Rapport TP

TL Traitement Du Signal

Réalisé par:

Nabil ABDELOUAHED
Yann MARTIN

June 2, 2025

Contents

1 Partie 1: Quelques opérations de base sur les signaux

1.1 Signal numérique de synthèse

1.1.1 Génération du signal

Un signal sinusoïdal de fréquence f_0 est généré par la fonction suivante:

$$x[n] = \sin\left(2\pi f_0 \frac{n}{f_e}\right)$$

où f_e est la fréquence d'échantillonnage et N le nombre d'échantillons.

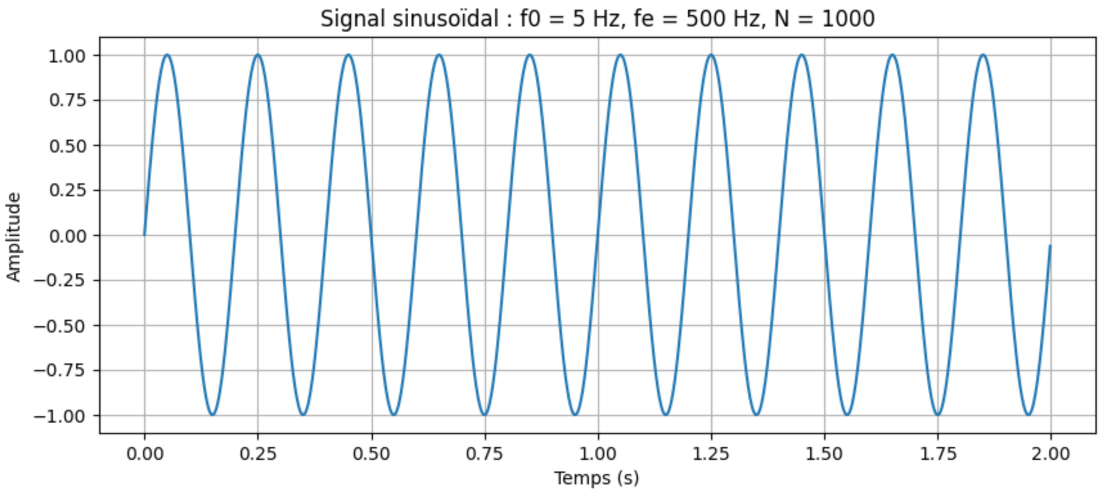


Figure 1: Signal sinusoïdal échantillonné

1.1.2 Énergie et puissance

L'énergie d'un signal discret $x[n]$ est donnée par :

$$E = \sum_{n=0}^{N-1} x[n]^2$$

Et la puissance moyenne par :

$$P = \frac{1}{N} \sum_{n=0}^{N-1} x[n]^2$$

Considérons un signal sinusoïdal discret de la forme :

$$x[n] = A \cdot \sin\left(2\pi f_0 \frac{n}{f_e}\right)$$

La puissance moyenne théorique d'un signal périodique est calculée par la formule:

$$P = \lim_{N\to\infty} \frac{1}{N} \sum_{n=0}^{N-1} x[n]^2$$

En utilisant l'identité trigonométrique :

$$\sin^2(\theta) = \frac{1 - \cos(2\theta)}{2}$$

on obtient :

$$x[n]^2 = A^2 \cdot \frac{1 - \cos\left(4\pi f_0 \frac{n}{f_e}\right)}{2}$$

Ainsi, la puissance devient :

$$P = \lim_{N\to\infty} \frac{1}{N} \sum_{n=0}^{N-1} A^2 \cdot \frac{1 - \cos\left(4\pi f_0 \frac{n}{f_e}\right)}{2} = \lim_{N\to\infty} \frac{A^2}{2} - \frac{A^2}{2N} \sum_{n=0}^{N-1} \cos\left(4\pi f_0 \frac{n}{f_e}\right) = \frac{A^2}{2}$$

Donc dans notre cas la puissance moyenne théorique est égale à 0.5.

Pour la puissance moyenne calculée numériquement pour le signal échantillonné on a la méme formule mais sans la limite:

$$P = \frac{A^2}{2} - \frac{A^2}{2N} \sum_{n=0}^{N-1} \cos\left(4\pi f_0 \frac{n}{f_e}\right)$$

Cas idéal : si N est un multiple entier de la période du signal (i.e., N couvre un nombre entier de périodes), alors la somme des cosinus s'annule :

$$\sum_{n=0}^{N-1} \cos\left(4\pi f_0 t\right) = 0 \quad \Rightarrow \quad P = \frac{A^2}{2}$$

Cas général : si N n'est pas un multiple exact de la période, la somme ne s'annule pas et on observe une légère déviation de la puissance par rapport à $\frac{A^2}{2}$. Cela est dû au fait que le signal est tronqué entre deux points non symétriques.

En variant N on trouve plusieurs valeurs de la puissance moyenne qui restent proches de la valeur théorique:

