Analyse spectrale des séries temporelles de mesures dans l'espace interplanétaire

TP A2 - Guillaume BOGOPOLSKY

5 octobre 2018

TP A2 — Guillaume Bogopolsky
Physics of Plasma and Fusion Master's Degree
Université Paris-Saclay
Document auto-généré grâce à la commande *nbviewer* de *Jupyter*, et retouché manuellement. La version HTML est disponible ici.

1 Introduction

L'analyse spectrale d'un signal consiste à exhiber ses fréquences caractéristiques grâce à la transformée de Fourier en représentant dans l'espace des fréquences sa *power spectral density* (PSD). Toutefois, comme nous le verrons, il est impossible de distinguer la répartition temporelle des différents modes, contenue dans les phases. Pour récupérer cette information, nous allons utiliser une transformée en ondelettes. Cela nous permettra notamment d'étudier des données spatiales comme les traversées de régions très différentes (ici, la traversée de la magnétosphère terrestre par la mission CLUSTER).

Au cours de cet TP, nous allons étudier tout d'abord la transformée de Fourier discrète et la transformée en ondelettes de Morlet sur des signaux synthétiques, et étudier leurs différences, puis nous appliquerons ces méthodes à des données réelles.

Ce compte-rendu a été réalisé grâce aux Jupyter Notebook, il vous est donc possible de récupérer ses fichiers dans le dépôt GitHub et de l'éxecuter sur votre machine.

2 Eléments théoriques

Commençons par définir les outils mathématiques que nous allons utiliser dans cette partie. ### La transformée de Fourier discrète

Tout d'abord, définissons la transformée de Fourier discrète (DFT, *discrete Fourier transform*). Soit une série temporelle $u[j] = u(t_j)$ avec $t_j = j\Delta t = jT/N$ où T est le temps d'enregistrement, N le nombre de mesures et $j = 0, 1, \ldots, N-1$ l'indice des points de mesure temporelle. Nous allons décomposer ce signal sur un nombre fini des fréquences définies comme $f_n = n/T$ avec $n = 0, 1, \ldots, N-1$ l'indice des fréquences.
La DFT est alors définie comme suit :

$$\hat{u}[n] = \frac{1}{N} \sum_{j=0}^{N-1} u[j] e^{-2i\pi \frac{nj}{N}}$$

La transformée de Fourier inverse, c'est-à-dire l'opération qui à partir du spectre en fréquence reconstruit le signal temporel, est ainsi :

$$u[j] = \sum_{j=0}^{N-1} \hat{u}[n] e^{2i\pi \frac{nj}{N}}$$

Une telle transformée a des propriétés intéressantes : pour un signal réel, il est possible de montrer que :

$$\hat{u}[N-n] = \hat{u}^*[n]$$

Preuve :

$$\hat{u}[N-n] = \frac{1}{N} \sum_{j=0}^{N-1} u[j] e^{-2i\pi \frac{(N-n)j}{N}} = \frac{1}{N} \sum_{j=0}^{N-1} u[j] e^{-2i\pi j} e^{2i\pi \frac{nj}{N}} = \frac{1}{N} \sum_{j=0}^{N-1} u[j] e^{2i\pi \frac{nj}{N}} = \hat{u}^*[n]$$

Nous pouvons aussi démontrer que cette transformation conserve l'énergie :

$$\sum_{n=0}^{N-1} |\hat{u}[n]|^2 = \sum_{n=0}^{N-1} \hat{u}[n]\hat{u}[n]^* = \frac{1}{N^2} \sum_{n=0}^{N-1} \left(\sum_{j=0}^{N-1} u[j] e^{-2i\pi \frac{nj}{N}} \right) \left(\sum_{k=0}^{N-1} u[k] e^{2i\pi \frac{nk}{N}} \right)$$

En séparant le produit de sommes en une somme de sommes, il vient :

$$\sum_{n=0}^{N-1} |\hat{u}[n]|^2 = \frac{1}{N^2} \sum_{n=0}^{N-1} \left(\sum_{\substack{j=0 \\ k=0 \\ j=k}}^{N-1} u[j]u[k] + \sum_{\substack{j=0 \\ k=0 \\ j\neq k}}^{N-1} u[j]u[k] e^{2i\pi \frac{n(k-j)}{N}} \right)$$

Inverser les sommations permet d'écrire :

$$\sum_{n=0}^{N-1} |\hat{u}[n]|^2 = \frac{1}{N^2} \sum_{j=0}^{N-1} u[j]^2 \sum_{n=0}^{N-1} 1 + \sum_{\substack{j=0 \\ k=0 \\ j\neq k}}^{N-1} u[j]u[k] \sum_{n=0}^{N-1} e^{2i\pi \frac{n(k-j)}{N}}$$

où l'on reconnaît la somme des racines de l'unité qui vaut zéro. D'où :

$$\sum_{n=0}^{N-1} |\hat{u}[n]|^2 = \frac{1}{N} \sum_{j=0}^{N-1} |u[j]|^2$$

C'est le théorème de Parseval-Plancherel.

