Exercices

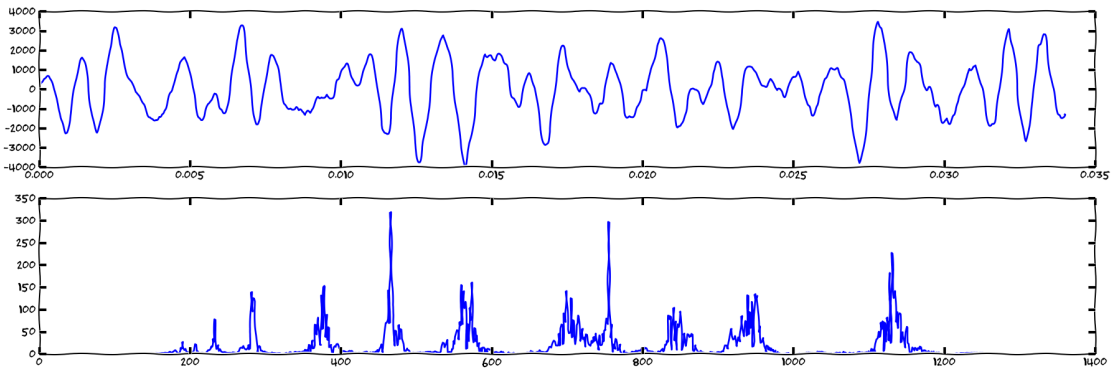
Autour de la transformée de Fourier discrète

Introduction

Des renseignements très utiles sur un signal donné [1] peuvent être obtenus en déterminant son spectre fréquentiel.

La découverte de l'intérêt de la décomposition spectrale d'un signal est due à Joseph Fourier qui a établi que tout signal périodique peut se décomposer en une somme finie de signaux sinusoïdaux de fréquences et d'amplitudes constantes. L'invention de Fourier, en 1811, à 43 ans, tire son origine de la résolution de l'équation de la chaleur à l'aide d'une fonction périodique, somme de fonctions trigonométriques.

C'est l'ensemble fini des amplitudes et des fréquences de ces fonctions trigonométriques que l'on nomme le spectre du signal.



L'outil mathématique pour obtenir la décomposition d'un signal périodique et par conséquent le spectre correspondant est le développement en séries de Fourier (DSF).

Dans le cas d'un signal qui n'est pas périodique, une analyse comparable du signal peut être effectuée pour mettre en évidence les composantes fréquentielles principales d'un signal. On a recours dans ce cas à un outil mathématique appelé transformée de Fourier (TF).

Enfin, l'analyse spectrale d'un signal numérique, donc discret, fait intervenir quant à elle un outil mathématique appelé transformée de Fourier discrète (TFD).

Signal d'entrée	(1) ⟶	Spectre	(2) ⟶	Spectre filtré	(3) ⟶	Signal de sortie
périodique	DSF		filtrage		DSF^{-1}	
non-périodique	TF		filtrage		TF^{-1}	
discret	TFD		filtrage		TFD^{-1}	

1. D'un point de vue mathématique, le signal est une fonction réelle ou complexe de la variable réelle.

DSF : Développement en séries de Fourier
TF : Transformée de Fourier
TFD : Transformée de Fourier discrète

Si le signal d'entrée est une fonction du temps, on parle alors d'un spectre de fréquences temporelles. C'est le cas de tous les signaux temporels de la physique.

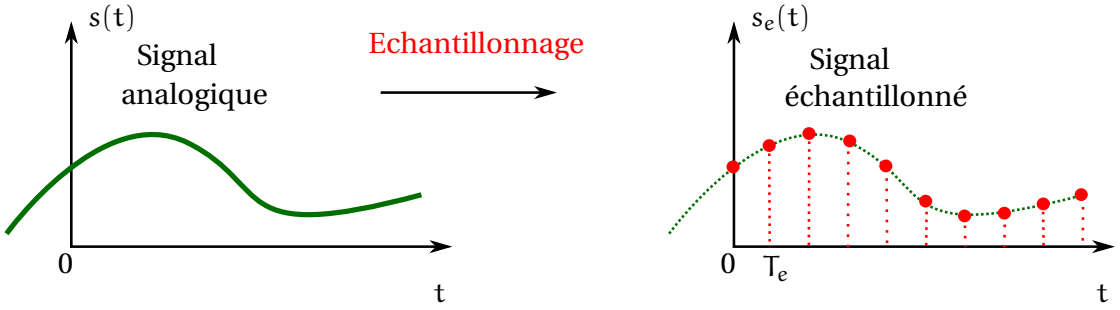
Si le signal d'entrée est une fonction de l'espace, on parle alors d'un spectre de fréquences spatiales. C'est le cas dans les images et leur traitement.

Dans les deux cas, l'opération de filtrage a lieu sur le spectre en fréquence. Le tableau ci-dessus en résume le principe.

En dehors de l'analyse spectrale d'un signal, les outils mathématiques que sont les séries de Fourier et la transformée de Fourier sont également utiles pour d'autres usages comme la résolution d'équations aux dérivées partielles, le calcul de la valeur de certaines intégrales, le calcul de la valeur de la somme de certaines séries, etc.

1 - Présentation de la TFD

Nous allons travailler sur des signaux numériques qui sont donc par nature discrétisés. Obtenir le spectre, ici en fréquence, d'un signal numérisé en fonction du temps, consiste à chercher la transformée de Fourier discrète (TFD) de la série de valeurs prises par le signal d'entrée.



Définition

La transformée de Fourier discrète – TFD – réalise la correspondance entre deux suites de N termes.

$$\{s_k, k \in [0, N-1]\} \longrightarrow \{S_m, m \in [0, N-1]\} \text{ avec}$$

$$S_m = \sum_{k=0}^{N-1} s_k e^{-j\frac{2\pi}{N}km}$$

En posant $w_N = e^{-j\frac{2\pi}{N}}$, racine N-ième de l'unité, la TFD s'écrit

$$S_m = \sum_{k=0}^{N-1} s_k w_N^{km} \tag{1}$$

s_k est réel alors que S_m est complexe. Le module de S_m, $|S_m|$ en fonction de m donne le spectre en amplitude du signal $\{s_k\}$ et $\arg(S_m)$ sa phase.

Mise en forme matricielle de la TFD

La somme précédente a une forme matricielle. Par exemple, pour $N = 4$

$$
\begin{aligned}
S_0 &= s_0(w_4)^{0.0} + s_1(w_4)^{1.0} + s_2(w_4)^{2.0} + s_3(w_4)^{3.0}\\
S_1 &= s_0(w_4)^{0.1} + s_1(w_4)^{1.1} + s_2(w_4)^{2.1} + s_3(w_4)^{3.1}\\
S_2 &= s_0(w_4)^{0.2} + s_1(w_4)^{1.2} + s_2(w_4)^{2.2} + s_3(w_4)^{3.2}\\
S_3 &= s_0(w_4)^{0.3} + s_1(w_4)^{1.3} + s_2(w_4)^{2.3} + s_3(w_4)^{3.3}
\end{aligned}
$$

soit

$$
\begin{pmatrix} S_0 \\ S_1 \\ S_2 \\ S_3 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & w & w^2 & w^3 \\ 1 & w^2 & w^4 & w^6 \\ 1 & w^3 & w^6 & w^9 \end{pmatrix} \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix}
$$

soit

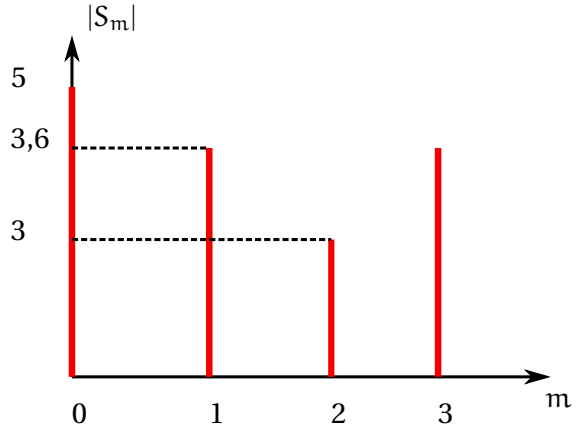
$$
\mathcal{S} = W_N.s \tag{2}
$$

où $\mathcal{S}$ est le vecteur complexe de la transformée de Fourier discrète du vecteur signal s

1 - Montrer que le vecteur signal $s = \{2, 3, -1, 1\}$ à comme TFD le vecteur spectre $S = \{5 + 0j, 3 - 2j, -3 + 0j, 3 + 2j\}$.

$$
\begin{pmatrix} 5 \\ 3-2j \\ -3 \\ 3+2j \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{pmatrix} \begin{pmatrix} 2 \\ 3 \\ -1 \\ 1 \end{pmatrix}
$$

2 - Le spectre consiste à tracer le module des composantes de $\mathcal{S}$ en fonction de m : $|S_m|(m)$
Tracer le spectre du vecteur signal précédent.



3 - Montrer que $\frac{S_0}{N}$ est la valeur moyenne du signal $\{s_k\}$.