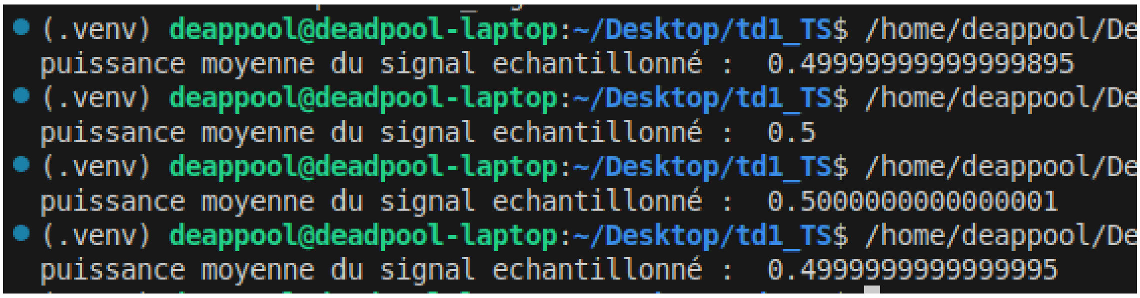


Figure 2: Énergie et puissance du signal

1.1.3 Quantification

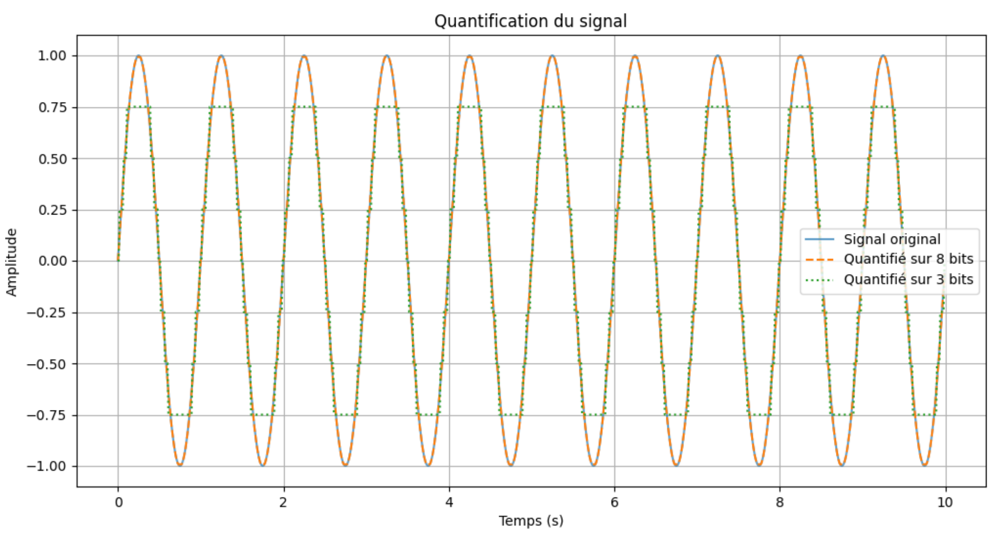


Figure 3: Quantification du signal à 3 et 8 bits

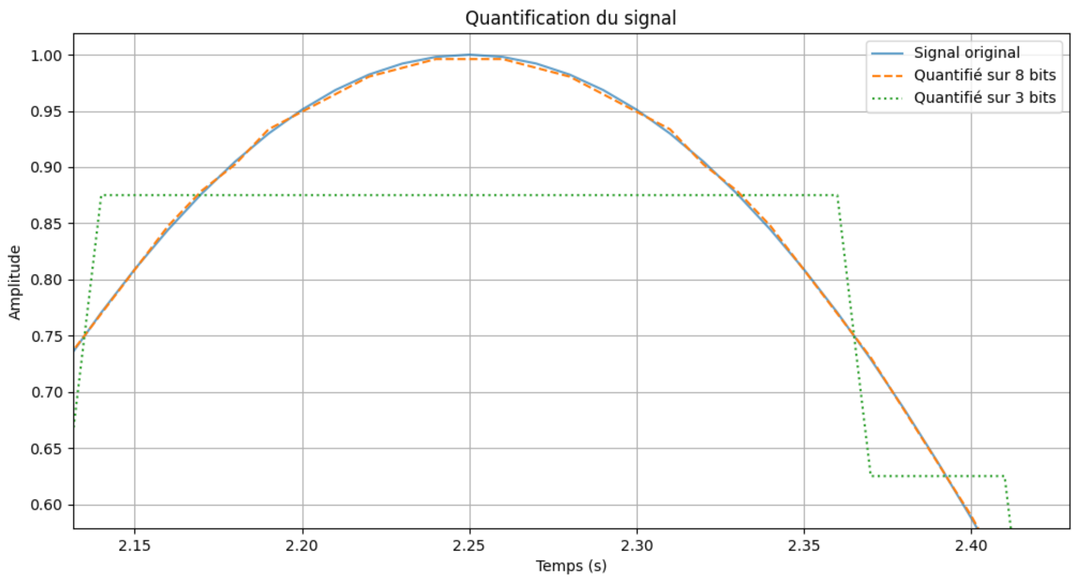


Figure 4: Zoom sur la quantification du signal à 3 et 8 bits

Pour la quantification à 3 bits, on retrouve bien 8 niveaux de quantification.
Le signal à 8 bits suit mieux la forme continue du signal d'origine mais on voit quand méme quelques erruers. À 3 bits, les marches sont plus visibles et le signal produit est significativement moins fidéle au signal d'origine.

SNR (Signal-to-Noise Ratio) :

$$\text{SNR} = 10 \log_{10} \left(\frac{E_{\text{signal}}}{E_{\text{bruit}}} \right)$$

```
Énergie du bruit de quantification (8 bits): 0.006400
SNR (8 bits): 49,92 dB

Énergie du bruit de quantification (3 bits): 6.236655
SNR (3 bits): 19,82 dB
```

Figure 5: SNR pour chaque niveau de quantification

On remarque que l'energie du bruit est plus élevée pour le signal quantifié à 3 bits. Le résultat est logique car on voit sur le graphe que ce signal est plus éloigné du signal d'origine comparé au signal quantifié à 8 bits. On a la méme conclusion en raisonant sur le SNR: SNRq8 > SNRq3.

1.2 Signal audio

1.2.1 Enregistrement

Les mots « Bonjour » et « ChatGpt » ont été enregistrés via Audacity (voir figure 6).

1.2.2 Restitution à différentes fréquences

L'audio est lu à f_e, $2f_e$ et $\frac{f_e}{2}$.
Effets observés :

- **Durée :** doubler la fréquence de restitution divise la durée par deux (voix accélérée), tandis que la diviser par deux double la durée (voix ralentie).

- **Hauteur :** multiplier la fréquence de restitution rend la voix plus aiguë (fréquences doublées), la diminuer la rend plus grave (fréquences divisées).

Lorsque la fréquence de restitution est trop grande ou trop petite comparée à la fréquence d'échantillonnage, le son devient incompréhensible. Le son est trop rapide ou trop lent, mais aussi on ne reconnaît plus les mots. Cela s'explique notamment par le déplacement des formants, c'est-à-dire des pics de résonance caractéristiques des voyelles et consonnes, qui changent de position dans le spectre. Leur modification rend la parole méconnaissable, car ce sont eux qui permettent d'identifier les sons du langage.

1.2.3 Quantification du signal audio

- À 3 bits : le son devient rugueux et très bruité, fortement altéré.
- À 8 bits : la voix reste compréhensible mais moins naturelle.

1.2.4 Extraction et séparation de mots

Après repérage visuel, les deux mots ont été extraits via tranches temporelles.

