In [1]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
```

Densité spectrale du bruit d'une machine à laver

In [2]:
```
from scipy.io.wavfile import read
samplerate, amplitude = read('data/machine_a_laver.wav')
```

In [ ]:
```
# question 2
print(samplerate)
delta_t = 1/samplerate
N = len(amplitude)
print(N)
T = A compléter
print(T)
```

In [ ]:
```
# question 2 le tracé de la courbe temporelle
t = np.arange(N)*A compléter #Le temps

plt.plot(A compléter)
plt.xlabel('Temps [s]')
plt.ylabel('Amplitude')
```

In [5]:
```
# Calcul de la DSP
from scipy.signal import periodogram
freq, psd = periodogram(amplitude, samplerate)
```

In [ ]:
```
# Question 3
mask = (freq>=10) & (freq<50) # Fréquence de 10 à 50 Hz
plt.loglog(freq[mask], psd[mask])
plt.grid(which='both')

mask_pic = np.abs(freq-20)<1.5 # Fréquence autour de 20Hz

plt.loglog(freq[mask& ~mask_pic], psd[mask& ~mask_pic])
plt.loglog(freq[mask_pic], psd[mask_pic])
```

In [ ]:
```
# Question 5

Px = A compléter #Puissance du signal
print(Px)

Delta_f = A compléter # incrément de fréquence de la DSP
print(np.sum(psd)*A compléter)#Puissance du signal via la DSP
```

In [ ]:
```
# Question 6

print('Signal', np.sum(psd[A compléter]*A compléter))
print('Bruit', np.sum(psd[A compléter])*A compléter)
```

Filtre en Python

In [ ]:
```
# Création d'un signal porte

samplerate = 44100
delta_t = 1/samplerate
signal = np.zeros(samplerate*3)
signal[samplerate:samplerate*2] = 1
t = np.arange(len(signal))*delta_t
plt.plot(t, signal)
```

In [ ]:
```
# Question 1

import numpy as np
signal_tilde = np.fft.rfft(signal)
signal_2 = np.fft.irfft(signal_tilde)
plt.plot(t, signal_2)
```

In [ ]:
```
# Question 2

np.fft.rfftfreq(len(signal), d=delta_t)
# Commenter le résultat, freq max, pas de fréquence.
```

In [ ]:
```
def pass_bas(signal, f_c, samplerate=44100):
    signal_tilde = np.fft.rfft(signal)
    freqs = np.fft.rfftfreq(len(signal), 1/samplerate)
    H = 1/(1+1J*(freqs/f_c))
    signal_2 = np.fft.irfft(H*signal_tilde)
    return signal_2
```

In [ ]:
```
def pass_haut(signal, f_c, samplerate=44100):
    signal_tilde = np.fft.rfft(signal)
    freqs = np.fft.rfftfreq(len(signal), 1/samplerate)
    H = 1J*(freqs/f_c)/(1+1J*(freqs/f_c))
    signal_2 = np.fft.irfft(H*signal_tilde)
    return signal_2
```

In [ ]:
```
# Question 3 : Filtre passe bas sur signal porte

signal_filtre = pass_bas(signal, f_c= A compléter)
plt.plot(t, signal_filtre)
# Commenter
```

In [ ]:
```
# Question 4 : Filtre passe haut sur signal porte

signal_filtre = pass_haut(signal, f_c= A compléter)
plt.plot(t, signal_filtre)
# Commenter
```

In [ ]:
```
# Question 5 : Filtre passe bas sur machine à laver

from scipy.io.wavfile import read
samplerate, amplitude = read('data/machine_a_laver.wav')

amplitude_filtree = pass_bas(amplitude, A compléter, samplerate=samplerate)
```

In [ ]:
```
# Question 5 suite: Tracé dernière seconde du signal filtré

t = np.arange(N)*delta_t

plt.plot(t[A compléter], amplitude[A compléter])
plt.plot(t[A compléter], amplitude_filtree[A compléter])
plt.xlabel('Temps [s]')
plt.ylabel('Amplitude')
```

Onde gravitationnelle

In [21]:
```
# Question 1

from readligo import loaddata
filename_H1 = 'data/H-H1_LOSC_4_V1-1126259446-32.hdf5'
strain_H1, time_H1, chan_dict_H1 = loaddata(filename_H1, 'H1')

filename_L1 = 'data/L-L1_LOSC_4_V1-1126259446-32.hdf5'
strain_L1, time_L1, chan_dict_L1 = loaddata(filename_L1, 'L1')

# Question 2
time = time_H1 #temps
print(len(time))
dt = A compléter  #Pas de temps
print(dt)
samplerate = int(1/dt) # échantilonnage
```

In [ ]:
```
# Question 3 Tracé le signal sur 10 secondes autour de l'évenement

tevent = 1126259462         # Mon Sep 14 09:50:45 GMT 2015
deltat = A compléter        # seconds around the event