S_0 correspond à l'amplitude de la fréquence nulle et est nécessairement réelle. Cette amplitude *serait* la valeur moyenne du signal s_k si on la divisait par N. En effet, on a

$$
S_0 = \sum_{k=0}^{N-1} s_k \qquad \rightarrow \qquad \frac{S_0}{N} = \frac{1}{N} \sum_{k=0}^{N-1} s_k = \langle s_k \rangle
$$

---

4 - Le spectre en fréquence est donc le module des coefficients S_m complexes obtenus par la TFD. Montrer que pour un spectre de N valeurs, $S_{N-m} = \overline{S_m}$ et que par conséquence le spectre tracé sur la totalité des points est dédoublé : le spectre attendu et son spectre miroir.

Pour $1 \leq m \leq N-1$, on montre que $S_{N-m} = \overline{S_m}$ ce qui réduit la partie utile du spectre en fréquence, puisque l'on trace le module de S_m, $|S_m|(m)$.

En effet,

$$\begin{aligned} S_{N-m} &= \sum_{k=0}^{N-1} s_k e^{-j\frac{2\pi}{N}k(N-m)} \\ &= \sum_{k=0}^{N-1} s_k e^{-j\frac{2\pi}{N}kN}.e^{j\frac{2\pi}{N}km} \\ &= \sum_{k=0}^{N-1} s_k e^{j\frac{2\pi}{N}km} \\ &= \overline{S_m} \end{aligned}$$

Finalement, pour tracer le spectre en fréquence, soit le module de S_m en fonction de m, nous n'avons besoin que des valeurs S_m, pour $1 \leq m \leq N/2$.

---

2 - Algorithme naïf d'obtention du vecteur spectre $\mathcal{S}$

On charge les bibliothèques suivantes :

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

La bibliothèque `numpy` permet de travailler simplement avec des matrices à valeurs complexes. Une annexe donne quelques fonctions utiles.

Fonction w_N

5 - Coder la fonction `w(N)` définie pour l'équation (1). Cette fonction prend en argument N, le nombre de points du signal.

On construit dans un premier temps la fonction `w(N)` définie pour l'équation (1).

```
def w(N):
    return np.exp((-2j*np.pi)/(N))
```

TFD à partir de la somme

6 - Écrire une fonction `TFD1(s)`, qui prend en argument un vecteur signal `s` et qui renvoie un vecteur spectre `S`, à partir de la définition donnée par l'équation (1).

```
def TFD1(s):
    N = len(s)
    S = np.zeros(N,dtype=complex)
```

```
    for m in range(N): #de 0 a N-1 pour N valeurs
        for k in range(N):
            S[m]+=s[k]*(w(N))**(k*m)
    return S
```

---

7 - Vérifier que la fonction précédente donne $S = \{5, 3-2j, -3, 3+2j\}$ à partir du signal $s = \{2, 3, -1, 1\}$.

On prendra donc ici pour s, `s = np.array([2,3,-1,1],dtype = complex)`

On pourra arrondir les flottants avec la fonction numpy : `np.round()`

```
>>> s = np.array([2,3,-1,1],dtype = complex)
>>> s
array([ 2.+0.j,  3.+0.j, -1.+0.j,  1.+0.j])
>>> TFD1(s)
array([ 5. +0.00000000e+00j,  3. -2.00000000e+00j, -3. -9.79717439e-16j,
        3. +2.00000000e+00j])
>>> np.round(TFD(s)
array([ 5.+0.j,  3.-2.j, -3.-0.j,  3.+2.j])
>>>
```

TFD à partir de la matrice W_N

8 - Écrire une fonction `M_TFD(N)` qui prend en argument le nombre d'éléments du signal – dimension du vecteur signal – et qui génère la matrice W_N adéquate définie par (2).

```
def M_TFD(N):
    ''' renvoie la matrice de la TDF d ordre N'''
    M_W = np.zeros((N,N),dtype = complex)
    for m in range(N):
        for k in range(N):
            M_W[m,k] = (w(N))**(k*m)
    return M_W
```

---

9 - Écrire une fonction `TFD(s)`, qui prend en argument un vecteur signal et qui renvoie la TFD à partir de son écriture matricielle.

```
def TFD(s):
    ''' entree : s un vecteur colonne signal
    en sortie S un vecteur colonne frequence'''
    N = len(s)
    return np.dot(M_TFD(N),s) #np.dot(a,b) realise le produit matriciel
```

Vérifier à nouveau que la fonction créée donne $S = \{5, 3-2j, -3, 3+2j\}$ à partir du signal $s = \{2, 3, -1, 1\}$.

Complexité

10 - Évaluer la complexité de l'algorithme naïf.

Les boucles imbriquées de la fonction `MTFD(s)` indique que

$$\mathcal{C}(N) = \mathcal{O}(N^2)$$

TFD inverse : obtenir le vecteur signal s

Une fois le signal filtré, et par conséquent le vecteur spectre $\mathcal{S}$ modifié, nous construisons le signal de sortie à partie de la TDF inverse telle que :

$$s_k = \frac{1}{N}\sum_{m=0}^{N-1} S_m w_N^{-mk}$$

soit, sous forme matricielle,

$$s = \frac{1}{N}\overline{W_N}\mathcal{S} \tag{3}$$

où $\overline{W_N}$, la matrice conjuguée de W_N est telle que

$$\overline{W_N}.W_N = N.I_N$$

I_N est la matrice identité d'ordre N et $\overline{W_N} = (\overline{w_N^{km}})_{0\leq m,k\leq N-1}$.

11 - Écrire une fonction `M_TFDI(N)` qui prend en argument le nombre `N` d'éléments du vecteur spectre et qui génère la matrice $\overline{W_N}$, matrice conjuguée de W_N.

```
def M_TFDI(N):
    ''' renvoie la matrice conjugate de W_N d ordre N'''
    return M_TFD(N).conjugate()
```

---

12 - Écrire une fonction `TFDI(S)` qui prend en argument un vecteur spectre $\mathcal{S}$ et que renvoie le vecteur signal *s*.

Tester cette fonction à partir de l'exemple précédent.

```
def TFDI(S):
    ''' entree : S un vecteur colonne frequence
    en sortie un vecteur colonne signal s'''
    N = len(S)
    return 1/N*np.dot(M_TFDI(N),S) #Ne pas oublier 1/N
```

Filtrage

Le principe du filtrage est relativement simple.

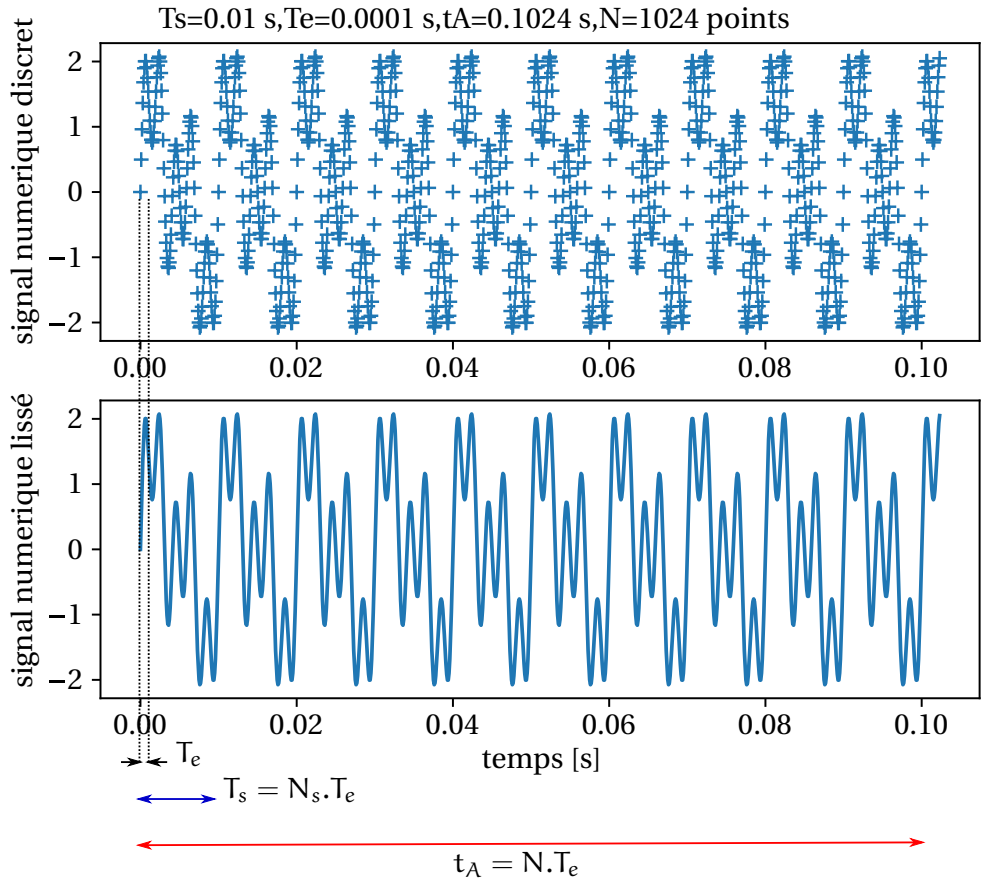
On peut modifier [2] les composantes (complexes) du vecteur spectre $\mathcal{S}$ associées aux fréquences du signal.