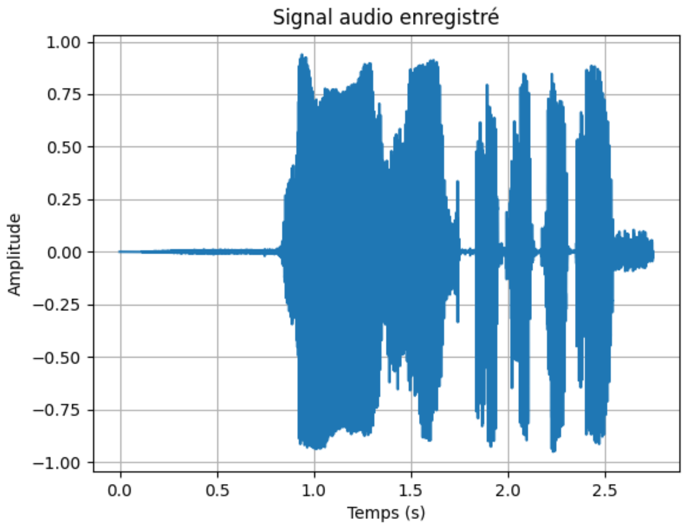
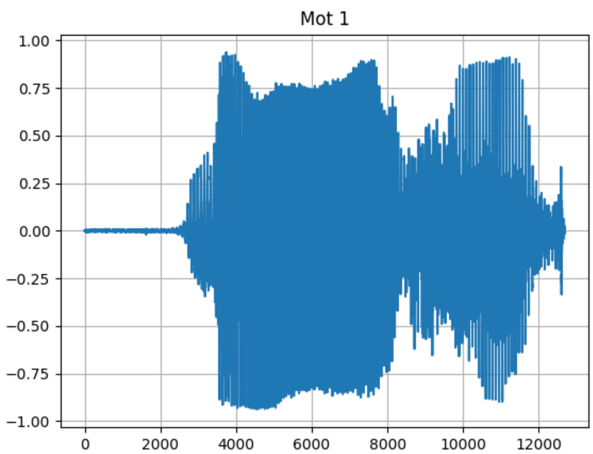
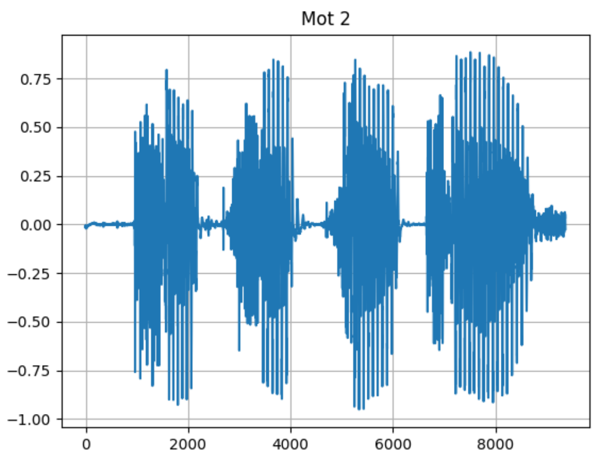


Figure 6: Signal audio enregistré



(a) Mot 1: "Bonjour"



(b) Mot 2: "ChatGpt"

Figure 7: Séparation des mots dans le signal audio

2 Partie 2: Classification des signaux

2.1 Exemple de calcul théorique

Soit un signal sinusoïdal discret défini par :

$$x[n] = A\sin(2\pi f n T_e)$$

L'autocorrélation théorique du signal discret est définie par :

$$R_{xx}[k] = \lim_{N\to\infty} \frac{1}{N} \sum_{n=0}^{N-1} x[n] \cdot x[n+k]$$

En remplaçant l'expression de $x[n]$:

$$R_{xx}[k] = \lim_{N\to\infty} \frac{1}{N} \sum_{n=0}^{N-1} A\sin(2\pi f n T_e) \cdot A\sin(2\pi f (n+k) T_e)$$

$$= A^2 \cdot \lim_{N\to\infty} \frac{1}{N} \sum_{n=0}^{N-1} \sin(2\pi f n T_e) \cdot \sin(2\pi f n T_e + 2\pi f k T_e)$$

En utilisant l'identité trigonométrique :

$$\sin(a)\sin(b) = \frac{1}{2}[\cos(a-b) - \cos(a+b)]$$

On a :

$$\sin(2\pi f n T_e) \cdot \sin(2\pi f n T_e + 2\pi f k T_e) = \frac{1}{2}\left[\cos(2\pi f k T_e) - \cos(4\pi f n T_e + 2\pi f k T_e)\right]$$

$$\Rightarrow R_{xx}[k] = \frac{A^2}{2}\cos(2\pi f k T_e)$$

car la somme de $\cos(4\pi f n T_e + \cdot)$ sur une grande fenêtre N tend vers 0.
Conclusion : l'autocorrélation théorique du signal sinusoïdal discret est donnée par :

$$\boxed{R_{xx}[k] = \frac{A^2}{2}\cos(2\pi f k T_e)}$$

où :

- A est l'amplitude du signal,
- f est la fréquence du signal en Hz,
- $T_e = \frac{1}{f_e}$ est la période d'échantillonnage,
- $k \in \mathbb{Z}$ est le décalage (lag) discret.

2.2 Programmation

On calcule l'autocorrélation pour un signal sinusoïdal échantillonné en utilisant la formule :

$$R_{xx}[k] = \frac{1}{N} \sum_{n=0}^{N-1} x[n] \cdot x[n+k]$$

```python
def autocorrelation_manual(x):
N = len(x)
r = np.zeros(2*N - 1)
lags = np.arange(-N + 1, N)
for k in range(-N + 1, N):
    somme = 0
    for n in range(N - abs(k)):
        somme += x[n] * x[n + k] if k >= 0 else x[n - k] * x[n]
    r[k + N - 1] = somme
return lags, r
```

On compare avec l'autocorrelation calculée avec numpy et on trace la différence entre les deux résultats pour visualiser les écarts. Pour convertir les abcisses en secondes, on utilise la formule :

$$\mathrm{t} = \mathrm{k}.T_e = \frac{\mathrm{k}}{f_e}$$

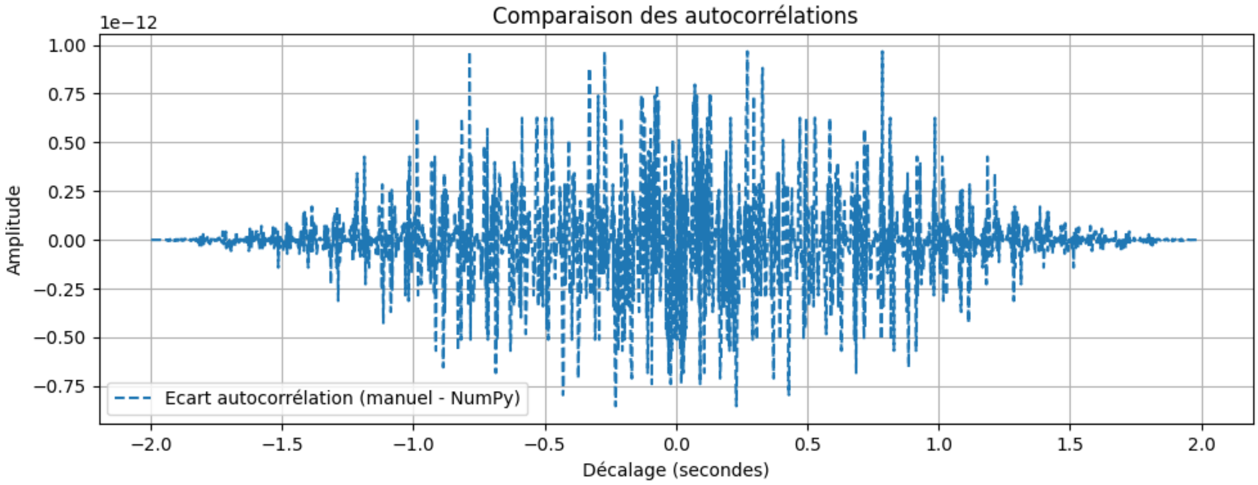


Figure 8: Ecart entre l'autocorrélation manuelle et celle de numpy

L'ecart est trop petit entre les deux méthodes (de l'ordre de 10^{-12}). Ceci est dû à la précision numérique des calculs et au fait que l'autocorrelation numpy est calculée différemment en utilisant plusieurs techniques d'optimisation.

