Exercice 8

8.1 À la recherche des ondes gravitationnelles

Le 11 février 2016, la collaboration LIGO-VIRGO a annoncé la découverte de la première onde gravitationnelle[a]. Le signal `GW150914` est compatible avec les prédictions de la relativité générale et correspondrait à la chute en spirale puis à la fusion d'une paire de trous noirs. C'est la première détection d'ondes gravitationnelles directe.

Figure 8.1 – Vues aériennes des deux interféromètres de l'expérience LIGO.

La détection a été faite le 14 septembre 2015 vers 9h50 sur les deux sites américains de l'expérience LIGO. Chaque site, en Louisiane et dans l'état de Washington, héberge deux interféromètres qui mesurent l'amplitude de la déformation de l'espace-temps aussi appelée *strain,* suite au passage d'une onde gravitationnelle.
Dans cet exercice, on propose une première analyse des signaux de cet événement `GW150914`.

8.1.1 Prérequis

Les fichiers de données LIGO sont sérialisés dans le format binaire `hdf5`. Vous aurez donc besoin dans un premier temps d'installer la librairie `h5py` *via* `pip`. Il faudra dans un second temps télécharger les fichiers de données suivant:

- signal `H1` issu du détecteur Hanford,
- signal `L1` issu du détecteur Livingstone.

Enfin, l'expérience LIGO fournit un script python permettant la lecture des fichiers de données. Après l'avoir téléchargé, placez ce fichier dans le répertoire courant où votre code d'analyse se trouvera.

8.1.2 Premiers coups d'oeil aux données

À l'aide de la fonction `loaddata` du script `readligo.py`, charger en mémoire les données (temps, *strain*) de chacun des deux interféromètres. Représenter la déformation (*strain*) en fonction du temps pour chaque détecteur puis sachant que le temps de l'événement `GW150914` est $t = 1126259462.44$ s, représenter les données à $\pm$ 5 secondes autour du temps de référence.
Calculer la moyenne et la variance du signal autour du signal `GW150914`. En comparant ces valeurs à celles issues d'une région temporelle vierge de signal, en déduire qu'à ce stade de l'analyse, le signal `GW150914` est compatible avec du bruit.

[a] The LIGO Sientific Collaboration, the Virgo Collaboration, *"Observation of gravitational waves from a binary black hole merger"*, Physical Review Letters, vol. 116, no 6, 2016 arxiv 1602.0387.

8.1.3 Analyse en fréquence du signal

On peut également représenter les données dans le domaine de Fourier afin d'avoir une idée plus précise du contenu en fréquence. On peut en particulier représenter la densité spectrale d'amplitude (*Amplitude Spectral Density* ou *ASD*) qui correspond à la racine carrée de la densité spectrale de puissance (*Power Spectral Density* ou *PSD*). Cette dernière est définie comme le carré du module de la transformée de Fourier, divisé par la largeur de bande spectrale. Cette quantité est représentative de la répartition fréquentielle de la puissance d'un signal suivant les fréquences qui le composent.
Sans rentrer dans plus de détail mathématique, on utilisera pour estimer la *PSD* la fonction `welch` basée sur la méthode éponyme[b] disponible dans la librairie `scipy.signal`. Sachant que la fréquence d'échantillonage du signal est de 4096 Hz et que le nombre d'échantillon pour la transformée de Fourier correspond à 4 fois la fréquence d'échantillonage. Calculer puis représenter en échelle log – log la densité spectrale d'amplitude des deux signaux. Limiter le domaine de fréquence à l'intervalle [20 – 2000] Hz : en-deçà de 20 Hz, les données sont pas correctement étalonnées; au-delà de 2000 Hz, les données ne peuvent contenir d'information en fréquence au-delà de la fréquence de Nyquist qui correspond à la moitié du taux d'échantillonnage soit 2048 Hz.
Indication: les seuls paramètres que l'on fournira à la fonction `welch` *sont la fréquence d'échantillonage* `fs` *et la longueur de chaque segment* `nperseg = 4*fs`.