Nous avons parlé plus haut de la PSD (*power spectral density*). Elle se définit : $S[n] = 2T\,|\hat{u}[n]|$ avec $n = 0, 1, \ldots, N/2 - 1$, car la deuxième moitié du spectre est identique à la première (comme nous l'avons montré précédemment). $S[n]$ s'exprime en $\mathrm{V}^2\ \mathrm{Hz}^{-1}$. Nous pouvons également réécrire le théorème de Plancherel avec la PSD :

$$\sum_{n=0}^{N-1} S[n] = \frac{2T}{N} \sum_{j=0}^{N-1} |u[j]|^2$$

2.1 La transformée en ondelettes de Morlet

Soit l'ondelette de Morlet définit de la façon suivante par rapport au temps t :

$$\psi_0(t) = \pi^{-1/4} e^{-i\omega_0 t} e^{-t^2/2}$$

La dilatation et la translation de cette ondelette permet de définir par convolution la transformée en ondelettes :

$$\mathcal{W}(\tau, t) = \sum_{j=0}^{N-1} u(t_j) \psi^*((t_j - t)/\tau)$$

avec t_j la position autour de laquelle on effectue la convolution dans une fenêtre Δt, et τ un paramètre de dilatation temporelle ou échelle temporelle. Nous choisissons cette échelle de sorte qu'une relation simple existe avec les fréquences de Fourier : $1/f \simeq \tau$ en posant $\omega_0 = 6$.
Le carré du module du coefficient d'ondelette $|\mathcal{W}(\tau, t)|^2$ représentant le *quantum* d'énergie des fluctuations de $u(t)$ sur la surface $\Delta T \times \Delta\tau$ autour d'un moment t à l'échelle τ, nous pouvons tracer le scalogramme de la transformée sur un graphe à deux dimensions (t, τ). Il est aussi possible de remonter à la puissance spectrale équivalente à la PSD via le spectre des fluctuations en intégrant sur la variable temporelle :

$$S_w = 2\delta t \mathcal{W}(f)^2 = \frac{2\delta t}{N} \sum_{i=0}^{N-1} |\mathcal{W}(f, t_j)|^2$$

Vous trouverez plus de détails ici.

Maintenant que nos transformées sont définies, nous allons les implémenter puis les mettre en applications sur des signaux synthétiques.

3 Application à des signaux synthétiques

3.1 Transformée de Fourier

Nous implémentons les fonctions dans un fichier annexe *function.py*. La DFT est d'abord réalisée de façon intuitive avec des sommes :

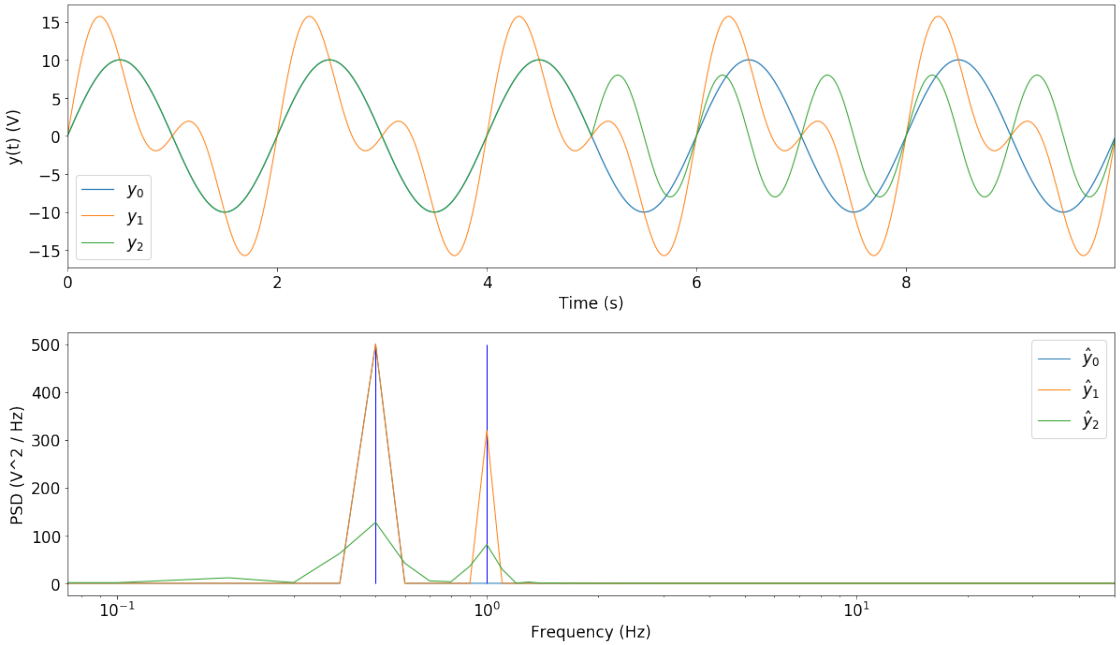
```
def dft(data):
    """
    Discrete Fourier transform of real data.
    Input : Time serie of size N
    Output : Time serie of size N//2+1
    """
    N = len(data)
    output = np.zeros(N, dtype=complex)
    for n in range(N):
        for j in range(N):
            output[n] += data[j] * np.exp(-2 * np.pi * 1j * n * j / N)
    return output[:int(N/2) + 1] / N
```

Une fonction pour générer la liste des fréquences de Fourier est également pratique :

```
def dftfreq(N, T):
    """
    Returns frequency coordinates for the DFT functions.
    Input: Size of the time series, and time step.
    Output: Frequency serie of size N//2+1
    """
    return np.arange(N//2 + 1) / T
```

