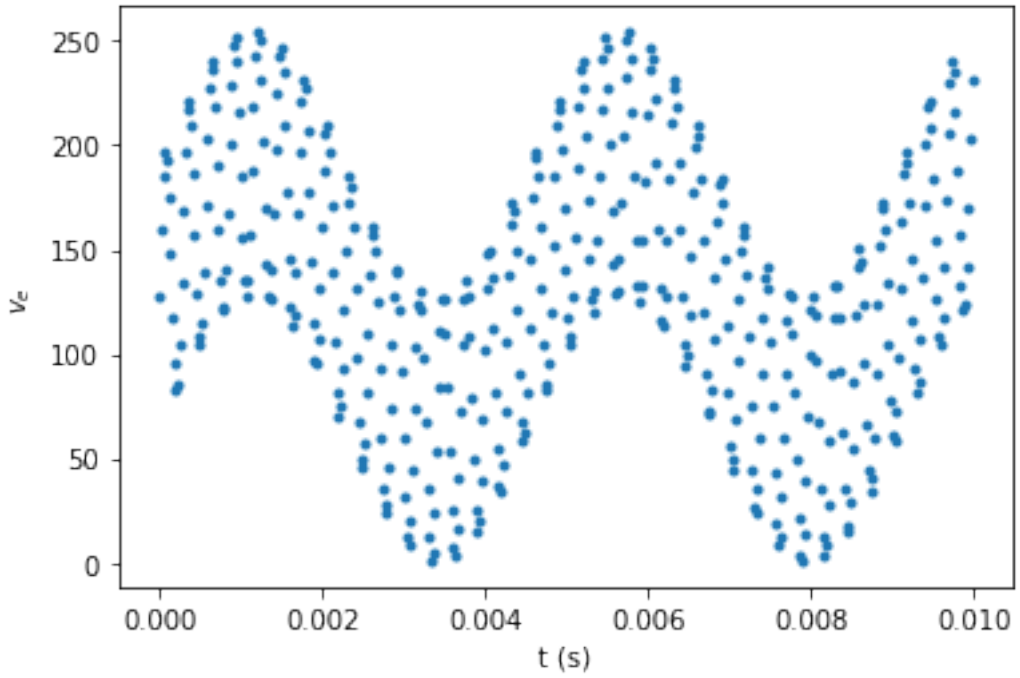
[28]:
```
NomFichier = 'son.wav'
Monson = wave.open(NomFichier,'w')
parametres = (nbCanal,nbOctet,Fe,N,'NONE','not compressed')# tuple
Monson.setparams(parametres)    # création de l'en-tête (44 octets)
for i in range(0,N):
    # canal gauche
    # 127.5 + 0.5 pour arrondir à l'entier le plus proche
    valG = wave.struct.pack('B',int(128.0 + 127.5*0.5*ve[i]))
    # canal droit
    valD = wave.struct.pack('B',int(128.0 + 127.5*0.5*ve[i]))
    Monson.writeframes(valG + valD) # écriture frame

Monson.close()
```

visualisation de l'effet de quantification sur 1 octet = 8 bits

[29]:
```
Tmin = 0
Tmax = 0.01

Se = np.zeros(N)
for i in range(0,N):
    Se[i] = int(128.0 + 127.5*0.5*ve[i])
vT = np.arange(N)*Te
```

```
plt.plot(vT[int(Tmin/Te):int(Tmax/Te)],Se[int(Tmin/Te):int(Tmax/Te)],'.')
plt.xlabel('t (s)')
plt.ylabel('$v_{e}$')
```

[29]:
```
Text(0, 0.5, '$v_{e}$')
```



2.5 lecture du fichier son

[32]:
```
Fichier = open(NomFichier,'rb')
data = Fichier.read()
tailleFichier = len(data)
print('\nTaille du fichier',NomFichier, ':', tailleFichier,'octets')
print("Lecture du contenu de l'en-tête (44 octets) :")
print(binascii.hexlify(data[0:44]))
print("Nombre d'octets de données :",tailleFichier - 44)
Fichier.close()
winsound.PlaySound('son.wav',winsound.SND_FILENAME)
```

```
Taille du fichier son.wav : 176444 octets
Lecture du contenu de l'en-tête (44 octets) :
b'5249464634b10200574156456666d74201000000001000200 44ac0000885801000200080064617 4
6110b10200'
Nombre d'octets de données : 176400
```