On calcule le rapport des énergies entre la différence entre les deux méthodes et l'autocorrelation de numpy:

$$\text{rapport} = \frac{\sum_{k=0}^{N-1} \text{diff}[k]^2}{\sum_{k=0}^{N-1} \text{autocorr}[k]^2}$$

On trouve plusieurs valeurs de rapport pour différentes valeurs de N. Ceci est dû à la variation de l'énergie du signal en fonction de N, car l'autocorrelation et l'énergie sont sensibles à la longueur du signal (somme de 0 à N-1). On rajoute donc des valeurs au numérateur et au dénominateur dans le calcul de ce rapport (qui ne sont pas équivalentes vu l'écart entre les deux méthodes) ce qui expliques des valeurs de rapport différentes.

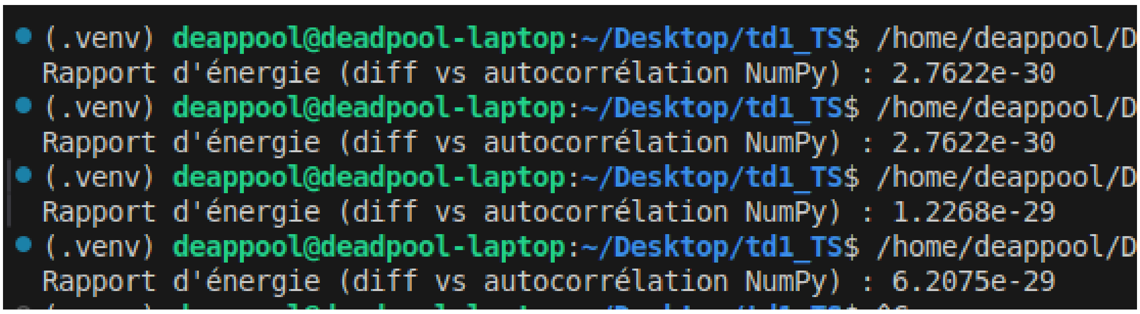


Figure 9: Valeurs du rapport pour plusieurs valeurs de N

2.3 Application à la classification de quelques signaux simples

On génére les signaux demandés avec le code suivant :

```
# Parametres communs
fe = 11000           # frequence d'echantillonnage
duration = 1      # duree en secondes
t = np.linspace(0, duration, int(fe * duration), endpoint=False)

# 1. Signal sinusoidal de 200 Hz
f = 200
sinus = np.sin(2 * np.pi * f * t)

# 2. Signal triangulaire centre en 0 (serie de Fourier avec 10 termes)
tri = np.zeros_like(t)
for k in range(1, 11):
    n = 2 * k - 1  # uniquement les harmoniques impaires
    tri += ((-1)**((k+1)) / n**2) * np.sin(2 * np.pi * n * f * t)
tri *= (8 / (np.pi**2))  # normalisation serie de Fourier

# 3. Bruit blanc gaussien
taille = fe  # 1 seconde
bruit = np.random.randn(taille)
bruit /= np.max(np.abs(bruit))  # normalisation pour eviter la saturation
```