Appliquons cette fonction aux signaux synthétiques $y_0(t)$, $y_1(t)$ et $y_2(t)$ définis également dans le même fichier :

```
In [2]: import numpy as np
        import matplotlib.pyplot as plt
        import function as fct
        plt.rcParams['figure.figsize'] = (17,10)
        plt.rcParams['font.size'] = 17
        # Constants
        N = int(1e3)
        T = 10
        dt = T/N
        A0, f0 = 10, 0.5
        A1, f1 = 8, 1

In [3]: t = np.arange(N) * dt
        y0, y1, y2 = fct.y0(t, A0, f0), fct.y1(t, A0, A1, f0, f1), fct.y2(t, A0, A1, f0, f1, T
        PSD0 = 2 * T * np.abs(fct.fdft(y0))**2
        PSD1 = 2 * T * np.abs(fct.fdft(y1))**2
        PSD2 = 2 * T * np.abs(fct.fdft(y2))**2
        freq = fct.dftfreq(N, T)

In [5]: plt.subplot(211)
        plt.plot(t, y0, linewidth=1.2, label=r'$y_0$')
        plt.plot(t, y1, linewidth=1, label=r'$y_1$')
        plt.plot(t, y2, linewidth=1, label=r'$y_2$')
        plt.xlabel('Time (s)')
        plt.ylabel('y(t) (V)')
        plt.xlim((np.min(t), np.max(t)))
        plt.legend()

        plt.subplot(212)
        plt.semilogx(freq, PSD0, linewidth=1, label=r'$\hat{y}_0$')
        plt.semilogx(freq, PSD1, linewidth=1, label=r'$\hat{y}_1$')
        plt.semilogx(freq, PSD2, linewidth=1, label=r'$\hat{y}_2$')
        plt.vlines((f0, f1), 0, 500, linewidth=1, color='blue')
        plt.xlim((np.abs(np.min(freq)), np.max(freq)))
        plt.xlabel('Frequency (Hz)')
        plt.ylabel('PSD (V^2 / Hz)')
        plt.legend()
        plt.tight_layout()
```



Nous constatons tout d'abord que seule la moitié de la gamme de fréquence est couverte : en effet, la symétrie $\hat{u}[N-n] = \hat{u}^*[n]$ nous confirme que ces deux moitiés sont identiques pour des données réelles. Il n'est donc pas nécessaire de conserver le reste. De plus, la fréquence maximale couverte est la fréquence de Nyquist, donnée par le théorème de Shannon-Nyquist : $f_{max} = f_{echant}/2 = 1/(2dt)$.
Sur les spectres, les fréquences f_0 et f_1 sont bien visibles, mais une chose est très claire : en ne se basant que sur la transformée de Fourier, il est impossible de distinguer les signaux y_1 et y_2, qui diffèrent par le déphasage entre les modes associés à ces deux fréquences. Cela sera possible avec transformée en ondelettes.

Nous avons également confirmé la précision de notre DFT à l'aide des transformées implémentées dans `numpy.fft` (cf. TPA2-donnees-synthetiques.ipynb), avec succès.
Par ailleurs, vous avez peut-être remarqué précédemment l'utilisation de la fonction `fdft` au lieu de `dft`. En effet, la DFT implémentée comme nous l'avons décrit est très lente à cause de l'utilisation des doubles boucles de Python. Afin d'optimiser la vitesse d'exécution, nous mettons à profit la vitesse des opérations matricielles codées en C du paquet Numpy en implémentant `fdft` de la manière suivante :

```
def fdft(data):
    """
    Fast discrete Fourier transform of real data with Numpy and matrix operations.
    Input: Time serie of size N
    Output: Time serie of size N//2+1
    """
    N = len(data)
    exp = np.exp(-2 * np.pi * 1j / N * np.dot(np.arange(N)[:,np.newaxis], np.arange(N)[np.newa
    output = np.dot(data, exp) / N
    return output[:N//2+1]
```

Dans cette version, nous créons une matrice contenant tous les termes de l'exponentielle complexe en se basant sur la matrice $M_{n,j} = nj$ créée par produit matriciel, les axes étant choisi de façon adéquate. Il suffit ensuite de la multiplier avec notre vecteur de données pour obtenir la série fréquentielle. L'optimisation n'est pas encore optimale (la sortie étant tronquée, la routine calcule deux fois trop de valeurs), mais elle nous permet tout de même d'augmenter la vitesse d'exécution un facteur 48 !

```
In [6]: %%timeit
        test = 2*T*np.abs(fct.dft(y0))**2

3.97 s ś 148 ms per loop (mean ś std. dev. of 7 runs, 1 loop each)


In [7]: %%timeit
        test = 2*T*np.abs(fct.fdft(y0))**2

81.4 ms ś 579 ţs per loop (mean ś std. dev. of 7 runs, 10 loops each)
```