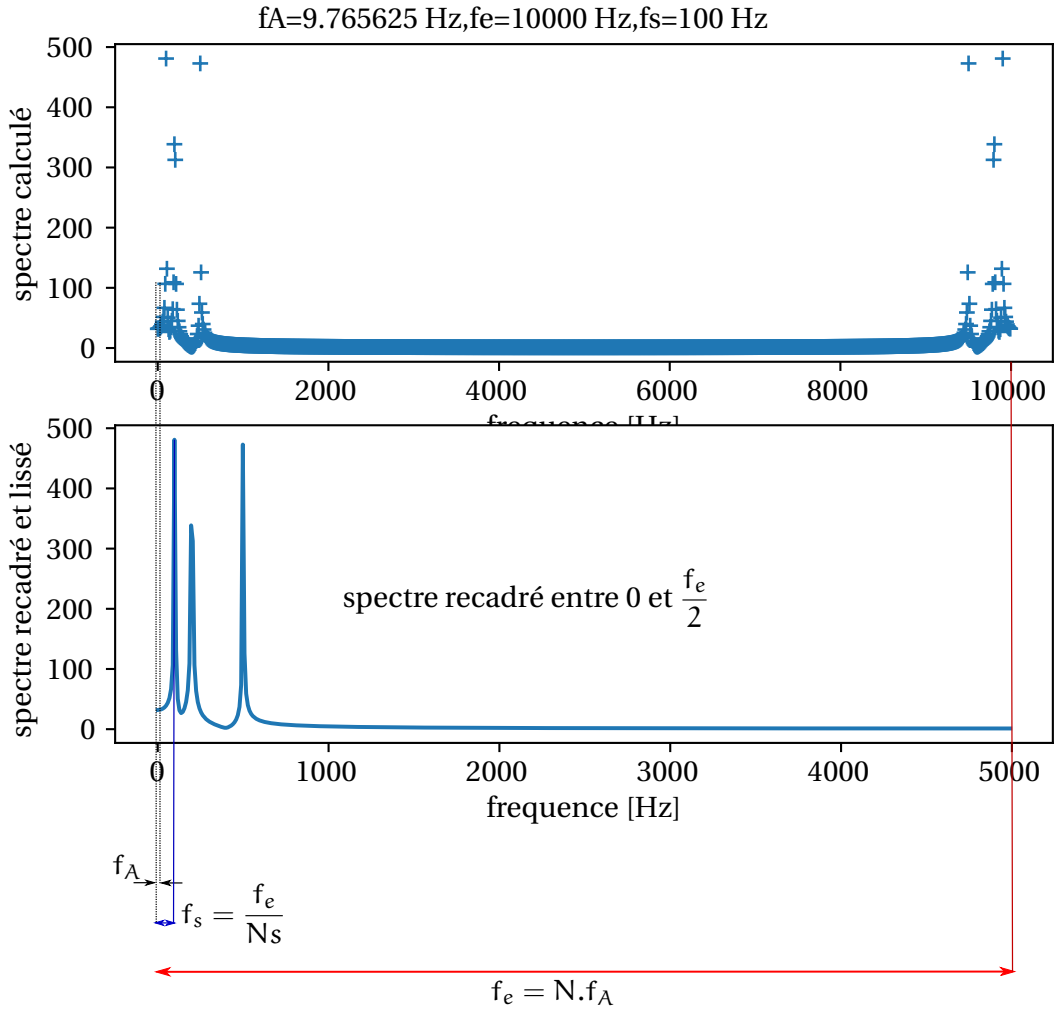
Pour que ce filtrage soit parlant pour un physicien, nous allons travailler sur un signal physique dont le fondamental est de fréquence f_s échantillonné à la fréquence f_e sur une durée totale d'acquisition t_A.

2. ... atténuer, amplifier, déphaser etc.

Présentation d'un signal physique

SIGNAL FONCTION DU TEMPS	SPECTRE FONCTION DE LA FRÉQUENCE
Avec Fourier, tout est inversé	
Durée totale d'acquisition du signal t_A	Fréquence d'échantillonnage du spectre $f_A = \frac{1}{t_A}$
Échelle temporelle la plus grande $\leftrightarrow$ Échelle fréquentielle la plus petite	
Période du signal $T_s = \frac{1}{f_s}$	Fréquence du signal f_s
Période d'échantillonnage du signal $T_e = \frac{T_s}{N_s}$	Fréquence d'échantillonnage du signal $f_e = N_s.f_s$
Échelle temporelle la plus petite $\leftrightarrow$ Échelle fréquentielle la plus grande	
N_s est le nombre de points échantillonnés par période $N_s > 2$ est le critère de Shannon	
$t_A = N.T_e$	$f_e = N.f_A$
N est le nombre total de points échantillonnés du signal $N = 2^n$ doit être une puissance de 2 : $1024 = 2^{10}$	





Construction et tracé du signal `signal`

13 - On souhaite construire un signal `s` de $N = 2^{10} = 1024$ échantillons, somme de trois sinusoïdes de même amplitude, mais de fréquences $f_s = 100$ Hz, $2.f_s$ et $5.f_s$, et qui présente 100 échantillons par période $T_s = \frac{1}{f_s}$ du fondamental du signal.

Définir la fonction `fn(x)` qui prend en entrée `x` la fréquence du fondamental et renvoie le signal demandé.

```
def fn(x):
    return  np.sin(x) + np.sin(2*x) + np.sin(5*x)
```

14 - Compléter les lignes du code suivant :

```
N = 2**10 # le nombre d echantillons est une puissance de 2
fs = A COMPLETER #(en Hz) frequence du signal
Ts = 1/fs # (en s) du signal
Ns = A COMPLETER # nombre de points par periode du signal
fe = A COMPLETER fe est fonction de Ns et fs # frequence d echantillonage. Ns>2
                                             Shannon largement verifiee
Te= A COMPLETER # periode d echantillonage
```

```
tA = A COMPLETER # tA duree d acquisition/totale du signal
```

```
N = 2**10 # nombre de points lors de l acquisition puissance de 2
fs = 100 #(en Hz) du signal
Ts = 1/fs # (en s) du signal
Ns = 100  # nombre de points pour echantillonage
fe = Ns*fs # frequence de l echantillonage Ns>2 Shannon largement verifiee
Te=1/fe # periode d echantillonage
#duree d acquisition tA
tA = N*Te #
```

---

15 - A l'aide de la fonction `np.arange(debut,fin,pas)` présentée en annexe, générer une liste `numpy` des temps `t` sur le durée totale du signal avec le pas adéquate.

```
#on peut generer la liste des temps
t = np.arange(0,tA,Te) #N points car tA = N*Te
```

---

16 - Générer le signal `signal` à l'aide la fonction `fn(x)` puis tracer `signal`.

Pour le tracé, compléter le code suivant :

```
plt.plot(A COMPLETER)
plt.xlabel('temps [s]')
plt.ylabel('signal numerique lisse')
plt.show()
```

```
signal = fn(2*np.pi*fs*t)
plt.subplot(4,1,1)
plt.plot(t,signal)
plt.xlabel('temps [s]')
plt.ylabel('signal numerique lisse')
plt.show()
```

Calcul et tracé du spectre `spectre` de `signal`

17 - Calculer le spectre en fréquence du signal `signal`.

```
#Calcul du spectre
spectre = TFD(signal)
```

---

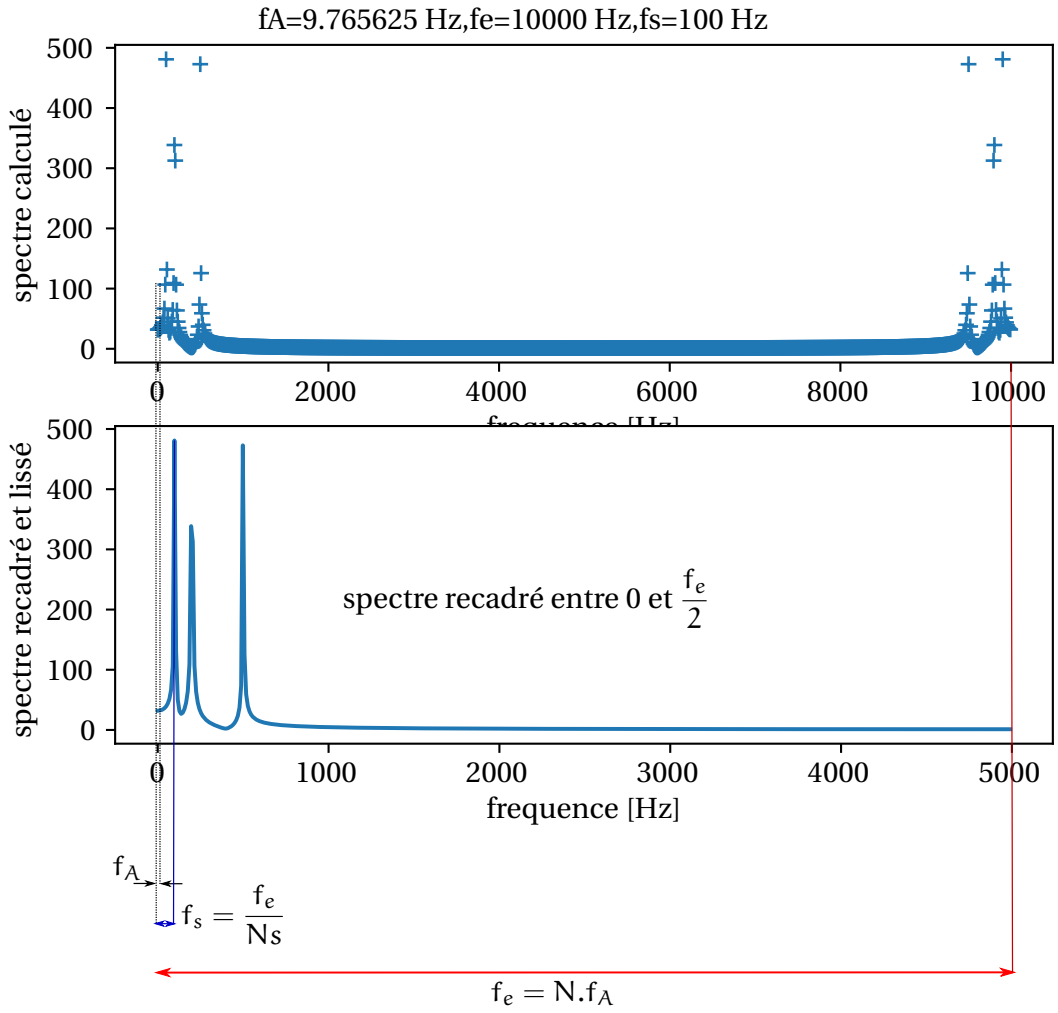
18 - Pour tracer le module des composantes complexes du spectre en fonction de `f`, il faut générer la liste `numpy` de fréquences. A partir de la fonction `numpy`, `np.arange(debut,fin,pas)`, générer la liste des fréquences `f` associées à ce signal.