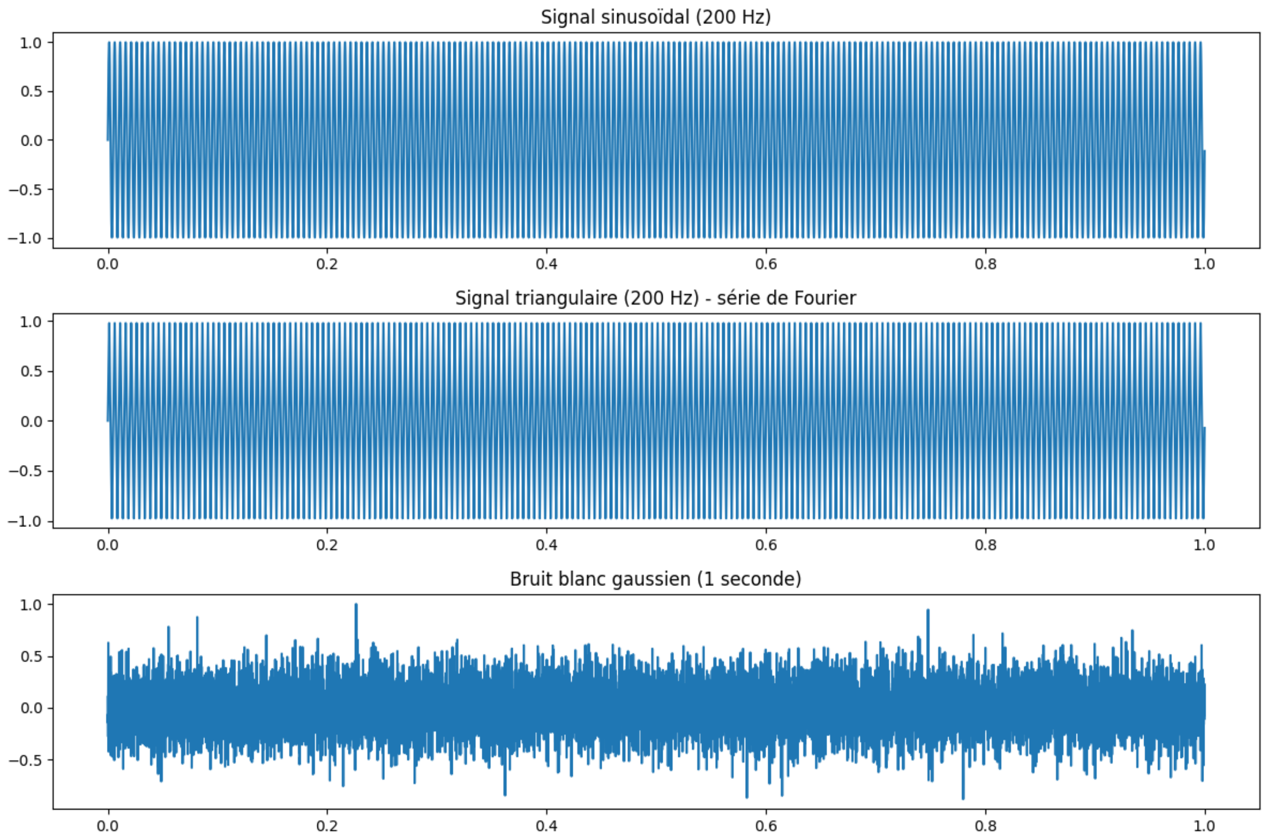


Figure 10: Signaux générés

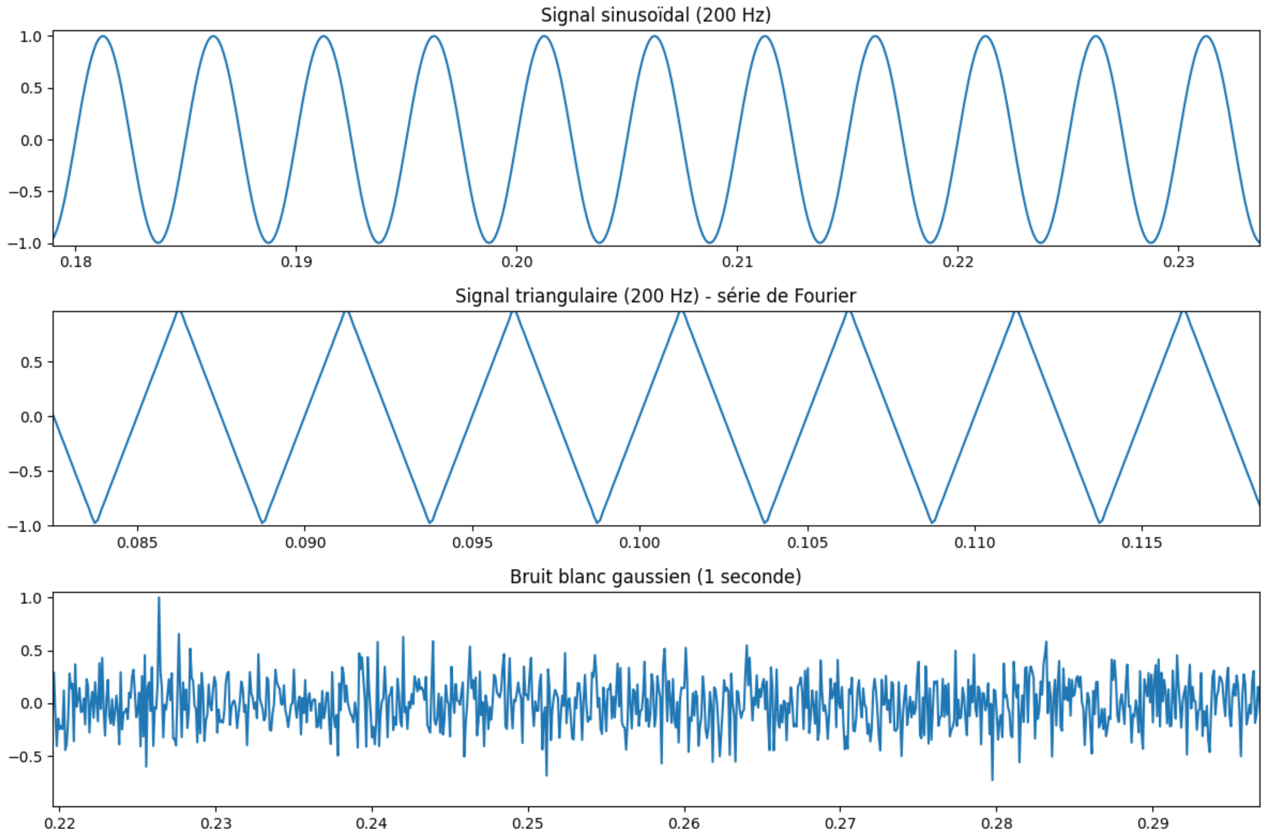


Figure 11: Signaux générés (zoom)

On calcule l'autocorrélation pour chaque signal sur une partie de durée de 30ms. C'est une durée typique en traitement du signal pour analyser la parole (équilibre entre résolution temporelle et fréquence). Elle est suffisante pour voir une structure (formants, périodicité...) sans mélanger des parties trop différentes du signal. On trace également l'autocorrélation du signal enregistré sur deux tranches de 30ms (et 80ms) à des moments différents (1000ms et 1500ms). Cela nous permettera de de comparer l'autocorrélation à différents moments afin de mettre en oeuvre la non-stationnarité du signal.

Un signal est dit **stationnaire** si ses propriétés statistiques, telles que la moyenne, la variance et l'autocorrélation, ne varient pas dans le temps.

L'analyse des autocorrélations montre que :

- Le signal vocal « aa » est **non stationnaire** car ses caractéristiques changent entre différentes tranches temporelles (voix humaine évolue dans le temps).
- Le signal de bruit blanc gaussien est **stationnaire**, ses propriétés statistiques étant constantes dans le temps (Pic au centre, décroissance rapide).
- Les signaux sinusoïdal et triangulaire, synthétisés sur quelques périodes, sont également **stationnaires** car ils sont périodiques et leurs statistiques ne varient pas dans le temps.