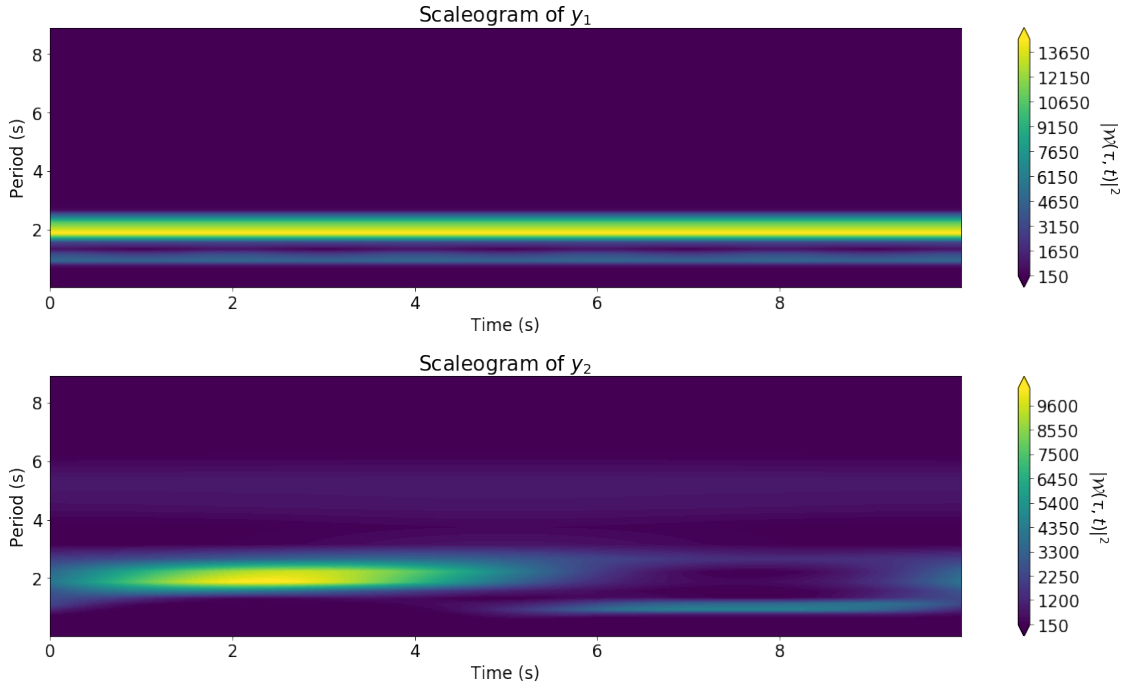
3.2 Transformée en ondelettes de Morlet

Afin de réaliser la transformée en ondelettes, nous allons utiliser une routine développée par C. Torrence, G. Compo pour leur article et adaptée en Python par E. Predybaylo, trouvable sur leur dépôt.

```
In [8]: import waveletFunctions as wav
```

Nous nous intéressons aux transformées des signaux y_1 et y_2, indiscernables par simple transformée de Fourier.

```
In [9]: wave1, period1, scale1, coi1 = wav.wavelet(y1, dt)
        wave2, period2, scale2, coi2 = wav.wavelet(y2, dt)

In [10]: plt.subplot(211)
         contour_plot = plt.contourf(t, period1, np.abs(wave1)**2, 100, extend='both')
         plt.xlabel('Time (s)')
         plt.ylabel(r'Period (s)')
         plt.title(r'Scaleogram of $y_{1}$')
         cbar = plt.colorbar(contour_plot)
         cbar.set_label(r'$\left| \mathcal{W}(\tau,t)\right| ^{2}$', rotation=270, labelpad=30)

         plt.subplot(212)
         contour_plot = plt.contourf(t, period2, np.abs(wave2)**2, 100, extend='both')
         plt.xlabel('Time (s)')
         plt.ylabel(r'Period (s)')
         plt.title(r'Scaleogram of $y_{2}$')
         cbar = plt.colorbar(contour_plot)
         cbar.set_label(r'$\left| \mathcal{W}(\tau,t)\right| ^{2}$', rotation=270, labelpad=30)
         plt.tight_layout()
```



Nous remarquons clairement la différence entre les deux signaux : dans le premier cas, les deux modes f_0 et f_1 existent sur toute la durée du signal, tandis que dans le deuxième cas, le mode $f_1 = 1,0$ Hz succède au mode $f_0 = 0,5$ Hz à partir de $T/2 = 5$ s.

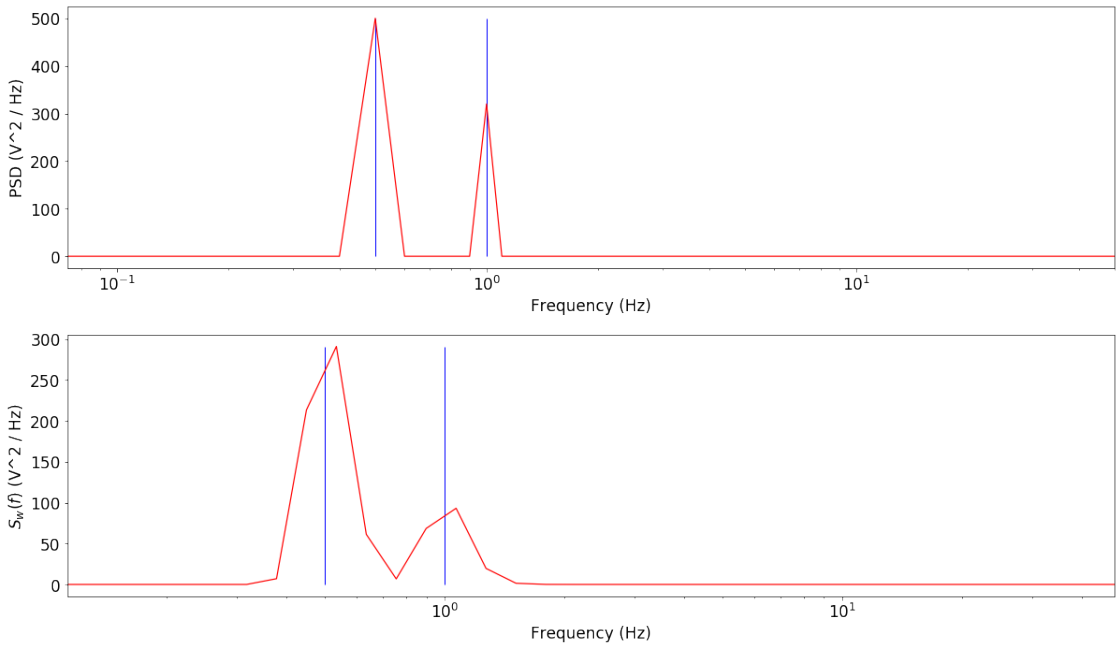
```
In [11]: print('T0 = ', 1/f0, ' T1 = ', 1/f1)

T0 =  2.0  T1 =  1.0
```