Tracer le module des composantes complexes du spectre en fonction de `f`. On indique que le module d'un complexe `A=0.1+2.j` et donné par `abs(A)`.

```
#Pour le spectre en frequence
fA=1/tA # On genere les valeurs de frequences
f = np.arange(0,fe,fA)
```

```
#Trace du spectre sur N//2
plt.subplot(4,1,2)
plt.plot(f[:N//2],abs(spectre[:N//2]))
plt.xlabel('frequence [Hz]')
plt.ylabel('spectre calcule')
plt.show()
```

Comme nous l'avons montré, le tracé du spectre se réduit à un tracé sur la moitié des valeurs calculées, soit ici de 0 à $\frac{f_e}{2}$.



Filtrage

19 - Réaliser un filtre passe-bas numérique consiste, dans une conception extrêmement simple, à mettre à 0 l'ensemble des valeurs du vecteur spectre supérieures à $1.5f_s$ si on a choisi une fréquence de coupure telle que $f_c = 1.5f_s$.

Écrire une fonction `filtrePBas(spectre)` qui prend en argument un vecteur spectre complexe et qui renvoie le spectre filtré à $1.5f_s$. L'appliquer au spectre précédent et tracer le module des composantes de ce spectre filtré.

```
#Filtrage passe-bas : on met a zero toutes les f>1.5fc
d = N//Ns #permet d'avoir l'indice de fs
def filtrePBas(spectre):
    ''' frequence de coupure a 1.5 fs'''
    spectre[int(1.5*d):]=0
    return spectre
```

```
spectre_filtre = filtrePBas(spectre)
```

```
#Trace du spectre filtre sur N//2
plt.subplot(4,1,3)
plt.plot(f[:N//2],abs(spectre_filtre[:N//2]))
plt.xlabel('frequence [Hz]')
plt.ylabel('spectre calcule')
plt.show()
```

20 - Une fois le vecteur spectre modifié, on lui applique la transformée de Fourier discrète inverse définit précédemment `TDFI(spectre)`. Pour terminer, tracer la partie réelle du signal filtré.

```
signal_filtre = TFDI(spectre_filtre)
```

```
plt.subplot(4,1,4)
plt.plot(t,signal_filtre).real)
plt.show()
```

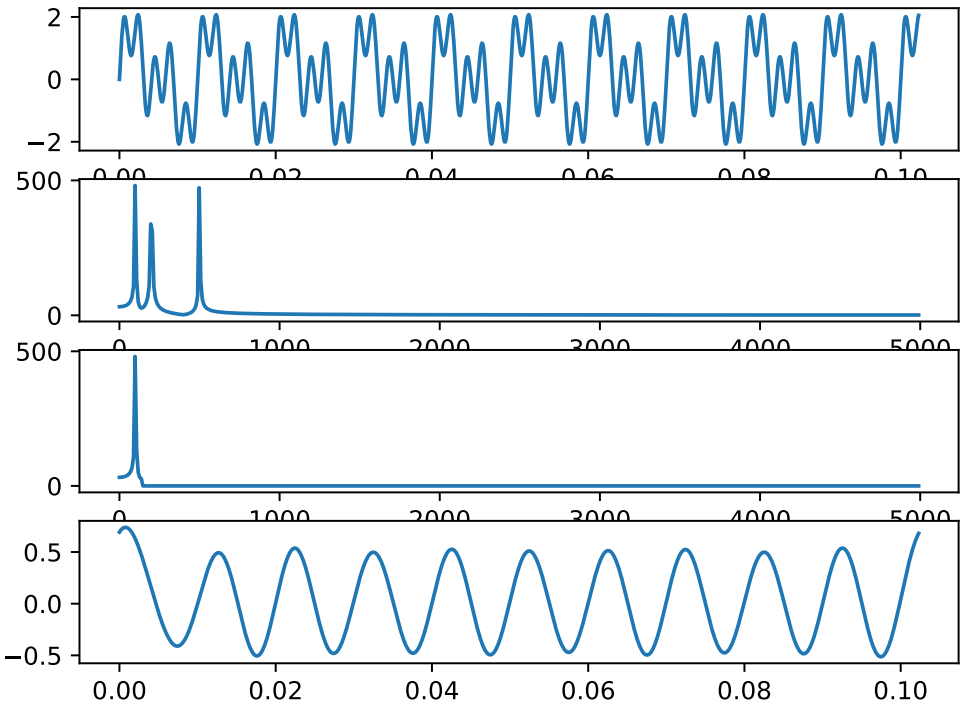


FIGURE 1 – Filtrage passe-bas

21 - Réaliser un filtre passe-bande numérique consiste, dans une conception extrêmement simple à mettre à 0 l'ensemble des valeurs du vecteur spectre inférieures et supérieures à deux valeurs choisies.

```
#Filtrage passe bande : on met a zero f<0.5fs et f>1.5fs
def filtrePBande(spectre):
    p = 0.70
    spectre[int((2-p)*2*d):]=0 #2d renvoie a la frequence 2.fs
    spectre[:int(p*2*d)]=0
    return spectre
```

```
spectre_filtre = filtrePBande(spectre)
```

Une fois le vecteur spectre modifié, on lui applique la transformée de Fourier discrète inverse et on extrait la partie réel du signal calculé.

```
signal_filtre = (TFDI(spectre_filtre)).real
```

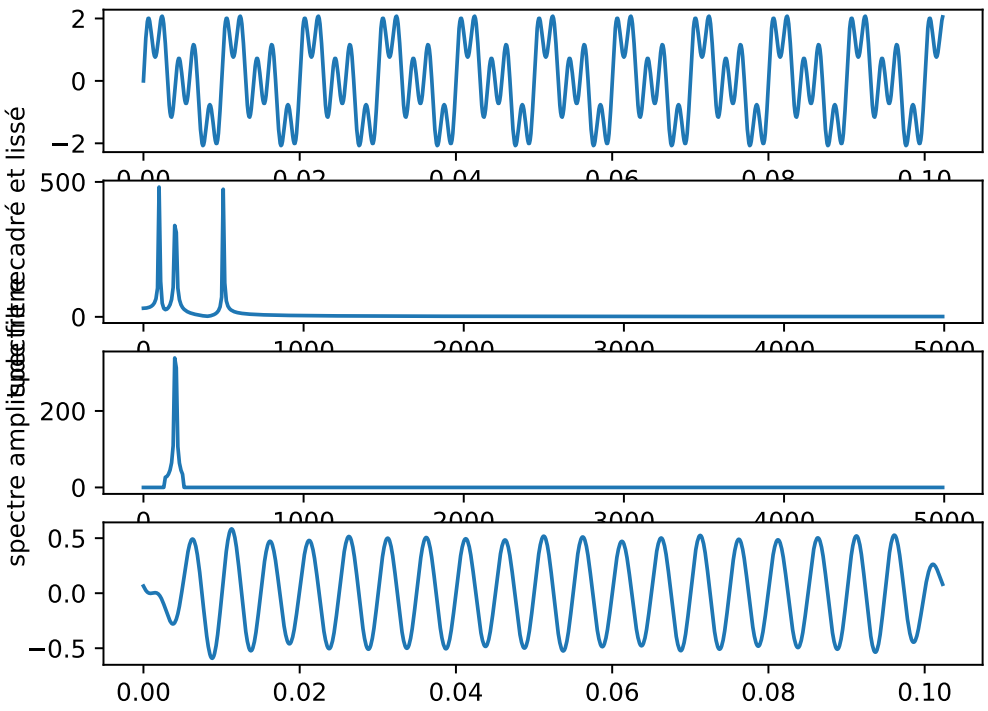


FIGURE 2 – Filtrage passe-bande autour de $2.f_s$

3 - Algorithme de TFD rapide : FFT de James Cooley et John Tukey (1965)

Complexité de l'algorithme naïf de TFD : un inconvénient rédhibitoire

22 - Quelle est la complexité d'une TFD?