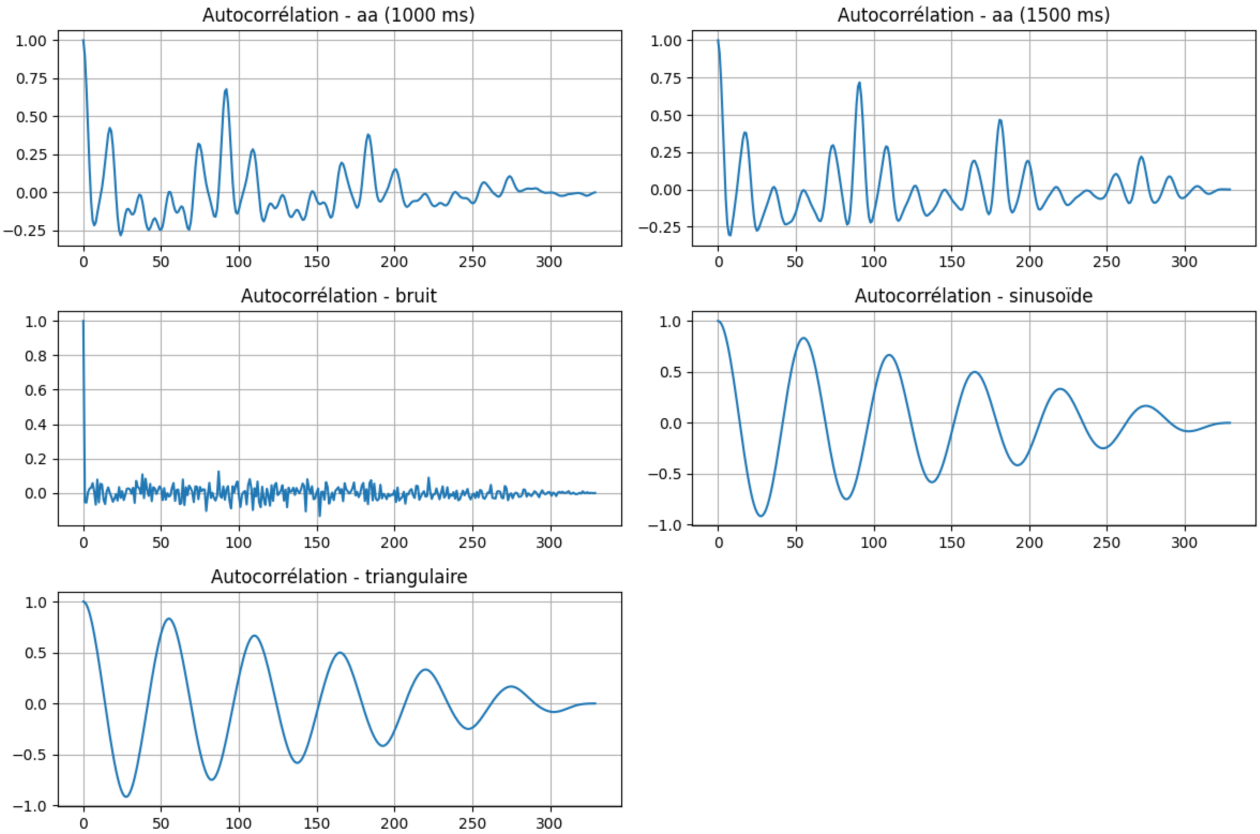


Figure 12: autocorrélation des signaux sur 30ms

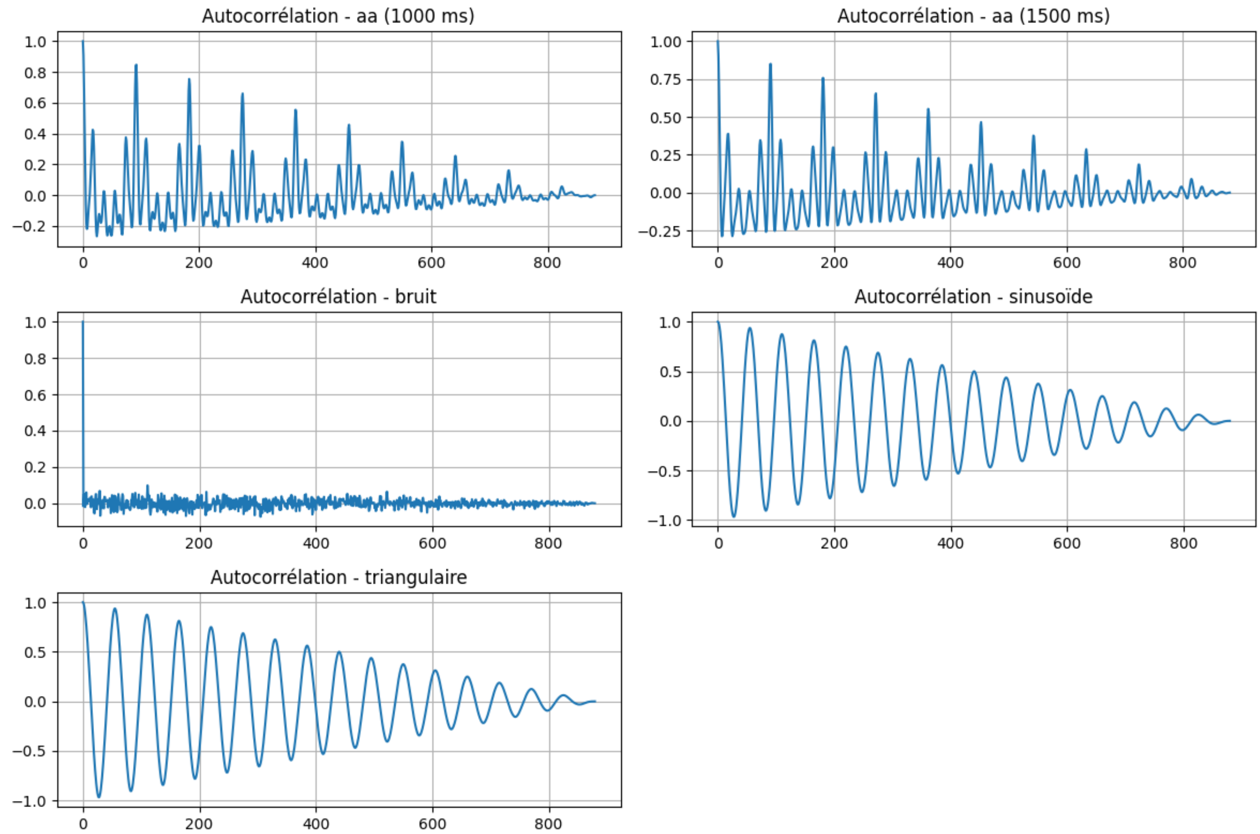


Figure 13: autocorrélation des signaux sur 80ms

2.4 Classification de signaux de parole voisés ou non voisés

Le signal est découpé en tranches de 30 ms pour analyser leur autocorrélation. Les résultats montrent clairement deux types de comportements :

- **Parties non voisées** (« chhhhh ») : l'autocorrélation est bruitée, sans périodicité marquée, caractéristique des sons fricatifs.
- **Parties voisées** (« aaaa ») : l'autocorrélation présente des pics réguliers, révélant une structure périodique due à la vibration des cordes vocales.

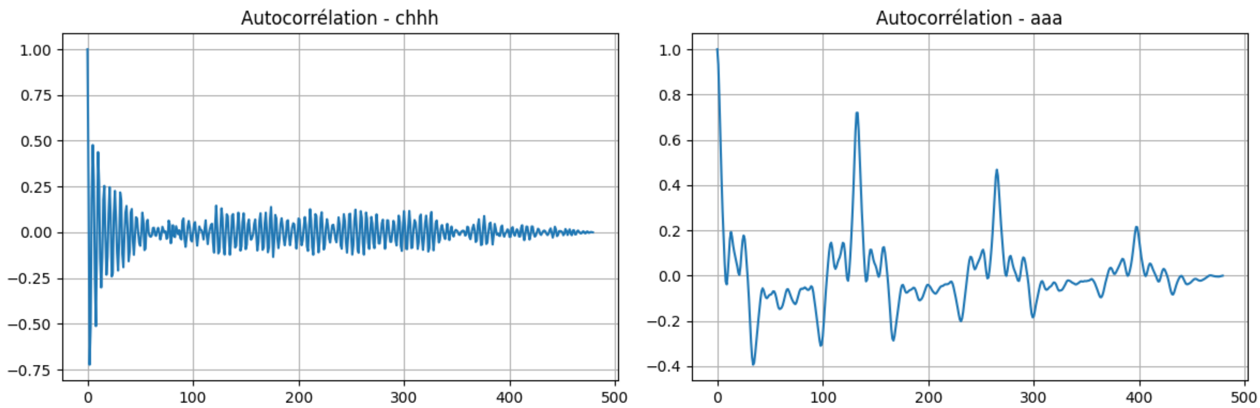


Figure 14: Autocorrélation de différentes tranches du mot « chat » : (gauche) non voisée, (milieu) transition, (droite) voisée.

L'autocorrélation permets d'estimer la fréquence fondamentale f_0 d'un signal voisé. On s'attend à ce qu'un signal périodique présente un maximum secondaire au retard correspondant à sa période fondamentale $T_0 = \frac{1}{f_0}$.

Pour identifier les pics secondaires dans l'autocorrélation, nous utilisons la fonction `find_peaks` de `scipy.signal`. Afin d'éviter de détecter des pics non significatifs dus au bruit ou à des résonances (formants), le paramètre **height** qui correspond à la hauteur minimale doit être soigneusement choisis pour ignorer les pics trop faibles dus au bruit. Nous avons fixé ce seuil à 0,3 après normalisation.