3 Filtrage numérique passe-bas

3.1 paramètre d'un filtre passe-bas

f_c fréquence de coupure en Hz

$\omega_c = 2\pi f_c$ pulsation de coupure en rad.s^{-1}

[33]:
```
fc = 220
omegac = 2*np.pi*fc
```

3.2 établissement de la relation de récurrence du filtre numérique

fonction de transfert filtre RC: $H = \frac{v_s}{v_e} = \frac{1}{1+j\frac{\omega}{\omega_c}}$

équation différentielle: $\frac{1}{\omega_c}\frac{dv_s}{dt} + v_s = v_e$

intégration entre iT_e et $(i+1)T_e$: $\frac{1}{\omega_c}\int_{iT_e}^{(i+1)T_e}\frac{1}{\omega_c}\frac{dv_s}{dt}dt + \int_{iT_e}^{(i+1)T_e} v_s dt = \int_{iT_e}^{(i+1)T_e} v_e dt$

donc $\frac{v_s((i+1)T_e)-v_s(iT_e)}{\omega_c} + \frac{v_s((i+1)T_e)+v_s(iT_e)}{2T_e} = \frac{v_e((i+1)T_e)+v_e(iT_e)}{2T_e}$

donc $v_s((i+1)T_e) = Av_s(iT_e) + B(v_e((i+1)T_e)+v_e(iT_e))$ avec $A = \frac{2-\omega_c T_e}{2+\omega_c T_e}$ et $B = \frac{\omega_c T_e}{2+\omega_c T_e}$

[34]:
```
A = (2.0-omegac*Te)/(2.0+omegac*Te)
B = omegac*Te/(2.0+omegac*Te)

vs = np.zeros(N)
for i in range(N-1) :
    vs[i+1] = A*vs[i]+B*(ve[i+1]+ve[i])
```

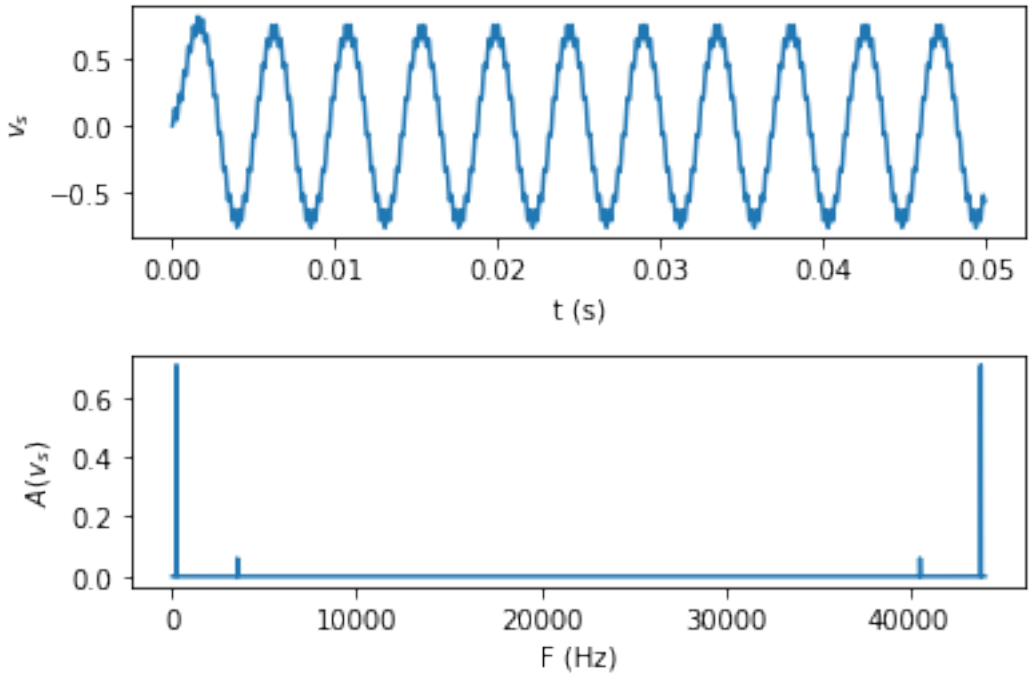
4 Signal de sortie

4.1 trace temporelle et spectre du signal de sortie

[35]:
```
Tmin = 0
Tmax = 0.05
fmin = 0
fmax = 44.1*10**3

Se = np.zeros(N)
for i in range(0,N):
    Se[i] = vs[i]
vT, vF = np.arange(N)*Te,np.arange(N)*deltaF
TF_Se = np.fft.fft(Se)
plt.subplots_adjust(hspace=.5)
plt.subplots_adjust(wspace=.5)
plt.subplot(211)
plt.plot(vT[int(Tmin/Te):int(Tmax/Te)],Se[int(Tmin/Te):int(Tmax/Te)])
plt.xlabel('t (s)')
plt.ylabel('$v_{s}$')
plt.subplot(212)
```

```
plt.plot(vF[int(fmin/deltaF):int(fmax/deltaF)],1/N*2*abs(TF_Se[int(fmin/deltaF):
 ↪int(fmax/deltaF)]))
plt.xlabel('F (Hz)')
plt.ylabel('$A(v_{s})$')
```