On multiplie une matrice $(N \times N)$ par une matrice colonne de taille N. On a donc une complexité en $\mathcal{O}(N^2)$.

Pour un vecteur de taille 1000, on a donc 10^6 opérations. Puis pour un vecteur de taille 3000, le temps de calcul est par $3^2 = 9$

Pour des petites valeurs de N, cela ne pose pas de problème. En revanche, le temps de calcul devient prohibitif pour des applications en temps réel de la TFD, par exemple, pour les sons audibles par une oreille humaine qui ont des fréquences s'étendant jusqu'à environ 20000 Hz.

Pour analyser correctement un son, il faudra donc l'échantillonner à 40000 Hz, nous avons environ 2 milliards d'opérations. Si c'est faisable en quelques secondes, cela interdit le traitement du son nécessaire en temps réel, comme la simple écoute d'un son numérisé.

Finalement, l'algorithme naïf est trop lent.

Dans la suite nous étudions l'algorithme de TFD rapide de Cooley et Tukey beaucoup plus rapide.

Histoire de la naissance de la TFD rapide, FFT : Fast Fourier Transform

Cet algorithme[3], y compris sa version récursive, a été inventé vers 1805 par Carl Friedrich Gauss, qui l'a utilisé pour interpoler les trajectoires des astéroïdes Pallas et Junon. Son travail est resté inaperçu du fait d'avoir été publié uniquement à titre posthume et en néo-latin. Il semble que Gauss n'ai pas analysé le temps de calcul asymptotique.

Diverses formes limitées ont également été redécouvertes à plusieurs reprises au cours du XIX^e et au début du XX^e siècle.

Les FFT sont devenues populaires après que James Cooley d'IBM et John Tukey de l'université de Princeton aient publiés un article en 1965 réinventant l'algorithme et décrivant comment l'exécuter facilement sur un ordinateur.

Tukey aurait eu l'idée lors d'une réunion du comité consultatif scientifique du président Kennedy sur les moyens de détecter les essais d'armes nucléaires en Union soviétique en utilisant les données des sismomètres situés à l'extérieur du pays.

La difficulté venait du fait que l'analyse de cette grande quantité de données (nombre de capteurs et durée d'acquisition) nécessitait des algorithmes rapides pour calculer la TFD.

Cette tâche était essentielle pour la ratification du projet d'interdiction des essais nucléaires afin que toute violation puisse être détectée sans qu'il soit nécessaire de visiter les installations soviétiques.

Un autre participant à cette réunion, Richard Garwin d'IBM, qui avait saisi le puissance de cette méthode, a mis Tukey en contact avec Cooley en s'assurant toutefois que Cooley ne connaissait pas le but initial. Il a alors été dit à Cooley que ce travail était nécessaire pour déterminer les périodicités des orientations de spin dans un cristal 3-D d' hélium-3.

La publication de l'article de Cooley et Tukey, suivi du développement simultané de convertisseurs analogique-numérique capables d'échantillonner à des fréquences allant jusqu'à 300 kHz a rendu l'algorithme de la FFT très opératoire.

3. L'histoire de la naissance de cet algorithme est tirée de l'encyclopédie en ligne Wikipedia.

Principe de l'algorithme récursif de la FFT : « diviser pour régner »

L'algorithme de la FFT divise, à chaque étage récursif, une TFD de taille N en deux TFD entrelacées de taille N/2.

Cette idée provient d'une écriture récursive de la TFD pour laquelle on sépare la somme initiale en deux sommes : une somme sur les indices pairs $k = 2p$, et une somme sur les indices impairs $k = 2p+1$. Cela impose à N d'être une puissance de 2 : $N = 2^q$.

23 - Monter que

$$S_m = \underbrace{\sum_{p=0}^{N/2-1} s_{2p} e^{-j\frac{2\pi}{N/2}pm}}_{\text{TDF des index pairs } s_{2p}} + e^{-j\frac{2\pi}{N}m} \underbrace{\sum_{p=0}^{N/2-1} s_{2p+1} e^{-j\frac{2\pi}{N/2}pm}}_{\text{TDF des index impairs } s_{2p+1}}$$

soit

$$S_m = E_m + e^{-j\frac{2\pi}{N}m} O_m$$

si on note la TFD des signaux d'index pair E_m (Even-index pour index pairs) et la TFD des entrées d'index impair O_m (Odd-index pour index impairs) telles que

$$E_m = \underbrace{\sum_{p=0}^{P-1} e_p e^{-j\frac{2\pi}{P}pm}}_{\text{TDF des index pairs } s_{2p}=e_p}$$

et

$$O_m = \underbrace{\sum_{p=0}^{P-1} o_p e^{-j\frac{2\pi}{P}pm}}_{\text{TDF des index impairs } s_{2p+1}=o_p}$$

Rappelons la définition de la transformée de Fourier discrète (TFD),

$$S_m = \sum_{k=0}^{N-1} s_k e^{-j\frac{2\pi}{N}km} = \sum_{k=0}^{N-1} s_k w_N^{km}$$

La première séparation peut donc s'écrire à partir des indices paires $(2p)$ et impaires $(2p+1)$:

$$S_m = \sum_{p=0}^{N/2-1} s_{2p} e^{-j\frac{2\pi}{N}2pm} + \sum_{p=0}^{N/2-1} s_{2p+1} e^{-j\frac{2\pi}{N}(2p+1)m}$$

On peut factoriser un multiplicateur commun de la seconde somme.

$$\sum_{p=0}^{N/2-1} s_{2p+1} e^{-j\frac{2\pi}{N}(2p+1)m} = e^{-j\frac{2\pi}{N}m} \sum_{p=0}^{N/2-1} s_{2p+1} e^{-j\frac{2\pi}{N}2pm}$$

d'où $$S_m = \underbrace{\sum_{p=0}^{N/2-1} s_{2p} e^{-j\frac{2\pi}{N/2}pm}}_{\text{TDF des index pairs } s_{2p}} + e^{-j\frac{2\pi}{N}m} \underbrace{\sum_{p=0}^{N/2-1} s_{2p+1} e^{-j\frac{2\pi}{N/2}pm}}_{\text{TDF des index impairs } s_{2p+1}}$$

Si on pose $e_p = s_{2p}$, $o_p = s_{2p+1}$ et $P = \dfrac{N}{2}$, il est apparait que les deux sommes sont la TFD de la partie à index pair et la TFD de la partie à index impair de la somme initiale.

$$S_m = \underbrace{\sum_{p=0}^{P-1} e_p e^{-j\frac{2\pi}{P}pm}}_{\text{TDF des index pairs } s_{2p}=e_p} + e^{-j\frac{2\pi}{N}m} \underbrace{\sum_{p=0}^{P-1} o_p e^{-j\frac{2\pi}{P}pm}}_{\text{TDF des index impairs } s_{2p+1}=o_p}$$

signal $\{s_k\}$	$\{s_0,$	$s_1,$	$s_2,$	$s_3,$	$s_4,$	$s_5,$	$s_6,$	$s_7\}$
indice k	0	1	2	3	4	5	6	7
division en 2 signaux	$\{s_0,$	$s_2,$	$s_4,$	$s_6\}$	$\{s_1,$	$s_3,$	$s_5,$	$s_7\}$
nouvel indice p	0	1	2	3	0	1	2	3
nouveaux signaux $\{e_p\}$ et $\{o_p\}$	$\{e_0,$	$e_1,$	$e_2,$	$e_3\},$	$\{o_1,$	$o_2,$	$o_3,$	$o_4\}$

On note la TFD des signaux d'index pair E_m (Even-index pour index pairs) et la TFD des entrées d'index impair O_m (Odd-index pour index impairs) et on obtient :

$$S_m = E_m + e^{-j\frac{2\pi}{N}m} O_m$$

---

24 - La particularité de la définition de la TFD, nous permet de ne faire que la moitié des calculs, soit N/2 au lieu de N. En effet, montrer que

$$S_{m+\frac{N}{2}} = E_m - e^{-j\frac{2\pi}{N}m} O_m$$

La particularité de la définition de la TFD, nous permet de ne faire que la moitié des calculs, soit N/2 au lieu de N. Explication.