8.1.4 Spectrogrammes des signaux

On peut également visualiser la puissance du signal à l'aide d'un spectrogramme dans l'espace fréquence–temps. À l'aide de la fonction `specgram` de `matplotlib`, représenter le spectrogramme pour chaque signaux à ± 5 secondes autour du temps supposé de l'événement. On utilisera sans plus de justification, les paramètres suivants pour le spectrogramme :

```
fs = 4096 # Hz
NFFT = int(fs/8)
noverlap = int(NFFT*15/16)
window = np.blackman(NFFT)
```

Les lignes spectrales présentes à 500, 1000 et 1500 Hz correspondent aux harmoniques des modes "violon" des fibres supportant les miroirs des interféromètres.

8.1.5 "Blanchiment" des données

Afin de mettre en évidence le signal de l'onde gravitationnelle, il convient alors de "blanchir" les données c'est-à-dire de diviser, dans le domaine Fourier, le signal par l'amplitude du bruit. Dans un premier temps, interpoler la *PSD* de chaque signal afin de pouvoir sur-échantillonner les signaux. Utiliser ensuite la fonction ci-dessous appliquée à chaque déformation-*strain* (`dt` correspond à l'écart temporel entre deux points successifs du signal).

```
def whiten(strain, interp_psd, dt):
    Nt = len(strain)
    freqs = np.fft.rfftfreq(Nt, dt)
    freqs1 = np.linspace(0, 2048., Nt/2+1)

    # whitening: transform to freq domain, divide by asd, then transform back,
    # taking care to get normalization right.
    hf = np.fft.rfft(strain)
    norm = 1./np.sqrt(1./(dt*2))
    white_hf = hf / np.sqrt(interp_psd(freqs)) * norm
    white_ht = np.fft.irfft(white_hf, n=Nt)
    return white_ht
```

Finalement, représenter les données "blanchies" sur l'intervalle temporel d'intérêt.

correction

8.1.6 Importation des libraires scientifiques

```
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
```

[b] Welch, P. D., *"The use of Fast Fourier Transform for the estimation of power spectra: A method based on time averaging over short, modified periodograms"*, IEEE Transactions on Audio and Electroacoustics, 1967

8.1.7 Importation de la librairie pour la lecture des données LIGO

```
import readligo as rl
```

8.1.8 Déclaration des propriétés de l'événement

Nom de l'événement

```
event_name = "GW150914"
```

Fichiers associés à l'événement

Les détections ont été réalisées par deux détecteurs :

- H1 correspond au détecteur d'Hanford situé dans l'état de Washington,
- L1 correspond au détecteur de Livingston situé dans l'état de Louisiane.

```
fn_H1 = "data/H-H1_LOSC_4_V2-1126259446-32.hdf5"
fn_L1 = "data/L-L1_LOSC_4_V2-1126259446-32.hdf5"
```

8.1.9 Lecture des événements

La fonction `loaddata` de la librairie `readligo` permet de récupérer la déformée (*strain*) en fonction du temps.

```
strain_H1, time_H1, _ = rl.loaddata(fn_H1)
strain_L1, time_L1, _ = rl.loaddata(fn_L1)
```

8.1.10 Premier aperçu des données temporelles

```
tevent = 1126259462.44
t = time_H1
deltat = 5 #sec
idxt = np.where(np.abs(t-tevent) < deltat)# & (t < tevent+dt))

plt.plot(t, strain_H1, label="H1 strain")
plt.plot(t, strain_L1, label="L1 strain")
# plt.plot(t[idxt]-tevent, strain_H1[idxt], label="H1 strain")
# plt.plot(t[idxt]-tevent, strain_L1[idxt], label="L1 strain")
plt.title("Événement " + event_name)
plt.xlabel("temps (s) depuis " + str(tevent))
plt.ylabel("strain [no unit]")
plt.legend()

np.mean(strain_H1[idxt]), np.std(strain_H1[idxt])
```

8.1.11 Distribution du signal en fréquence

Plotting these data in the Fourier domain gives us an idea of the frequency content of the data. A way to visualize the frequency content of the data is to plot the amplitude spectral density, ASD.
The ASDs are the square root of the power spectral densities (PSDs), which are averages of the square of the fast fourier transforms (FFTs) of the data.
They are an estimate of the "strain-equivalent noise" of the detectors versus frequency, which limit the ability of the detectors to identify GW signals.
They are in units of strain/$\sqrt{\text{Hz}}$. So, if you want to know the root-mean-square (rms) strain noise in a frequency band, integrate (sum) the squares of the ASD over that band, then take the square root.