Pour la tranche correspondant à la voyelle *« a »* dans l'enregistrement (partie voisée), nous obtenons après traitement:

- Indices du premier pic détecté : 133
- On en déduit une fréquence fondamentale :

$$f_0 = \frac{f_e}{\text{lag}} = \frac{16\,000}{133} \approx 120{,}3 \text{ Hz}$$

Ce résultat est conforme aux plages typiques pour une voix masculine (85–180 Hz). Il est important de noter que la fréquence fondamentale ne caractérise pas la voyelle elle-même, mais le locuteur.

3 Partie 3: Aspects fréquentiels

3.1 Echantillonnage

Le théorème de Nyquist-Shannon stipule qu'un signal bande limitée, dont la fréquence maximale est f_0, peut être parfaitement reconstruit à partir de ses échantillons si la fréquence d'échantillonnage vérifie :

$$f_e > 2f_0$$

La fréquence $f_e = 2f_0$ est appelée **fréquence de Nyquist** et constitue la limite minimale pour éviter le phénomène de repliement spectral (*aliasing*).

On échantillonne donc le signal selon trois cas distincts :

- **Cas 1 :** $f_e = 2f_0$ (fréquence de Nyquist),
- **Cas 2 :** $f_e = 5f_0$ (fréquence supérieure),
- **Cas 3 :** $f_e = 0{,}8f_0$ (fréquence inférieure).

On vois que pour les deux premiers cas, le signal échantillonné est fidèle à la forme du signal d'origine. En revanche, pour le troisième cas, l'échantillonnage à une fréquence inférieure à la fréquence de Nyquist entraîne un repliement spectral, rendant le signal méconnaissable.

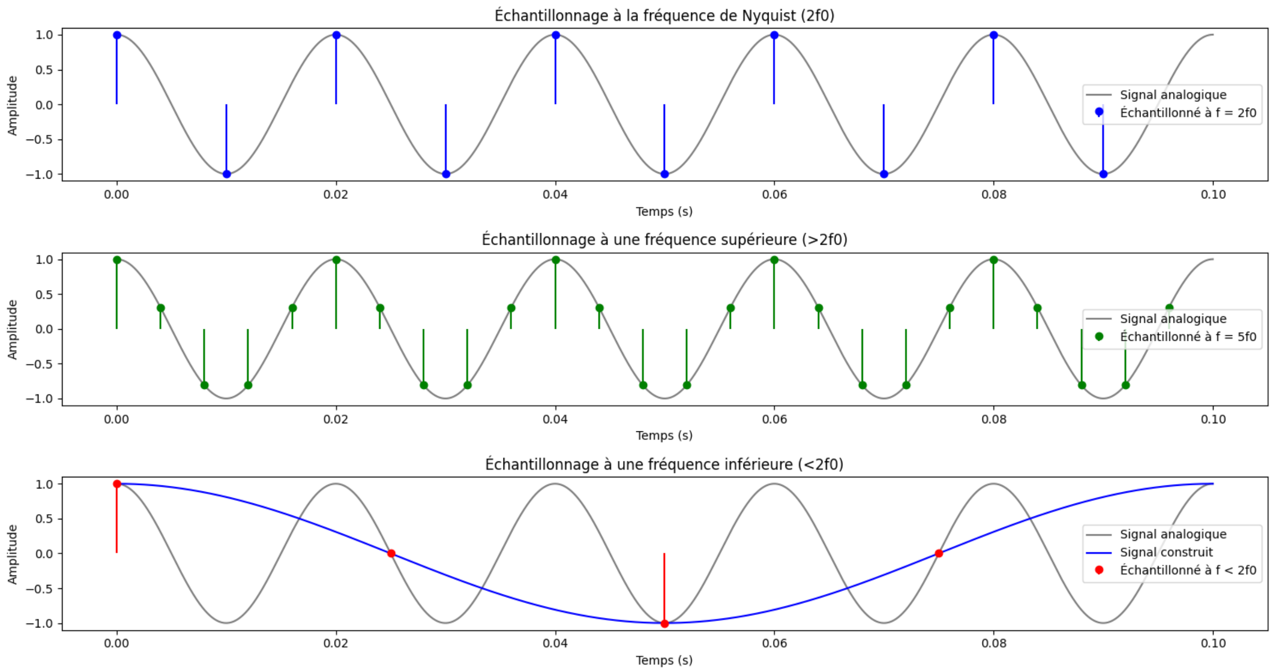


Figure 15: Signaux échantillonnés

3.2 La Transformée de Fourier discrète (TFD)

Nous avons implémenté une fonction en `Python` permettant de calculer et tracer la densité spectrale d'énergie d'un signal extrait d'un fichier `.wav`. La fonction lit le fichier audio, extrait une tranche du signal à partir d'un indice donné, applique une fenêtre de Hanning pour atténuer les effets de bord, puis calcule la transformée de Fourier rapide (`FFT`) sur 2^n points. La densité spectrale est obtenue en prenant le module au carré des coefficients de la FFT (représentant l'énergie en fréquence), normalisée et convertie en décibels (dB) selon la formule :

$$P_{\mathrm{dB}}(f) = 10\log_{10}\left(\frac{|\mathrm{FFT}(f)|^2}{\max(|\mathrm{FFT}(f)|^2)} + \varepsilon\right),$$

avec ε un terme petit (10^{-12}) pour éviter les problèmes numériques liés au logarithme de zéro.

La courbe obtenue est symétrique par rapport à l'axe des fréquences, car la FFT calcule les coefficients complexes pour les fréquences positives et négatives. On ne garde que la moitié positive du spectre.

Le choix de 2^n (taille de la FFT) influence la **résolution fréquentielle**. Plus n est grand, plus les fréquences sont discrétisées finement, ce qui permet de mieux isoler les pics correspondant aux composantes harmoniques du signal.