Calculons

$$S_{m+\frac{N}{2}} = \sum_{p=0}^{N/2-1} s_{2p} e^{-j\frac{2\pi}{N}2p(m+\frac{N}{2})} + \sum_{p=0}^{N/2-1} s_{2p+1} e^{-j\frac{2\pi}{N}(2p+1)(m+\frac{N}{2})}$$

Soit, en procédant comme précédemment :

$$\begin{aligned}
S_{m+\frac{N}{2}} &= \sum_{p=0}^{N/2-1} s_{2p} e^{-j\frac{2\pi}{N}2p(m+\frac{N}{2})} + e^{-j\frac{2\pi}{N}(m+\frac{N}{2})} \sum_{p=0}^{N/2-1} s_{2p+1} e^{-j\frac{2\pi}{N}2p(m+\frac{N}{2})} \\
&= \sum_{p=0}^{N/2-1} s_{2p} e^{-j\frac{2\pi}{N}2pm} e^{-j2\pi p} + e^{-j\frac{2\pi}{N}m} e^{-j\pi} \sum_{p=0}^{N/2-1} s_{2p+1} e^{-j\frac{2\pi}{N/2}pm} e^{-j2\pi p} \\
&= \sum_{p=0}^{N/2-1} s_{2p} e^{-j\frac{2\pi}{N}2pm} e^{-j2\pi p} - e^{-j\frac{2\pi}{N}m} \sum_{p=0}^{N/2-1} s_{2p+1} e^{-j\frac{2\pi}{N/2}pm} \\
&= E_m - e^{-j\frac{2\pi}{N}m} O_m
\end{aligned}$$

Finalement, le seul calcul des valeurs E_m et O_m, sur la moitié des valeurs, soit $\frac{N}{2}$, permet d'obtenir la totalité du spectre S_m.

$$\begin{cases}
S_m = E_m + e^{-j\frac{2\pi}{N}m} O_m & \text{pour } 0 \leq m \leq \frac{N}{2} - 1 \\
S_{m+\frac{N}{2}} = E_m - e^{-j\frac{2\pi}{N}m} O_m & \text{pour } \frac{N}{2} \leq m \leq N - 1
\end{cases}$$

Ce résultat exprime récursivement la TFD de longueur N en deux TFD de taille $\frac{N}{2}$ ce qui est au cœur de la transformée de Fourier discrète rapide dite FFT.

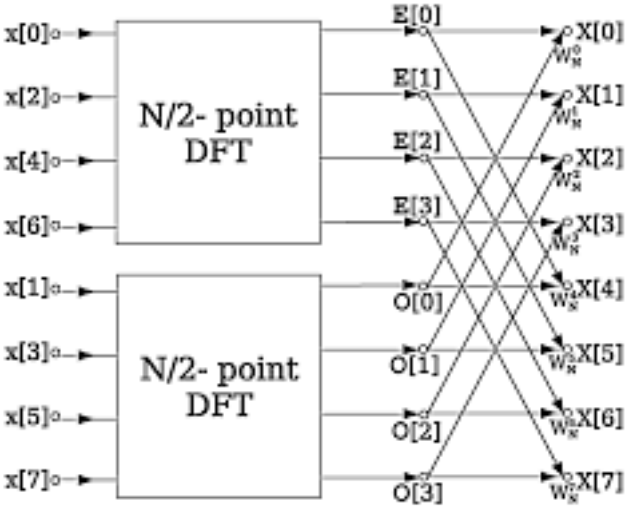


FIGURE 3 – Schéma de principe d'un étage récursif

Complexité

25 - Quelle est la complexité de ce nouvel algorithme ?

Partons des deux formulations vues précédemment et rappelons-nous que pour l'algorithme naïf, pour un vecteur N, on a $\mathcal{O}(N^2)$.

$$S_m = E_m + e^{-j\frac{2\pi}{N}m} O_m$$

Pour le premier étage récursif

1. Complexité de E_m qui met en jeu $N/2$ éléments est en $\mathcal{O}_1\left(\left(\frac{N}{2}\right)^2\right)$
2. De même pour O_m
3. Ce à quoi il faut ajouter N multiplications pour $e^{-j\frac{2\pi}{N}m} \times O_m$

La complexité de ce premier étage récursif est donc :

$$\mathcal{C}_1(N) = 2.\mathcal{O}_1\left(\left(\frac{N}{2}\right)^2\right) + \mathcal{O}_1(N)$$

Soit

$$\mathcal{C}_1(N) = 2\left(\frac{N}{2}\right)^2 + N = \frac{N^2}{4} + N$$

Pour le second étage récursif Pour ce second étage, les complexités de E_m et O_m sont réduites puisqu'il n'y a que $P = \frac{N}{2}$ valeurs.

$$\left(\frac{N}{2}\right)^2 \leftarrow \left[2\left(\frac{N}{4}\right)^2 + \frac{N}{2}\right]$$

La complexité de s'écrit alors

$$\begin{aligned}\mathcal{C}_2(N) &= 2\left[2\left(\frac{N}{4}\right)^2 + \frac{N}{2}\right] + N \\ &= 4\left(\frac{N}{4}\right)^2 + N + N \\ &= \frac{N^2}{2^2} + 2N\end{aligned}$$

Pour le troisième étage récursif Pour ce troisième étage, les complexités de E_m et O_m sont réduites de la même façon :

$$\left(\frac{N}{4}\right)^2 \leftarrow \left\{2\left(\frac{N}{8}\right)^2 + \frac{N}{4}\right\}$$

$$\begin{aligned}\mathcal{C}_3(N) &= 2\left[2\left\{2\left(\frac{N}{8}\right)^2 + \frac{N}{4}\right\} + \frac{N}{2}\right] + N \\ &= 4\left[2\left(\frac{N}{8}\right)^2 + \frac{N}{4}\right] + N + N \\ &= 8\left(\frac{N}{8}\right)^2 + N + N + N \\ &= \frac{N^2}{8} + 3N \\ &= \frac{N^2}{2^3} + 3N\end{aligned}$$

Pour le n-ième étage récursif

$$\mathcal{C}_n(N) = \frac{N^2}{2^n} + nN$$

Comme $N = 2^n$, et que $\log_2(N) = n.\log_2(2)$, soit $n = \log_2(N)$,
on obtient finalement

$$\mathcal{C}_n(N) = N + N.\log_2(N) \simeq \mathcal{O}(N.\log_2(N))$$

La FFT ramène le calcul de la transformée de Fourier discrète de $\mathcal{O}(N^2)$ à $\mathcal{O}(N\log(N))$. Cette réduction de complexité a ouvert la résolution de nombreux problèmes.

Pour poursuivre l'exemple du son, une centaine de milliers d'opérations seront mobilisées pour la FFT, au lieu de 2 milliards donc pour le TFD naïve : 10000 fois plus rapide.

Pseudo-code de l'algorithme récursif de la FFT

```
1. Fonction FFT(s)
2.     N = longueur de s # s est une puissance de 2
3.     si N = 1
4.       alors retourner s
5.     sinon
6.       e = liste des s a indice pair
7.       o = liste des s a indice impair
8.       E = la FFT de la liste e
9.       O = la FFT de la liste o
10.      pour m de 0 a N/2-1 faire
11.        S[m]   = E[m] + exp(-j*2*\pi /N*m)*O[m]
12.        S[m+N/2] = E[m] - exp(-j*2*\pi /N*m)*O[m]
13.    retourner S
```

1. La ligne `2.` calcule la longueur de la liste (type `array`) de `N` qui doit être une puissance d'un entier n : $N = 2^n$.
2. Les lignes `3.` et `4.` renvoient `s` car pour un seul élément, la transformée de Fourier est l'élément lui-même.
3. Les lignes `6.` et `7.` font la division de la liste initiales à partir de la parités des indices des grandeurs du signal dans la liste `s`.
4. Les lignes `8.` et `9.` sont proprement récursives.
5. Enfin, les lignes `11.` et `12.` construisent le vecteur spectre `S` complexe, à partir de la reformulation donnée plus haut.
6. La dernière ligne renvoie le spectre complexe.

Un algorithme qui divise les objets à traiter, qui traite les sous-objets puis les recombine, est appelé un algorithme « diviser pour régner ».

Code en Python

Tous les listes `numpy` que nous considérerons auront une taille en puissance de 2 : $N = 2^n$. Cette condition est assez restrictive même s'il y a des moyens de la lever.

Comme nous travaillerons avec des listes `numpy`. Il sera inutile de faire des vecteurs colonnes.

Diviser Nous devons scinder en deux moitiés les s_k du signal en deux listes constituées respectivement des éléments du signal d'indices pairs $s_{2p} = e_p$ et des éléments d'indices impairs $s_{2p+1} = o_p$.

26 - Définir une fonction `diviser(s)` qui prend en argument un signal `s` (liste `numpy`) et renvoie `e` et `s` deux listes constituées respectivement des éléments du signal d'indices pairs $s_{2p} = e_p$ et des éléments d'indices impairs $s_{2p+1} = o_p$

Par exemple.

```
def diviser(s):
  '''Prend un signal (liste np.array) en entree
    Renvoie deux listes np.array : e indice pair, o indice impair'''
    N = len(s)
    e = []
    o = []
    for i in range(N):
        if i%2 == 0:
            e.append(s[i])
        else:
            o.append(s[i])
    return np.array(e) , np.array(o)
```

ou bien

```
def diviser(s):
  '''Prend un signal (liste np.array) en entree
    Renvoie deux listes np.array : e indice pair, o indice impair'''
    e = s[::2]
    o = s[1::2]
    return e,o
```

ou encore

```
def diviser(s):
    '''Prend un signal (liste np.array) en entree
    Renvoie deux listes np.array : e indice pair, o indice impair'''
    e = np.array([s[i] for i in range(len(s)) if i % 2 == 0])
    o = np.array([s[i] for i in range(len(s)) if i % 2 == 1])
    return e,o
```

Algorithme de la FFT

27 - Définir la fonction `FFT(s)` qui prend en entrée un signal `s` et renvoie son spectre `S`.