Déclaration du taux d'échantillonage du signal et de la transformée de Fourrier

```
fs = 4096
nfft = 4*fs
from scipy.signal import welch as psd
freqs, Pxx_H1 = psd(strain_H1, fs = fs, nperseg = nfft)
freqs, Pxx_L1 = psd(strain_L1, fs = fs, nperseg = nfft)

plt.figure(figsize=(10,8))
plt.loglog(freqs, np.sqrt(Pxx_H1), label="H1 strain")
plt.loglog(freqs, np.sqrt(Pxx_L1), label="L1 strain")
plt.grid()
plt.xlim(20, 2000.)
plt.ylim(1e-24, 1e-19)
plt.legend()
```

Interpolation des PSD

```
from scipy.interpolate import interp1d
psd_H1 = interp1d(freqs, Pxx_H1)
psd_L1 = interp1d(freqs, Pxx_L1)
```

8.1.12 Whitening

From the ASD above, we can see that the data are very strongly "colored" - noise fluctuations are much larger at low and high frequencies and near spectral lines, reaching a roughly flat ("white") minimum in the band around 80 to 300 Hz.
We can "whiten" the data (dividing it by the noise amplitude spectrum, in the fourier domain), suppressing the extra noise at low frequencies and at the spectral lines, to better see the weak signals in the most sensitive band.
Whitening is always one of the first steps in astrophysical data analysis (searches, parameter estimation). Whitening requires no prior knowledge of spectral lines, etc; only the data are needed.
To get rid of remaining high frequency noise, we will also bandpass the data.
The resulting time series is no longer in units of strain; now in units of "sigmas" away from the mean.
We will plot the whitened strain data, along with the signal template, after the matched filtering section, below.

```
def whiten(strain, interp_psd, dt):
    Nt = len(strain)
    freqs = np.fft.rfftfreq(Nt, dt)
    freqs1 = np.linspace(0,2048.,Nt/2+1)

    # whitening: transform to freq domain, divide by asd, then transform back,
    # taking care to get normalization right.
    hf = np.fft.rfft(strain)
    norm = 1./np.sqrt(1./(dt*2))
    white_hf = hf / np.sqrt(interp_psd(freqs)) * norm
    white_ht = np.fft.irfft(white_hf, n=Nt)
    return white_ht

# now whiten the data from H1 and L1, and the template (use H1 PSD):
dt = time_H1[1] - time_H1[0]
strain_H1_whiten = whiten(strain_H1, psd_H1, dt)
strain_L1_whiten = whiten(strain_L1, psd_L1, dt)
```

8.1.13 Spectrogrammes

correction

```
deltat = 5 #sec
idxt = np.where(np.abs(t-tevent) < deltat)# & (t < tevent+dt))
# pick a shorter FTT time interval, like 1/8 of a selfecond:
NFFT = int(fs/8)
# and with a lot of overlap, to resolve short-time features:
NOVL = int(NFFT*15/16)
# and choose a window that minimizes "spectral leakage"
# (https://en.wikipedia.org/wiki/Spectral_leakage)
window = np.blackman(NFFT)

# the right colormap is all-important! See:
# http://matplotlib.org/examples/color/colormaps_reference.html
# viridis seems to be the best for our purposes, but it's new; if you don't have it, you can settle for ocean.
#spec_cmap='viridis'
# spec_cmap='ocean'

# Plot the H1 spectrogram:
plt.figure(figsize=(10,6))
plt.specgram(strain_H1[idxt], NFFT=NFFT, Fs=fs, window=window,
             noverlap=NOVL, xextent=[-deltat,deltat])
plt.xlabel('time (s) since '+str(tevent))
plt.ylabel('Frequency (Hz)')
plt.colorbar()
plt.axis([-deltat, deltat, 0, 2000])
plt.title('aLIGO H1 strain data near ' + event_name)
```