3.3 Comparaison entre les deux méthodes

Enfin, comparons pour le signal y_1 le spectre de Fourier et le spectre des wavelets obtenu est calculant la grandeur S_w définie précédemment :

```
In [12]: spWy1 = 2*dt*np.sum(np.abs(wave1)**2, axis=1)/wave1.shape[1]

In [14]: plt.subplot(211)
         plt.semilogx(freq, PSD1, linewidth=1.2, color='red')
         plt.xlim((np.abs(np.min(freq)), np.max(freq)))
         plt.vlines([f0, f1], np.min(PSD1), np.max(PSD1), color='blue', linewidth=1)
         plt.xlabel('Frequency (Hz)')
         plt.ylabel('PSD (V^2 / Hz)')

         plt.subplot(212)
         plt.semilogx(1/period1, spWy1, linewidth=1.2, color='red')
         plt.vlines([f0, f1], np.min(spWy1), np.max(spWy1), color='blue', linewidth=1)
         plt.xlim(np.abs(np.min(1/period1)), np.max(1/period1))
```

```
plt.xlabel('Frequency (Hz)')
plt.ylabel(r'$ S_{w}(f)$ (V^2 / Hz)')
plt.tight_layout()
```




```
In [15]: print(freq.shape, period1.shape)

(501,) (36,)
```

Nous constatons que le spectre en fréquence de Fourier est plus précis et défini en fréquence que le spectre des wavelets. Nous nous attendions à un tel résultats : en effet, afin de conserver la résolution temporelle, la transformée en ondelettes sacrifie sa résolution fréquentielle.
Ainsi, pour la transformée de Fourier, Δf a pour valeur :

```
In [16]: print('Delta_f = ', freq[42] - freq[41], 'Hz')

Delta_f =  0.10000000000000053 Hz
```

Alors que pour la transformée en ondelettes, cette valeur est variable, comme nous pouvons le voir dans l'array des coordonnées fréquentielles (obtenue avec $1/f \simeq T$) :

```
In [17]: print(1/period1)

[48.40066546 40.69994608 34.22443876 28.77920787 24.20033273 20.34997304
 17.11221938 14.38960393 12.10016636 10.17498652  8.55610969  7.19480197
  6.05008318  5.08749326  4.27805485  3.59740098  3.02504159  2.54374663
```

```
  2.13902742  1.79870049  1.5125208   1.27187332  1.06951371  0.89935025
  0.7562604   0.63593666  0.53475686  0.44967512  0.3781302   0.31796833
  0.26737843  0.22483756  0.1890651   0.15898416  0.13368921  0.11241878]
```

Comme la plage de fréquences parcourue par les deux transformées reste proche :

```
In [18]: print('Fourier :   ', freq[-1] - freq[0], 'Hz')
         print('Ondelettes :', 1/period1[0] - 1/period1[-1], 'Hz')

Fourier :    50.0 Hz
Ondelettes : 48.28824667898603 Hz
```

Soit en moyenne $\Delta f = 48,29/36 = 1,34$ Hz, donc un ordre de grandeur supérieur au pas de la transformée de Fourier.
C'est la conséquence du principe d'incertitude de Heisenberg, qui relie la résolution en fréquence Δf (ou en échelle temporelle $\Delta\tau = 1/\Delta f$) et la résolution temporelle ΔT :

$$\Delta f \Delta T \sim const.$$

Nous allons maintenant mettre en application ces transformations sur des signaux réels.