# index into the strain time series for this time interval:
mask = ((time_H1 >= tevent-deltat) & (time_H1 < tevent+deltat))

# index into the strain time series for this time interval:
mask = ((time_H1 >= tevent-deltat) & (time_H1 < tevent+deltat))

plt.plot(time_H1[mask] - tevent, strain_H1[mask])
```

In [ ]:
```
#Question 4 DSP sur signal entier avec fonction périodogramme

from scipy.signal import periodogram
f_p, psd_p = periodogram(strain_H1, samplerate)

plt.loglog(f_p[A compléter:], psd_p[A compléter:])#10 Hz à 2 kHz
```

In [ ]:
```
# Question 4 DSP méthode de Welch

from scipy.signal import welch
f, psd = welch(strain_H1, samplerate, nperseg=samplerate)

plt.loglog(f[A compléter:], psd[A compléter:])#10 Hz à 2 kHz
# Commenter les différences avec la DSP calculer avec le périodogramme
```

In [ ]:
```
# Question 5

print(psd[f==100]) #Valeur de PSD à 100 Hz

# Filtrage entre 30 et 300 Hz en utilisant un passe haut et un passe bas
strain_H1_filt = pass_haut(pass_bas(strain_H1, f_c=A compléter, samplerate=1/dt), f_c=A compléter, samplerate=1

# plot +- 5 seconds around the event:
tevent = 1126259462      # Mon Sep 14 09:50:45 GMT 2015
deltat = 5               # seconds around the event

# index into the strain time series for this time interval:
mask = ((time_H1 >= tevent-deltat) & (time_H1 < tevent+deltat))
plt.plot(time_H1[mask] - tevent, strain_H1_filt[mask])
# Comparer la courbe obtenue à celle obtenue sans filtrage

# Variance du signal filtré et non filtré
print(np.var(strain_H1_filt))
print(np.var(strain_H1))
```

In [ ]:
```
# Question 6: Filtrage des signaux parasites, on enlève les pics parasites de la DSP correspondant aux oscillat
# vu sur le signal temporel

from scipy.interpolate import interp1d

def whiten(strain, dt):
    freqs_welch, psd_welch = welch(strain, fs=1/dt, nperseg=int(1/dt))
    interp_psd = interp1d(freqs_welch, psd_welch)

    strain_tilde = np.fft.rfft(strain)
    N = len(strain)
    freqs = np.fft.rfftfreq(N, dt)

    gain = 1 / np.sqrt(interp_psd(freqs))
    gain = gain/gain.max()
    white_strain_tilde = strain_tilde * gain
    white_strain = np.fft.irfft(white_strain_tilde)
    return white_strain

strain_H1_whiten = whiten(strain_H1, dt)
strain_L1_whiten = whiten(strain_L1, dt)

# plot +- 5 seconds around the event:
tevent = 1126259462         # Mon Sep 14 09:50:45 GMT 2015
deltat = A compléter        # seconds around the event

# index into the strain time series for this time interval:
mask = ((time_H1 >= tevent-deltat) & (time_H1 < tevent+deltat))

plt.plot(strain_H1_whiten[mask])

# Variance du signal sans les oscillations parasites
print(np.var(strain_H1_whiten))
```

In [ ]:
```
# Question 7 : Sans oscillations parasites + Filtres! sur les deux détecteurs

strain_H1_whiten_filt = pass_haut(pass_bas(A compléter))
strain_L1_whiten_filt = pass_haut(pass_bas(A compléter))

# plot +- 0.5 seconds around the event:
tevent = 1126259462         # Mon Sep 14 09:50:45 GMT 2015
deltat = 0.5                # seconds around the event

# index into the strain time series for this time interval:
mask = ((time_H1 >= tevent-deltat) & (time_H1 < tevent+deltat))

# Tracé des deux signaux filtrés
plt.subplot(2, 1, 1)

plt.plot(time_H1[mask] - tevent, strain_H1_whiten_filt[mask])
plt.xlim(0.3, 0.5)
plt.subplot(2, 1, 2)

delay = 0 # Délais entre les deux signaux, A compléter, mettre à 0 au départ
plt.plot(time_L1[mask] - tevent + delay, -strain_L1_whiten_filt[mask])
plt.xlim(0.3, 0.5)
```

In [ ]:
```
## Question 8
# Correlation avec un délai de 7ms

start = int(len(time)/2 - 2.5*samplerate)
stop = int(len(time)/2 + 2.5*samplerate)
delai = int(7E-3*samplerate)
correlation = -strain_L1_whiten_filt[start:stop] * strain_H1_whiten_filt[start+delai:stop+delai]
correlation_filt = pass_bas(correlation, f_c=10, samplerate=samplerate)
plt.plot(time_L1[start:stop]-tevent, correlation_filt)
plt.xlim(0, 1)
```