Effectuer récursivement les FFT des deux moitiés en divisant en deux moitiés les deux moitiés précédentes, jusqu'à n'obtenir que des "moitiés" à un seul élément. On est alors ramené au cas où $N = 1$. Il s'agit alors, après avoir mis en mémoire, après avoir empilés les différents calculs non exécutés en mémoire, de les calculer effectivement, à l'envers, à partir du cas où $N = 1$.

```
def FFT(s):
    N = len(s) #N est une puissance de 2
    S = np.zeros((N),dtype=complex)
    if N == 1:
        return np.array([s[0]+0.j])
    else:
        e,o = diviser(s)
        E = FFT(e)
        O = FFT(o)
        for m in range(N//2):
            S[m] = E[m] + np.exp((-2j*np.pi*m)/(N))*O[m]
            S[m+N//2] = E[m] - np.exp((-2j*np.pi*m)/(N))*O[m]
    return S
```

Test

28 - Vérifier que la FFT donne $S = \{5, 3-2j, -3, 3+2j\}$ à partir du signal $s = \{2, 3, -1, 1\}$.

FFT inverse

Le travail pour la FFT peut être refait rapidement pour concevoir une FFT inverse.
A partir de

$$s_k = \frac{1}{N}\sum_{m=0}^{N-1} S_m e^{j\frac{2\pi}{N}mk}$$

Si on note la TFDI des spectres d'index pair e_k (Even-index pour index pairs) et la TFDI des entrées d'index impair o_k (Odd-index pour index impairs) et on obtient :

$$s_k = \frac{1}{2}(e_k + e^{j\frac{2\pi}{N}k} o_k)$$

De même, on peut s'épargner le calculs de la totalité des composantes du signal en calculant

$$s_{k+\frac{N}{2}} = \frac{1}{2}(e_k - e^{j\frac{2\pi}{N}k} o_k)$$

Finalement

$$\begin{cases} s_k = \frac{1}{2}(e_k + e^{j\frac{2\pi}{N}k} o_k) & \text{pour } 0 \leq k \leq \frac{N}{2} - 1 \\ s_{k+\frac{N}{2}} = \frac{1}{2}(e_k - e^{j\frac{2\pi}{N}k} o_k) & \text{pour } \frac{N}{2} \leq k \leq N - 1 \end{cases}$$

Rappelons la définition de la transformée de Fourier discrète inverse (TFDI),

$$s_k = \frac{1}{N} \sum_{m=0}^{N-1} S_m e^{j\frac{2\pi}{N}mk}$$

La première séparation peut donc s'écrire à partir des indices paires $(2p)$ et impaires $(2p+1)$:

$$s_k = \frac{1}{N} \sum_{p=0}^{N/2-1} s_{2p} e^{j\frac{2\pi}{N}2pk} + \frac{1}{N} \sum_{p=0}^{N/2-1} s_{2p+1} e^{j\frac{2\pi}{N}(2p+1)k}$$

On peut factoriser un multiplicateur commun de la seconde somme.

$$\frac{1}{N} \sum_{p=0}^{N/2-1} s_{2p+1} e^{j\frac{2\pi}{N}(2p+1)k} = e^{j\frac{2\pi}{N}k} \frac{1}{N} \sum_{p=0}^{N/2-1} s_{2p+1} e^{j\frac{2\pi}{N}2pk}$$

$$\text{d'où} \qquad s_k = \frac{1}{2} \cdot \underbrace{\frac{2}{N} \sum_{p=0}^{N/2-1} s_{2p} e^{j\frac{2\pi}{N}2pk}}_{\text{TDFI des index pairs}} + e^{j\frac{2\pi}{N}k} \frac{1}{2} \cdot \underbrace{\frac{2}{N} \sum_{p=0}^{N/2-1} s_{2p+1} e^{j\frac{2\pi}{N}2pk}}_{\text{TDFI des index impairs}}$$

Si on pose $E_p = S_{2p}$, $O_p = S_{2p+1}$ et $P = \frac{N}{2}$, il est apparait que les deux sommes sont la TFDI de la partie à index pair et la TFDI de la partie à index impair de la somme initiale.

$$s_k = \frac{1}{2} \left(\underbrace{\frac{1}{P} \sum_{p=0}^{P-1} E_p e^{j\frac{2\pi}{P}pk}}_{\text{TDFI des index pairs } E_p = S_{2p}} + e^{j\frac{2\pi}{N}k} \underbrace{\frac{1}{P} \sum_{p=0}^{P-1} O_p e^{j\frac{2\pi}{P}pk}}_{\text{TDFI des index impairs } O_p = S_{2p+1}} \right)$$

On note la TFDI des spectres d'index pair e_k (Even-index pour index pairs) et la TFDI des entrées d'index impair o_k (Odd-index pour index impairs) et on obtient :

$$s_k = \frac{1}{2}(e_k + e^{j\frac{2\pi}{N}k} o_k)$$

De même, on peut s'épargner le calculs de la totalité des composantes du signal en calculant

$$s_{k+\frac{N}{2}} = \frac{1}{2}(e_k - e^{j\frac{2\pi}{N}k} o_k)$$

Finalement

$$\begin{cases} s_k = \frac{1}{2}(e_k + e^{j\frac{2\pi}{N}k} o_k) & \text{pour } 0 \leq k \leq \frac{N}{2} - 1 \\ s_{k+\frac{N}{2}} = \frac{1}{2}(e_k - e^{j\frac{2\pi}{N}k} o_k) & \text{pour } \frac{N}{2} \leq k \leq N-1 \end{cases}$$

---

29 - Définir la fonction `FFTI(S)` qui prend en entrée un spectre `S` et renvoie son signal complexe `s`.

```
def FFTI(S):
    N = len(S) #N est une puissance de 2
    s = np.zeros((N),dtype=complex)
    if N == 1:
        return np.array([S[0]+0.j])
    else:
        E,O = diviser(S)
        e = FFTI(E)
        o = FFTI(O)
        for k in range(N//2):
            s[k] = 1/2*(e[k] + np.exp((2j*np.pi*k)/(N))*o[k])
            s[k+N//2] = 1/2*(e[k] - np.exp((2j*np.pi*k)/(N))*o[k])
    return s
```

---

4 - Application à l'analyse d'un son

Les sons que vous allez traiter sont des signaux de codes DTMF (Dual Tone Multiple Frequency) utilisés pour distinguer par leur doublet de fréquences les touches numérotées d'un clavier téléphonique (DTMF sur wikipedia).

L'appuie sur une touche génère donc un son composé de deux fréquences, selon le tableau suivant :

	1 209 Hz	1 336 Hz	1 477 Hz	1 633 Hz
697 Hz	1	2	3	A
770 Hz	4	5	6	B
852 Hz	7	8	9	C
941 Hz	*	0	#	D

Travailler avec un fichier son sous Python

Deux modules sont pratiques pour lire et extraire les données importantes d'un fichier son au format `.wav`.

```
import soundfile as sf
import sounddevice as sd
```

Lire et extraire des données du fichier `son.wav`

```
s, fe = sf.read(r'son.wav')
#s est le signal
#fe la frequence d echantillonnage du signal
#Le nombre d echantillon est donne par s.shape[0] pour la voie 0
N = s.shape[0]
```

Écouter le fichier `son.wav`

```
#Pour ecouter le son
sd.play(s,fe)
```

Analyser le son `son1.wav`

L'échantillonnage est-il en 2^n ?

Le nombre d'échantillons N $\neq 2^n$. Ici, `N = 22051`. On va ajouter des '0' pour avoir $N = 2^n$. Initialement p est tel que $N = 2^p$ avec p non entier. On a donc $\log_2(N) = p.\underbrace{\log_2(2)}_{=1}$. Dans notre exemple, $p = 14.42855646219518$

On prend pour n la valeur entière immédiatement supérieure :

$$n = \text{int}(p) + 1 = \text{int}(\log_2(N)) + 1$$

soit finalement, $n = 15$.

Ajoutons ensuite au signal les zéros manquants.

30 - Construire une fonction `puissance2(liste)`

1. qui prend en entrée une liste `liste`
2. et renvoie
   (a) le nombre de points en 2^n avec n l'entier supérieur à p de la nouvelle liste
   (b) ainsi que la nouvelle liste des valeurs du signal complétée des '0' nécessaires.

```
def puissance2(liste):
    N = len(liste)
    N_nouv = 2**(int(np.log2(N))+1)
    N_zeros=np.zeros(N_nouv-N)
    #Ajouter des N_zeros 0 pour avoir N=2^n
    return N_nouv,np.concatenate((liste,N_zeros))

N_nouv,s_nouv = puissance2(s)
```

De quelle touche s'agit-il ?

Analyser le son `son3.wav`

31 - En prenant les mêmes précautions que précédemment, indiquer de quelle(s) touche(s) il s'agit.

Analyser le son `son4.wav`

32 - Déduire de l'analyse spectrale le numéro de téléphone tapé sur le clavier DTMF?