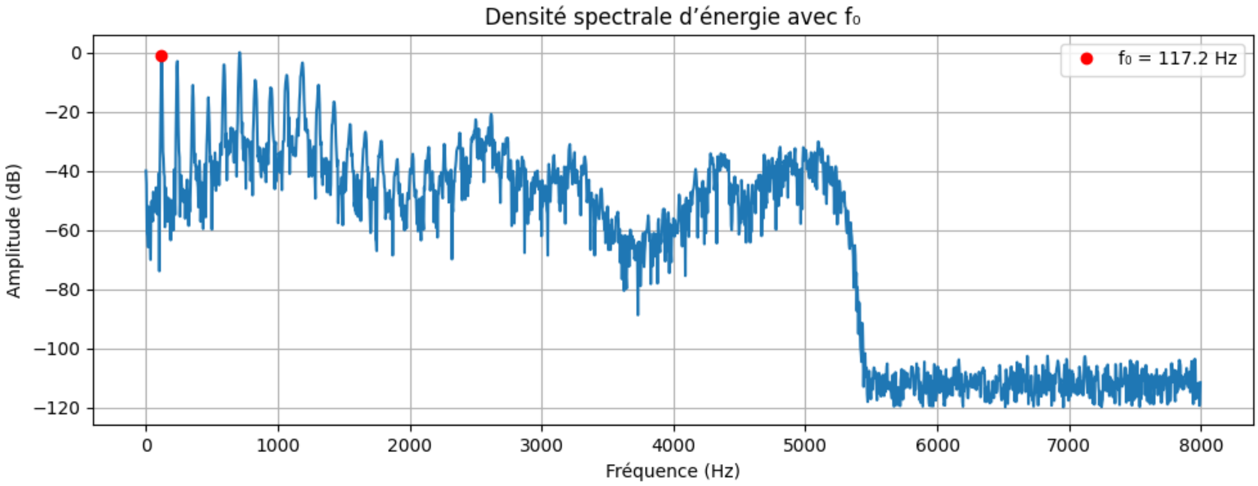


Figure 16: FFT

Sur le spectre obtenu, on observe une structure en pics caractéristiques d'un signal périodique voisé (ici le phonème *aaa*). La fréquence fondamentale correspond au premier pic significatif au-dessus du bruit. À partir du spectre, nous détectons une fréquence fondamentale de **117,2 Hz**.

Ce résultat est très proche de celui obtenu par l'autocorrélation dans le domaine temporel, où nous avions trouvé une fréquence de **120,3 Hz** (3.1 Hz de différence). Cette légère différence peut s'expliquer par :

- la résolution fréquentielle limitée de la FFT (dépendante de la taille de la tranche),
- les effets de la fenêtre de Hanning qui modifie légèrement le contenu fréquentiel,
- et les arrondis ou approximations dans la localisation du pic.

Malgré cela, les deux méthodes convergent vers une estimation très cohérente de la fréquence fondamentale, typique d'une voix d'homme (comprise entre 85 et 180 Hz).

Le *zero-padding* consiste à compléter un signal par des zéros à la fin. Cette opération n'ajoute aucune nouvelle information, mais elle permet d'augmenter artificiellement la longueur du signal, ce qui améliore la **résolution fréquentielle** du spectre après transformation de Fourier.

Nous avons implémenté une fonction `zero_padding` en `Python`, prenant en entrée le signal initial et le nombre de zéros à ajouter, et renvoyant le signal étendu.

```
def zero_padding(signal, nb_zeros):
return np.hstack((signal, np.zeros(nb_zeros)))
```

Nous avons ensuite comparé les densités spectrales d'énergie (obtenues par FFT) avant et après zero-padding. Le résultat montre que :

- les **positions des pics** dans le spectre restent inchangées (le contenu fréquentiel est le même),
- mais les courbes sont **plus lisses** comme on peut le voir dans la figure 18 et les pics **mieux localisés** après padding, ce qui facilite l'estimation des fréquences fondamentales,
- le zero-padding agit donc comme une **interpolation en fréquence**.

Cela est particulièrement utile pour distinguer des pics proches ou évaluer plus précisément une fréquence fondamentale.

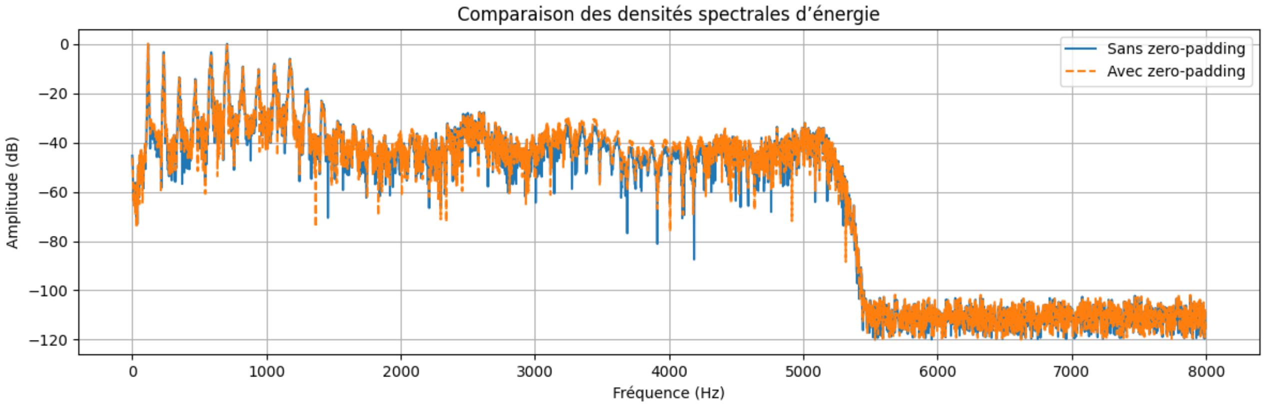


Figure 17: FFT avec et sans zero-padding

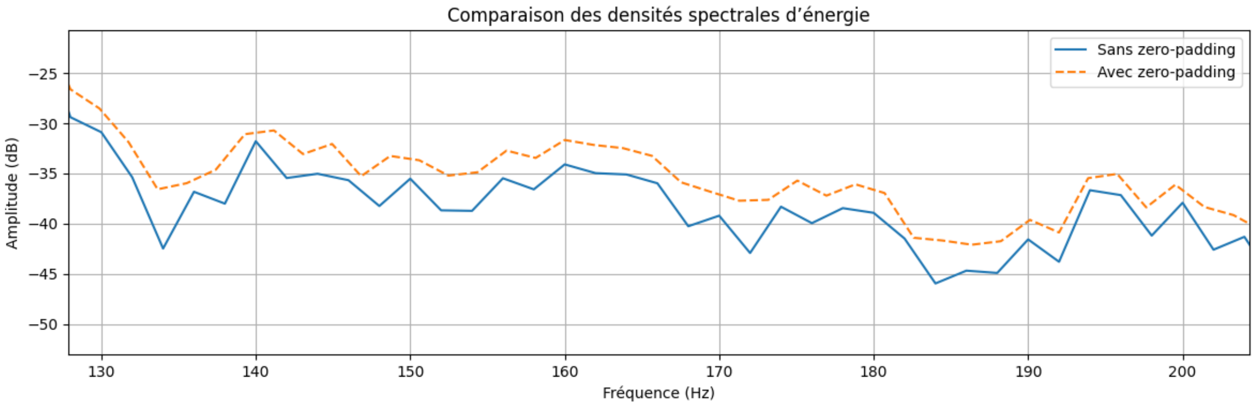


Figure 18: FFT avec et sans zero-padding (zoom)

On peut faire directement du zero-padding dans la fonction **np.fft.fft(x, n)**.

- Si n est supérieur à la longueur de x, la fonction ajoute des zéros à la fin de x jusqu'à atteindre la longueur n.
- Si n est inférieur ou égal à la longueur de x, la fonction tronque x à la longueur n.

3.3 Changement de fréquence d'échantillonnage