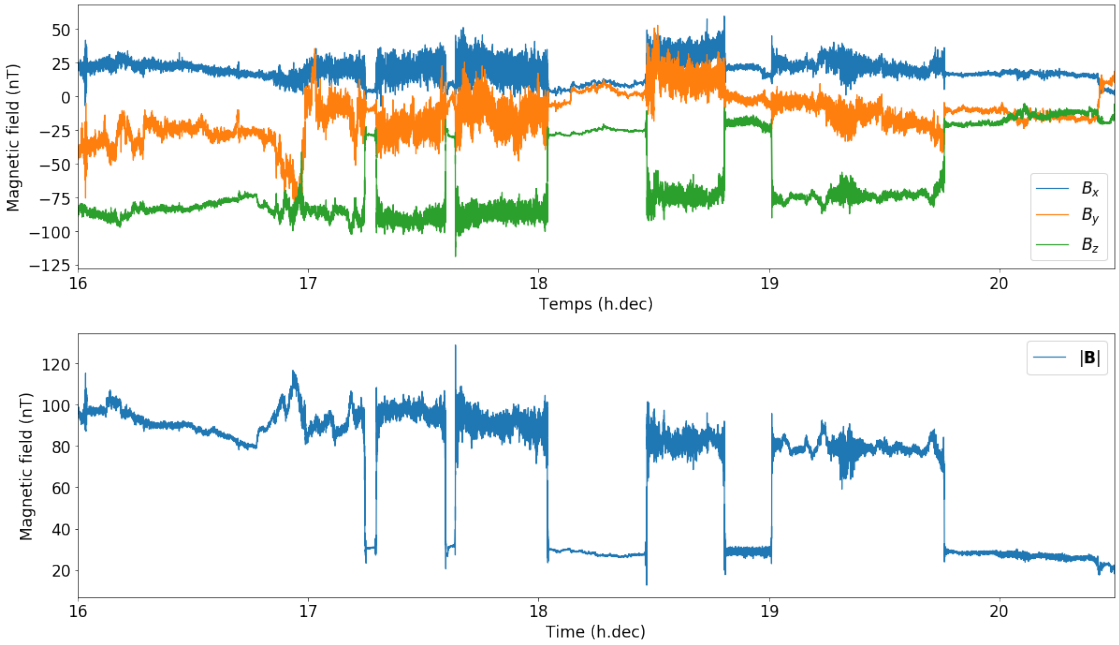
4 Application à des signaux réels

Nous allons maintenant étudier des données spatiales provenant de la mission CLUSTER lancée en 2000. Elle est constituée de quatre satellites qui ont pour missions d'étudier l'interaction entre le vent solaire et la magnétoshpère terrestre en mesurant notamment le champ magnétique en trois dimensions grâce à la répartition spatiale des quatres satellites.
Aujourd'hui, nous nous intéressons au relevé des mesures de champs magnétique du 31 mars 2001 qui présente de multiples traversées de la zone de choc entre le vent solaire et la magnétosphère terrestre.

En utilisant un module de `scipy` pour lire le fichier en `.sav`, nous pouvons tracer ces données et afin d'avoir une meilleure idée de ce qu'il se passe, calculons le module du vecteur champ magnétique :

```
In [19]: from scipy.io import readsav
         data = readsav('data_choc.sav', verbose=False)   # Reading data from .sav file

In [20]: t = data['t_sc']
         Bx = data['bx']
         By = data['by']
         Bz = data['bz']
         dt = t[42] - t[41]     # Timestep
         K = 3600               # Constant for seconds to decimal hours conversion
         mod = np.sqrt(Bx**2 + By**2 + Bz**2)     # Computing the modulus

In [21]: plt.subplot(211)
         plt.plot(t/K, Bx, label=r'$B_x$', linewidth=1.2)
```

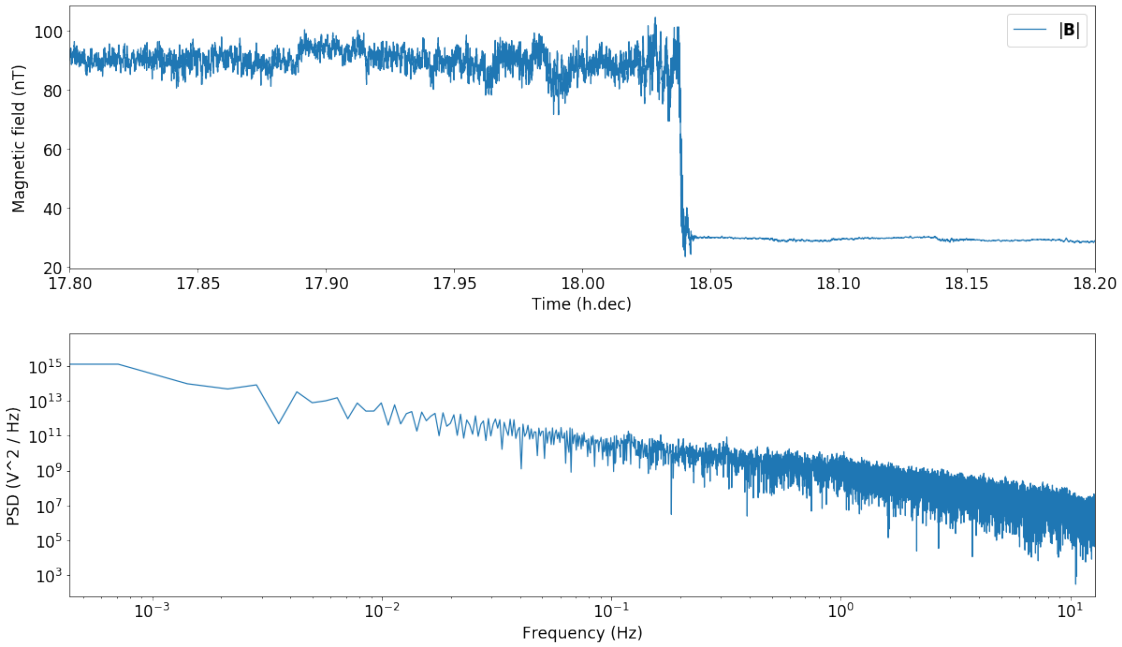
```
plt.plot(t/K, By, label=r'$B_y$', linewidth=1.2)
plt.plot(t/K, Bz, label=r'$B_z$', linewidth=1.2)
plt.xlim((np.min(t/K), np.max(t/K)))
plt.xlabel('Temps (h.dec)')
plt.ylabel('Magnetic field (nT)')
plt.legend()

plt.subplot(212)
plt.plot(t/K, mod, label=r'$\left| \mathbf{B}\right|$', linewidth=1.2)
plt.xlim((np.min(t/K), np.max(t/K)))
plt.xlabel('Time (h.dec)')
plt.ylabel('Magnetic field (nT)')
plt.legend()
plt.tight_layout()
```