[35]:
```
Text(0, 0.5, '$A(v_{s})$')
```



4.2 encodage dans un fichier .wav

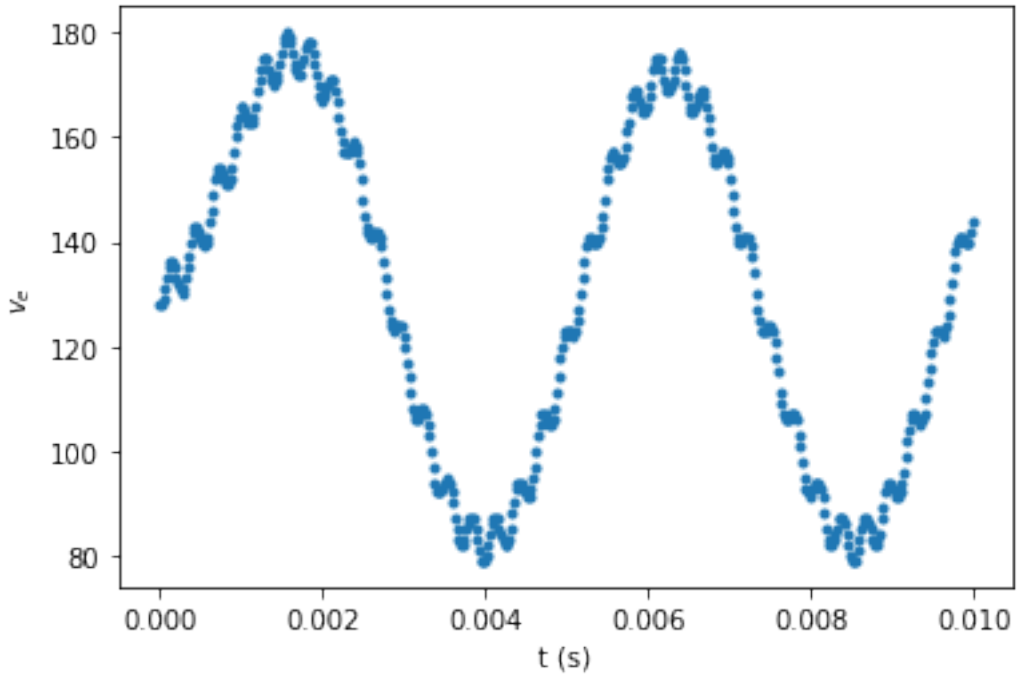
écriture dans le fichier .wav

[36]:
```
NomFichier = 'son.wav'
Monson = wave.open(NomFichier,'w')
parametres = (nbCanal,nbOctet,Fe,N,'NONE','not compressed')# tuple
Monson.setparams(parametres)    # création de l'en-tête (44 octets)
for i in range(0,N):
    # canal gauche
    # 127.5 + 0.5 pour arrondir à l'entier le plus proche
    valG = wave.struct.pack('B',int(128.0 + 127.5*0.5*vs[i]))
    # canal droit
    valD = wave.struct.pack('B',int(128.0 + 127.5*0.5*vs[i]))
    Monson.writeframes(valG + valD) # écriture frame

Monson.close()
```

visualisation de l'effet de quantification sur 1 octet = 8 bits

[37]:
```
Tmin = 0
Tmax = 0.01

Se = np.zeros(N)
for i in range(0,N):
    Se[i] = int(128.0 + 127.5*0.5*vs[i])
vT = np.arange(N)*Te
plt.plot(vT[int(Tmin/Te):int(Tmax/Te)],Se[int(Tmin/Te):int(Tmax/Te)],'.')
plt.xlabel('t (s)')
plt.ylabel('$v_{e}$')
```