---

1. Les premières difficultés
   (a) Le nombre d'échantillons $N \neq 2^n$. Ici, `N=55459`.
   (b) A l'écoute, le signal est une succession de 10 sons.
2. Les solutions envisagées
   (a) Ajouter des '0' pour avoir $N = 2^n$ en se servant de `puissance(liste)`
   Faisons la FFT du signal complété des zéros adéquates.

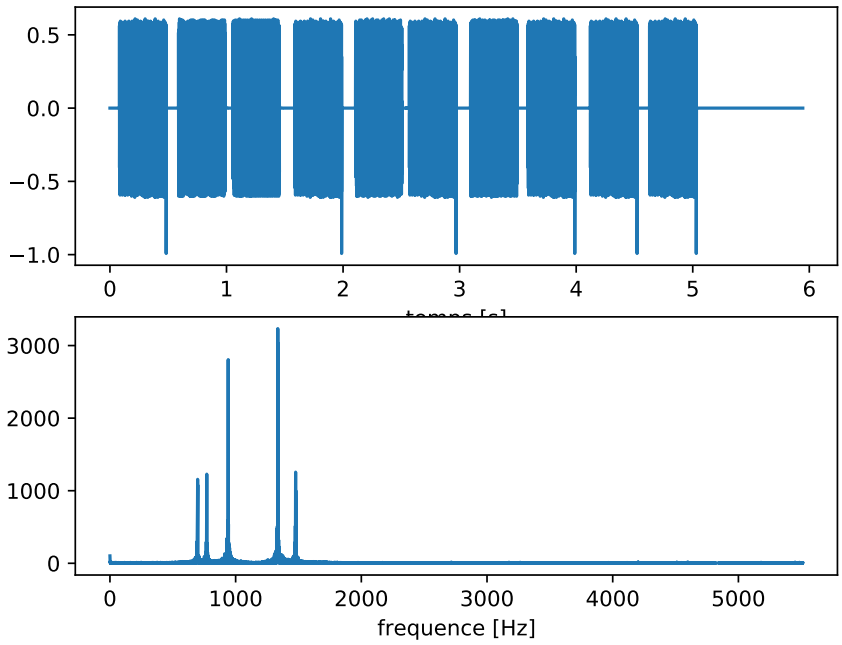


FIGURE 4 – Signal complété de '0' pour avoir $N = 2^n$.

   (b) Travailler en isolant les différents signaux. On prendra des tranches du signal initial – 10 tranches donc – de largeur le nombre d'échantillons initial divisé par dix : $N/10$.

```
#on travaille sur une tranche (le signal est coupe en 10 tranches)
tranche=int(len(s)/10)
N=tranche
k = 0
s=s[k*tranche:(k+1)*tranche]
```

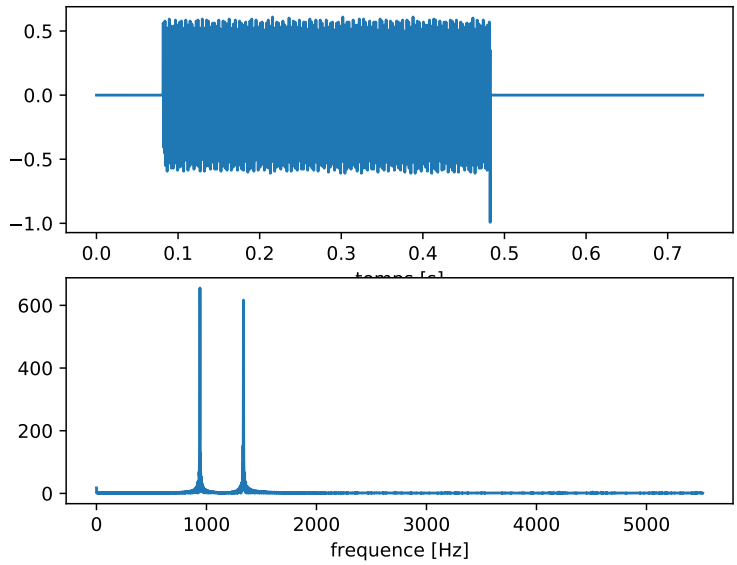


FIGURE 5 – Pour la première tranche : touche `0 (941 Hz, 1336 Hz)`

Nous avons les touches :
— 0 (941 Hz, 1336 Hz)
— 3 (694 Hz, 1476 Hz)
— 2 (698 Hz, 1330 Hz)
— 0 (941 Hz, 1336 Hz)
— 6 (771 Hz, 1476 Hz)
— 0 (941 Hz, 1336 Hz)
— 5 (770 Hz, 1336 Hz)
— 0 (941 Hz, 1336 Hz)

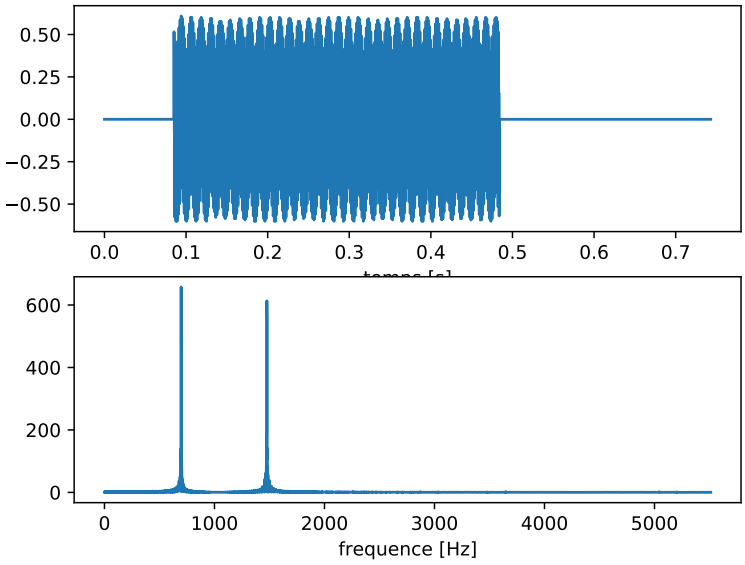


FIGURE 6 – Pour la seconde tranche : touche `3 (694 Hz, 1476 Hz)`

5 - Annexe : numpy

Créer une matrice de zéros complexes

```
M = np.zeros((n,m),dtype =complex)
```

```
>>> np.zeros((3,2),dtype = complex)
array([[ 0.+0.j,  0.+0.j],
       [ 0.+0.j,  0.+0.j],
       [ 0.+0.j,  0.+0.j]])
```

Accéder à une ligne, une colonne, une valeur d'une matrice

```
>>> A=np.array([[1.0-2.j,3.0+1.j],[0.1+0.2j,0.0+0.0j],
       [ 0.+0.j,  0.+0.j]])
>>> A
array([[1. -2.j , 3. +1.j ],
       [0.1+0.2j, 0. +0.j ],
       [0. +0.j , 0. +0.j ]])
```

`A[1,:]` renvoie les valeurs de toutes les colonnes de la ligne d'indice 1 (la seconde ligne), donc la seconde ligne.

```
>>> A[1,:]
array([0.1+0.2j, 0. +0.j ])
```

`A[:,0]` renvoie les valeurs de toutes les lignes de la colonne d'indice 0 (la première colonne), donc la première colonne.

```
>>>A[:,0]
array([1. -2.j , 0.1+0.2j, 0. +0.j ])
```

`A[0,1]` renvoie la valeur de la lignes d'indice 0 et de la colonne d'indice 1.

```
>>> A[0,1]
(3+1j)
```

Produit matricielle

Si `A` est une matrice $(n \times m)$ et que `B` une matrice de taille $(m \times p)$, le produit matriciel `AB` s'écrit avec numpy.

```
np.dot(A,B)
```

```
>>> A=np.array([[1.0-2.j,3.0+1.j],[0.1+0.2j,0.0+0.0j]])
>>> A
array([[1. -2.j , 3. +1.j ],
       [0.1+0.2j, 0. +0.j ]])

>>> B = np.array([[1.5-2.5j],[1.0+0.7j]])
>>> B
array([[1.5-2.5j],
       [1. +0.7j]])

>>> C = np.dot(A,B)
>>> C
array([[-1.2 -2.4j ],
       [ 0.65+0.05j]])
```

Matrice conjuguée

Si $\mathcal{A}$ est une matrice, sa matrice conjuguée $\overline{\mathcal{A}}$ est donnée par la méthode `conjugate()`

```
A.conjugate()
```

Ajouter des éléments à une liste `numpy`

Attention, ici, `A` et `B` sont des listes `numpy` et non des tableaux à une ligne.

```
np.concatenate((A,B))
```

```
>>> A=np.array([1.0-2.j,3.0+1.j,1.5-2.5j,0.0+0.j])
>>> A
array([1. -2.j , 3. +1.j , 1.5-2.5j, 0. +0.j ])
>>> B = np.array([1.5-2.5j,1.0+0.7j])
>>> B
array([1.5-2.5j, 1. +0.7j])
>>> C = np.concatenate((A,B))
>>> C
array([1. -2.j , 3. +1.j , 1.5-2.5j, 0. +0.j , 1.5-2.5j, 1. +0.7j])
```

Créer une liste `numpy`

```
np.arange(debut,fin,pas)
```

```
>>> L=np.arange(0,2,0.3)
>>> L
array([0. , 0.3, 0.6, 0.9, 1.2, 1.5, 1.8])
```