Nous remarquons de multiples franchissements de la zone de choc marqués par les discontinuités : en effet, cette zone est située à l'équilibre entre la magnétosphère et le vent solaire dont la vitesse varie, donc sa position varie également. Elle correspond à une onde de choc non collisionnelle entre le vent solaire qui est supersonique ($v_{sw} \simeq 500$ km/s alors que la vitesse du son dans le milieu est de $v_{m,son} \simeq 70$ km/s) et le milieu de la magnétogaine terrestre où il devient subsonique. Ainsi, comme sa vitesse diminue, l'énergie cinétique est convertie en énergie magnétique, d'où l'augmentation du module du champ magnétique dans la magnétospère de $|\mathbf{B}| \simeq 30$ nT à $|\mathbf{B}| \simeq 90$ nT.

Dans cette étude, nous allons nous concentrer sur un unique franchissement de la zone de choc, par exemple entre 17,8 h et 18,2 h. Réalisons ensuite la transformée de Fourier de cet intervalle (en utilisant une `fft` pour des raisons de rapidité et d'usage de mémoire avec `fdft`).

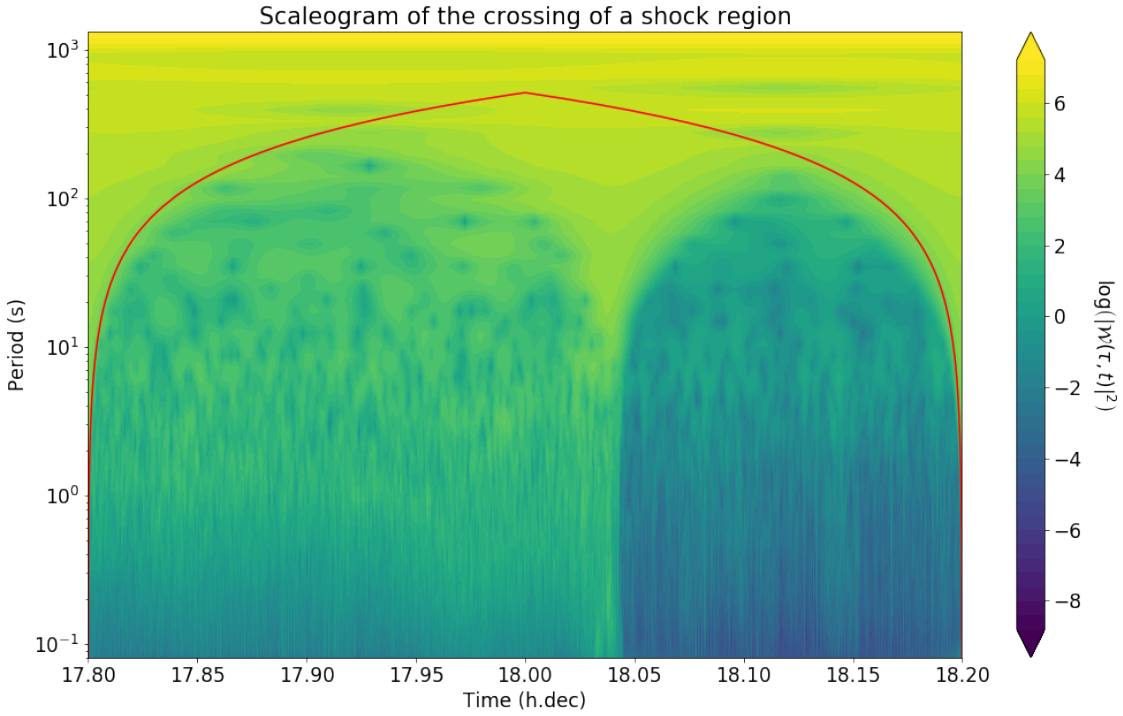
```
In [22]: mask = ((t < 18.2*K) & (t > 17.8*K))        # Mask to select the correct data
         T = t[mask][-1] - t[mask][0]                # Duration of the selected range
         PSDa0 = 2 * T * np.abs(np.fft.rfft(mod[mask]))**2
         freqa0 = np.fft.rfftfreq(len(mod[mask]), dt)

In [23]: plt.subplot(211)
         plt.plot(t[mask]/K, mod[mask], label=r'$\left| \mathbf{B}\right|$', linewidth=1.2)
         plt.xlim((np.min(t[mask]/K), np.max(t[mask]/K)))
         plt.xlabel('Time (h.dec)')
         plt.ylabel('Magnetic field (nT)')
         plt.legend()

         plt.subplot(212)
         plt.loglog(freqa0, PSDa0, linewidth=1.2)
         plt.xlim((np.min(freqa0), np.max(freqa0)))
         plt.xlabel('Frequency (Hz)')
         plt.ylabel('PSD (V^2 / Hz)')
         plt.tight_layout()
```