[37]:
```
Text(0, 0.5, '$v_{e}$')
```



4.3 lecture du fichier son

[39]:
```
Fichier = open(NomFichier,'rb')
data = Fichier.read()
tailleFichier = len(data)
print('\nTaille du fichier',NomFichier, ':', tailleFichier,'octets')
print("Lecture du contenu de l'en-tête (44 octets) :")
print(binascii.hexlify(data[0:44]))
print("Nombre d'octets de données :",tailleFichier - 44)
Fichier.close()
winsound.PlaySound('son.wav',winsound.SND_FILENAME)
```

```
Taille du fichier son.wav : 176444 octets
Lecture du contenu de l'en-tête (44 octets) :
b'5249464634b1020057415645666d74201000000001000200044ac0000885801000200080064617
46110b10200'
Nombre d'octets de données : 176400
```

5 Application à un enregistrement

[40]:
```
Fe, data = scipy.io.wavfile.read('0283_avant.wav')
print('fréquence d échantillonage', Fe, 'Hz')
v_e = data[:,1]
```

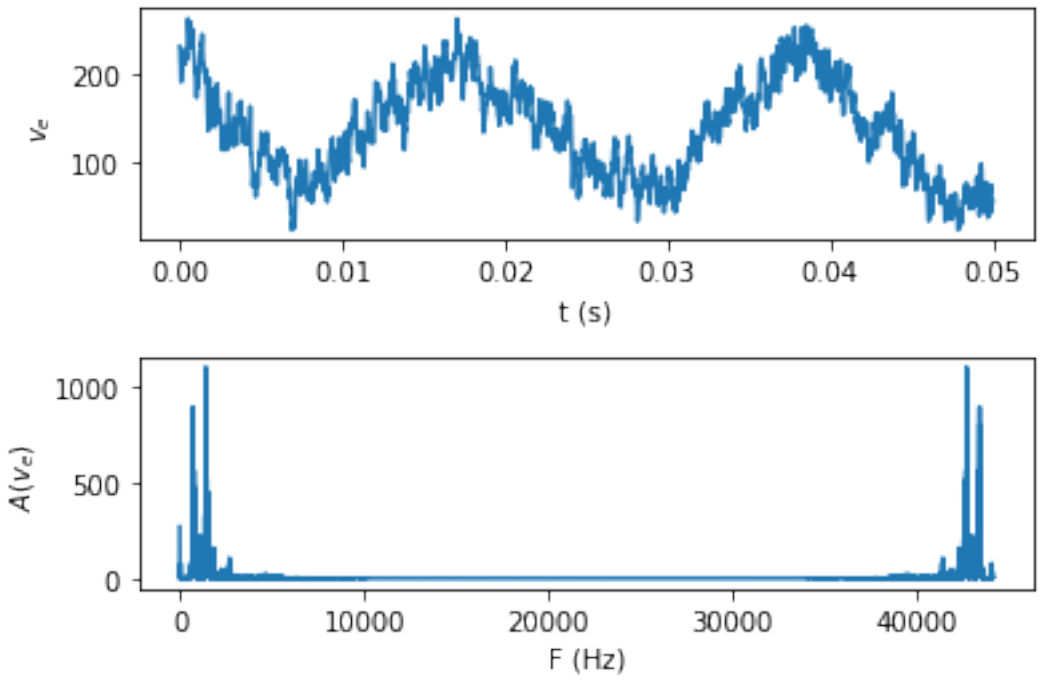
```
fréquence d échantillonage 44100 Hz
```

[41]:
```
N = v_e.size
Te=1/Fe
deltaF=Fe/N
```

[42]:
```
Se = np.zeros(N)
for i in range(0,N):
    Se[i] = v_e[i]
vT, vF = np.arange(N)*Te,np.arange(N)*deltaF
TF_Se = np.fft.fft(Se)
```

[43]:
```
Tmin = 0
Tmax = 0.05
fmin = 0
fmax = 44.1*10**3

plt.subplots_adjust(hspace=.5)
plt.subplots_adjust(wspace=.5)
plt.subplot(211)
plt.plot(vT[int(Tmin/Te):int(Tmax/Te)],Se[int(Tmin/Te):int(Tmax/Te)])
plt.xlabel('t (s)')
plt.ylabel('$v_{e}$')
plt.subplot(212)
plt.plot(vF[int(fmin/deltaF):int(fmax/deltaF)],1/N*2*abs(TF_Se[int(fmin/deltaF):
 ↪int(fmax/deltaF)]))
plt.xlabel('F (Hz)')
plt.ylabel('$A(v_{e})$')
```