Le pas de temps limite la gamme de fréquence couverte à $\Delta f = 12,8$ Hz par théorème de Shannon-Nyquist, et le bruit de la transformée nous empêche de voir les structures fréquentielles clairement. Appliquons la transformée en ondelettes sur cette plage de données.

```
In [24]: wavea0, perioda0, scalea0, coia0 = wav.wavelet(mod[mask], dt)

In [25]: contour_plot = plt.contourf(t[mask]/K, perioda0, np.log10(np.abs(wavea0)**2), 42, ext
         plt.plot(t[mask]/K, coia0, color='red')
```

```
plt.ylim((np.min(perioda0), np.max(perioda0)))
plt.yscale('log')
plt.xlabel('Time (h.dec)')
plt.ylabel(r'Period (s)')
plt.title(r'Scaleogram of the crossing of a shock region')
cbar = plt.colorbar(contour_plot)
cbar.set_label(r'$\log \left( \left| \mathcal{W}(\tau,t)\right| ^{2}\right)$', rotatio
plt.show()
```



Dans ce scalogramme, nous voyons apparaître des structures verticales, significative de phénomènes localisés dans le temps mettant en jeu toutes les échelles de période (contrairement aux ondes de la partie précédente, ayant une période fixée). Nous observons aussi l'existence de valeurs non physiques en-dehors du *cone of influence* (COI) tracé en rouge sur la figure ci-dessus. Comme nous nous y attendions, la gamme d'échelles temporelles résolues est plus petite que pour la transformée de Fourier (par principe d'incertitude d'Heisenberg) :

$$\Delta f_{wavelet} = f_{max} - f_{min} = \frac{1}{T_{min}} - \frac{1}{T_{max}} = 12,39 \text{ Hz}$$

Pour obtenir ce résultat, nous utilisons le fait que $1/f \simeq T$ dans notre cas.

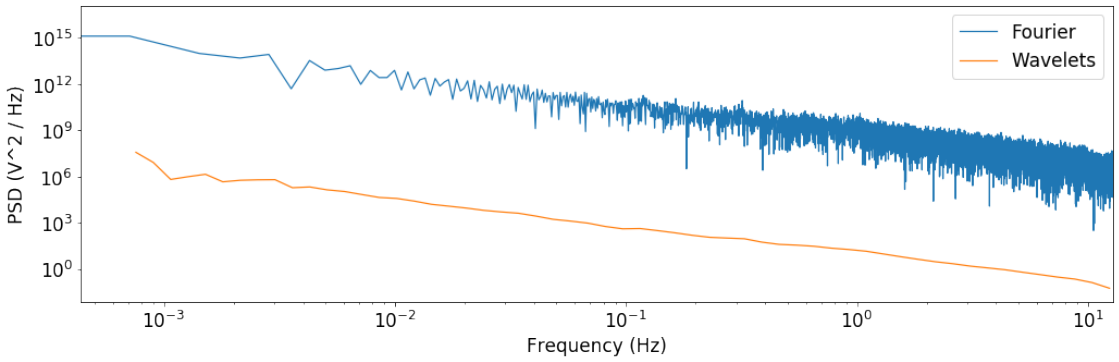
```
In [26]: print(1/perioda0[0] - 1/perioda0[-1])

12.3898140972904
```

Nous observons bien une densité d'énergie plus grande dans la zone de gauche qui correspond au champ magnétique dans la magnétogaine que dans la zone du vent solaire. Entre ces deux zones, l'onde de choc est caractérisée par sa localisation temporelle et la mise en jeu de toutes les échelles temporelles du plasma. Tout l'intérêt de la transformée en ondelettes est ici exploité : nous pouvons localiser dans le temps les différents modes temporels présents.

Comparons enfin le spectre de Fourier avec le spectre des wavelets :

```
In [27]: spWa0 = np.sum(np.abs(wavea0)**2, axis=1)/wavea0.shape[1]
```

```
In [28]: plt.figure(figsize=(17,5))
         plt.loglog(freqa0, PSDa0, linewidth=1.2, label='Fourier')
         plt.loglog(1/perioda0, spWa0, linewidth=1.2, label='Wavelets')
         plt.xlim((np.min(freqa0), np.max(freqa0)))
         plt.xlabel('Frequency (Hz)')
         plt.ylabel('PSD (V^2 / Hz)')
         plt.legend()
         plt.show()
```



Nous constatons de la même façon que précédemment que la résolution en fréquence de la transformée en ondelettes est bien moins grande que la transformée de Fourier, mais elle nous permet de conserver l'information sur la localisation temporelle des modes du signal. Les deux sont donc très efficaces dans leurs utilisations respectives, et je dirais même plus, complémentaires. Il est d'ailleurs tout aussi simple d'interpréter les résultats des deux transformées (avec un léger avantage pour Fourier tout de même).
Par contre, la transformée en ondelettes a une précision en basse fréquences plus faible que la transformée de Fourier.

5 Conclusion

Au cours de ce TP, nous avons introduit la transformée de Fourier via une DFT manuelle, ainsi que la transformée en ondelettes de Morlet. Nous avons pu exhiber la différence entre les deux, et leur application à des données réelles nous a permis de montrer leurs domaines d'applications, leurs forces et leurs faiblesses. Finalement, nous ne pouvons que dire que ces deux approches sont complémentaires lors de l'étude d'un signal temporel, car elles permettent de mettre en évidence des phénomènes différents.

Je tiens à remercier Olga Alexandrova et Baptiste Cecconi du LESIA pour leur encadrement durant cette journée et leur passion communicative. Merci beaucoup !