[43]:
```
Text(0, 0.5, '$A(v_{e})$')
```



[48]:
```
fc = 200
omegac = 2*np.pi*fc
```

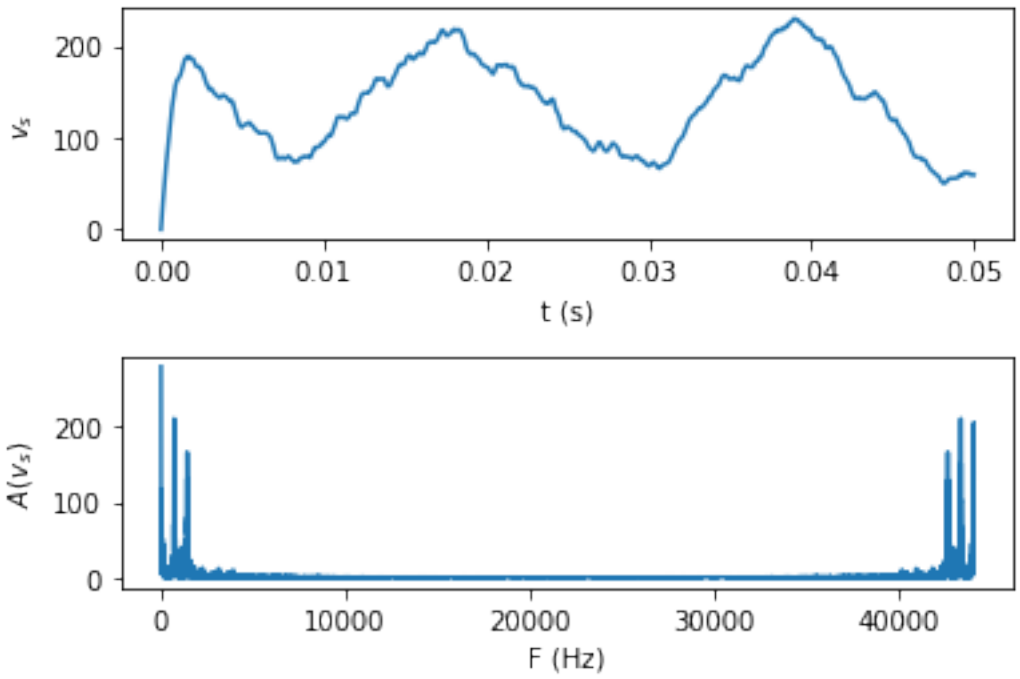
[49]:
```
A = (2.0-omegac*Te)/(2.0+omegac*Te)
B = omegac*Te/(2.0+omegac*Te)

v_s = np.zeros(N)
for i in range(N-1) :
    v_s[i+1] = A*v_s[i]+B*(v_e[i+1]+v_e[i])
```

```
C:\Users\remib\AppData\Local\Continuum\anaconda3\lib\site-
packages\ipykernel_launcher.py:6: RuntimeWarning: overflow encountered in
short_scalars
```

[50]:
```
Tmin = 0
Tmax = 0.05
fmin = 0
fmax = 44.1*10**3

Se = np.zeros(N)
for i in range(0,N):
    Se[i] = v_s[i]
vT, vF = np.arange(N)*Te,np.arange(N)*deltaF
TF_Se = np.fft.fft(Se)
plt.subplots_adjust(hspace=.5)
```

```
plt.subplots_adjust(wspace=.5)
plt.subplot(211)
plt.plot(vT[int(Tmin/Te):int(Tmax/Te)],Se[int(Tmin/Te):int(Tmax/Te)])
plt.xlabel('t (s)')
plt.ylabel('$v_{s}$')
plt.subplot(212)
plt.plot(vF[int(fmin/deltaF):int(fmax/deltaF)],1/N*2*abs(TF_Se[int(fmin/deltaF):
 ↪int(fmax/deltaF)]))
plt.xlabel('F (Hz)')
plt.ylabel('$A(v_{s})$')
```

[50]:
```
Text(0, 0.5, '$A(v_{s})$')
```



[52]:
```
NomFichier = '0283_apres.wav'
Monson = wave.open(NomFichier,'w')
parametres = (nbCanal,nbOctet,Fe,N,'NONE','not compressed')# tuple
Monson.setparams(parametres)     # création de l'en-tête (44 octets)
for i in range(0,N):
    # canal gauche
    # 127.5 + 0.5 pour arrondir à l'entier le plus proche
    valGbis = wave.struct.pack('B',int(128.0 + 127.5*(2**(-16))*v_s[i]))
    # canal droit
    valDbis = valGbis
    Monson.writeframes(valGbis + valDbis) # écriture frame

Monson.close()
```

[ ]: