Nom : LEVEQUE
Prénom : Clément
Numéro de candidat : 29574

Sommaire

- But :
  - Recenser les espèces d'oiseaux dans une ville
- Processus :
  - Enregistrements
  - Extractions des informations utiles
  - Reconnaissance

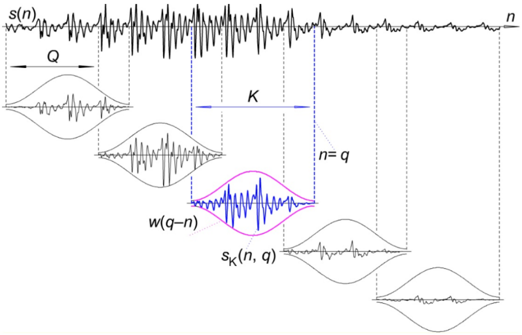
Coefficients de fréquence Mel

- Reconnaissance de parole
- Reconnaissance de son
- Dépend du temps

- Acquisition

- Acquisition
- Centrage :
  $f_n(x) = f(x) - m$
  m moyenne, f fonction du signal et f_n fonction centrée

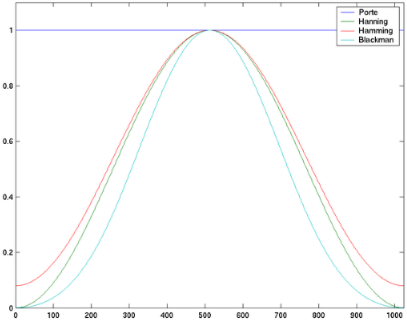
- Acquisition
- Centrage :
  $f_n(n) = f(n) - m$
  m moyenne, f fonction du signal et f_n fonction centrée
- Pré-accentuation :
  $f_a(n) = f(n) - cf(n-1)$
  $c \in [0,9;1]$, f fonction du signal et f_a fonction accentuée



https://yannick-1.gitbook.io/reconnaissance-automatique-de-la-parole/chapter1

- s signal, Q longueur d'une trame, K taille d'un échantillon, q l'échantillon, s_K signal extrait, w fonction de fenêtrage
- $s_K(n, q) = s(n)w(q - n)$

Fonctions de fenêtrage



https://fr.wikipedia.org/wiki/Fen

Fenêtre de Hamming :

$$w(n) = 0.54 - 0.46\cos(2\pi\frac{n-1}{K-1})$$

Décomposition en série de Fourier finie

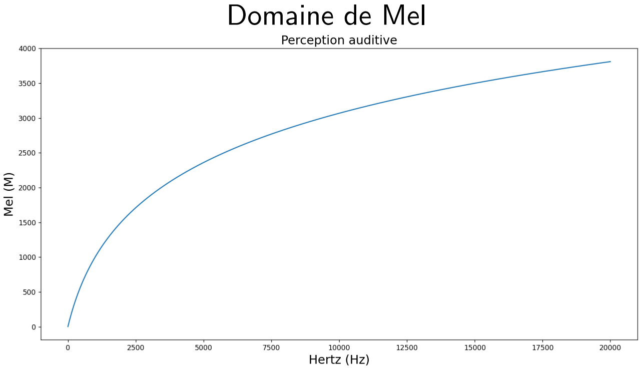
Pour tous les échantillons :

- N taille de la liste, $W = \exp(-\frac{2i\pi}{N})$, $(X_k)_{k\in[\![0;N-1]\!]}$ liste d'entrée et $(A_r)_{r\in[\![0;N-1]\!]}$ liste de sortie
- $\forall r \in [\![0; N-1]\!], A_r = \sum_{k=0}^{N-1} X_k W^{rk}$

Spectrogramme

Pour tous les échantillons :

- $\forall r \in [\![0; N-1]\!], A_r^s = \frac{|A_r|^2}{N}$
- $(A_r^s)_{r \in [\![0;N-1]\!]}$, liste des points du spectrogramme

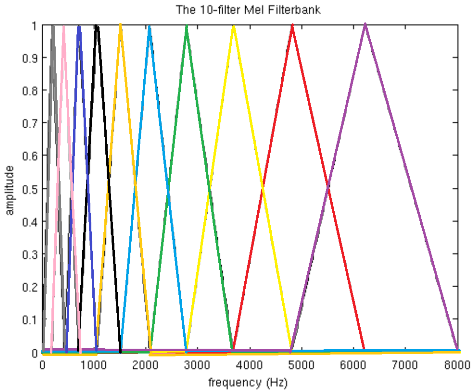


- $F_{mel}(F_{hz}) = 1125 \log(1 + \frac{F_{hz}}{700})$
- $F_{hz}(F_{mel}) = 700(\exp(\frac{F_{mel}}{1125}) - 1)$
- F_{hz} la fréquence en hertz et F_{mel} la fréquence en mel

Filtres

- F_{max} fréquence maximale et F_{min} fréquence minimale N nombre de filtres
- $M_{max} = F_{mel}(F_{max})$ et $M_{min} = F_{mel}(F_{min})$
- $(M_n)_{n \in [\![0;N+1]\!]}$ liste de $N+2$ points équitablement répartis entre M_{max} et M_{min}
- $(H_n)_{n \in [\![0;N+1]\!]}$ liste telle que $\forall n \in [\![0; N+1]\!], H_n = F_{hz}(M_n)$
- $\forall m \in [\![0; N-1]\!], \phi_m(k) =$
$$\begin{cases} 0 & \text{si} \quad k < H_{m-1} \\ \frac{k-H_{m-1}}{H_m - H_{m-1}} & \text{si} \quad H_{m-1} \leq k \leq H_m \\ \frac{H_{m+1}-k}{H_{m+1}-H_m} & \text{si} \quad H_m \leq k \leq H_{m+1} \\ 0 & \text{si} \quad k > H_{m+1} \end{cases}$$

Filtres



http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/eqn2

Passage au logarithme

- N taille de la liste, $(x_n)_{n\in[\![0;N-1]\!]}$ liste d'entrée et $(X_k)_{k\in[\![0;N-1]\!]}$ liste de sortie
- $\forall k \in [\![0; N-1]\!], X_k = \log(x_n)$

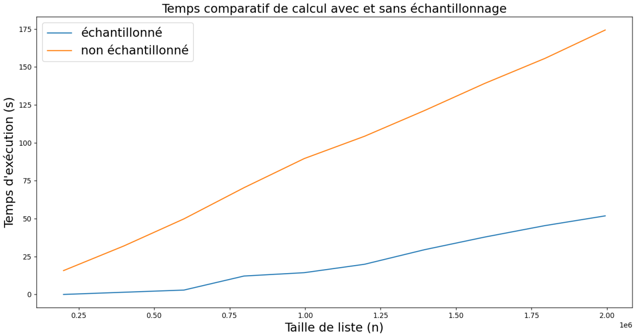
Transformée en cosinus

- N taille de la liste, $(x_n)_{n\in[\![0;N-1]\!]}$ liste d'entrée et $(X_k)_{k\in[\![0;N-1]\!]}$ liste de sortie
- $\forall k \in [\![0; N-1]\!], X_k = \sum_{n=0}^{N-1} x_n \cos(\frac{\pi}{2N}(2k+1)n)$

Temps d'exécution

Temps d'exécution

- Solution :
  - Echantillonnage
  - Période



Décomposition en série de Fourier finie

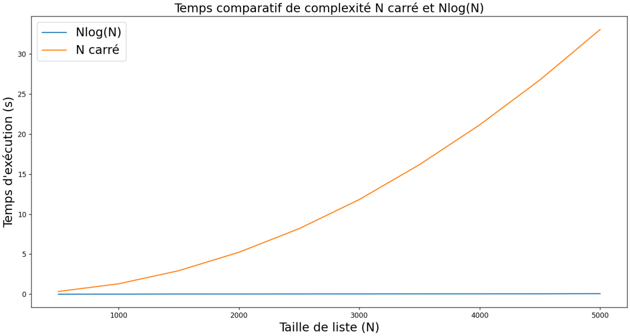
- Complexité $\mathcal{O}(N^2)$

Décomposition en série de Fourier finie

- Complexité $\mathcal{O}(N^2)$
- Solution :
  - Algorithme de Transformée de Fourier Rapide
  - Complexité $\mathcal{O}(N \log(N))$

Décomposition en série de Fourier finie

- Complexité $\mathcal{O}(N^2)$
- Solution :
  - Algorithme de Transformée de Fourier Rapide
  - Complexité $\mathcal{O}(N\log(N))$



Transformée de Fourier Rapide

- N taille de la liste, $W = \exp(-\frac{2i\pi}{N})$, $(X_k)_{k\in[\![0;N-1]\!]}$ liste d'entrée et $(A_r)_{r\in[\![0;N-1]\!]}$ liste de sortie
- $\forall r \in [\![0; N-1]\!], A_r = \sum_{k=0}^{N-1} X_k W^{rk}$
- $N = 2^p, p \in \mathbb{N}$

Transformée de Fourier Rapide

- Soit r $\in [\![0; N-1]\!], A_r = \sum_{k=0}^{N-1} X_k W^{rk}$
- Diviser $p-1$ fois par 2
- Devant chaque terme d'indice impair : $W^{r2^x}, x \in [\![0; p-2]\!]$ avec $W = \exp(-\frac{2i\pi}{N})$
- $\forall r \geq \frac{N}{2} = 2^{p-1}$ on pose $r = 2^{p-1} + j$
- $W^{2^x r} = W^{2^x j}$

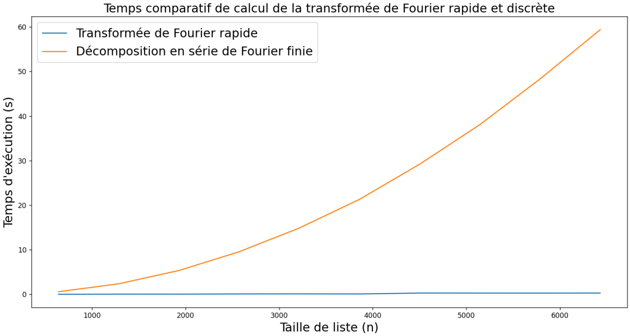
Transformée de Fourier Rapide

Résumé :

- Diviser le problème
- Calcul de la moitié des termes pour chaque sous-problèmes
- Remonter le problème avec les résultats des sous-problèmes précédents

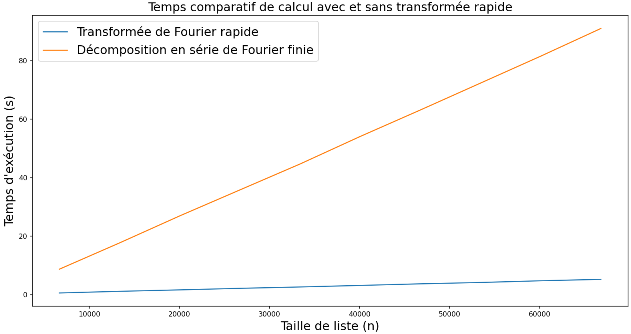
Transformée de Fourier Rapide

Efficacité de l'algorithme :



Transformée de Fourier Rapide

Efficacité du programme complet :

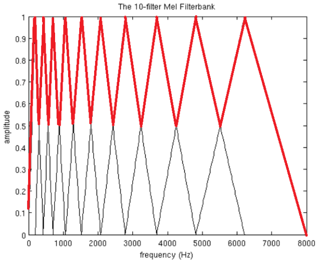


Transformée de Fourier Rapide

Causes de différences :

- Signaux séparés ou non
- Calculs différents

- Création des filtres



- On prend les valeurs maximales pour créer ϕ
- Filtrage : $f_{filtre} = f * \phi$

Transformée en cosinus discrète

- Complexité $\mathcal{O}(N^2)$
- Solution :
  - Adaptation de l'algorithme de Transformée de Fourier Rapide à la Transformée en cosinus
  - Complexité $\mathcal{O}(N\log(N))$

Transformée en cosinus rapide

- N taille de la liste, $(x_n)_{n\in[\![0;N-1]\!]}$ liste d'entrée et $(X_k)_{k\in[\![0;N-1]\!]}$ liste de sortie
- $\forall k \in [\![0; N-1]\!], X_k = \sum_{n=0}^{N-1} x_n \cos(\frac{\pi}{2N}(2k+1)n)$
- $N = 2^p, p \in \mathbb{N}$

Transformée en cosinus rapide

- Même principe que la Transformée de Fourier rapide
- Soit $k \in [\![0; N-1]\!], X_k = \sum_{n=0}^{N-1} x_n \cos(\frac{\pi}{2N}(2k+1)n)$

$$X_k = \sum_{n=0}^{\frac{N}{2}-1} x_{2n} \cos(\frac{\pi(2k+1)2n}{2N}) + \sum_{n=0}^{\frac{N}{2}-1} x_{2n+1} \cos(\frac{\pi(2k+1)(2n+1)}{2N})$$

- $2\cos(\frac{(2k+1)\pi}{2N})\cos(\frac{(2k+1)\pi(2n+1)}{2N}) = \cos(\frac{2n(2k+1)\pi}{2N}) + \cos(\frac{2(2k+1)(n+1)\pi}{2N})$

Transformée en cosinus rapide

D'où :

$$2\cos\left(\frac{\pi(2k+1)}{2N}\right)\sum_{n=0}^{\frac{N}{2}-1} x_{2n+1}\cos\left(\frac{\pi(2k+1)(2n+1)}{2N}\right) =$$

$$\sum_{n=0}^{\frac{N}{2}-1} x_{2n+1}\cos\left(\frac{\pi(2k+1)n}{N}\right) + \sum_{n=0}^{\frac{N}{2}-1} x_{2n+1}\cos\left(\frac{\pi(2k+1)(n+1)}{N}\right)$$

Transformée en cosinus rapide

C'est-à-dire :

$$\sum_{n=0}^{\frac{N}{2}-1} x_{2n+1} \cos\left(\frac{\pi(2k+1)(2n+1)}{2N}\right) =$$

$$\frac{1}{2\cos\left(\frac{\pi(2k+1)}{2N}\right)} \sum_{n=0}^{\frac{N}{2}-1} (x_{2n+1} + x_{2n-1}) \cos\left(\frac{\pi(2k+1)n}{N}\right)$$

Transformée en cosinus discrète

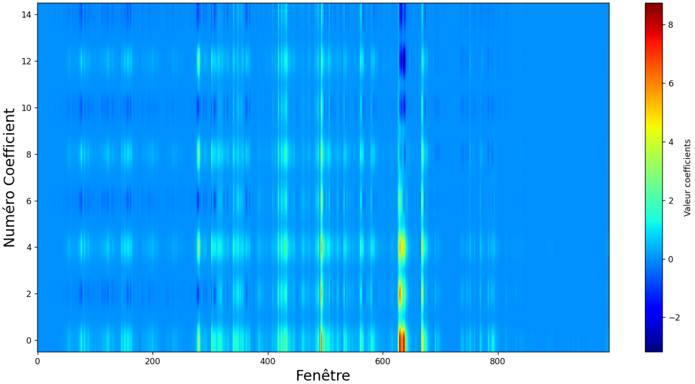
Après calculs :

- $\forall k \in [\![0; \frac{N}{2} - 1]\!], X_k = \sum_{n=0}^{\frac{N}{2}-1} x_{2n} \cos(\frac{\pi(2k+1)n}{N}) + \frac{1}{2\cos(\frac{\pi(2k+1)}{2N})} \sum_{n=0}^{\frac{N}{2}-1} (x_{2n+1} + x_{2n-1}) \cos(\frac{\pi(2k+1)n}{N})$
- $\forall k \in [\![0; \frac{N}{2} - 1]\!], X_{n-1-k} = \sum_{n=0}^{\frac{N}{2}-1} x_{2n} \cos(\frac{\pi(2k+1)n}{N}) - \frac{1}{2\cos(\frac{\pi(2k+1)}{2N})} \sum_{n=0}^{\frac{N}{2}-1} (x_{2n+1} + x_{2n-1}) \cos(\frac{\pi(2k+1)n}{N})$

Transformée en cosinus discrète

Résumé :

- Diviser le problème
- Calcul de la moitié des termes pour chaque sous-problèmes
- Remonter le problème avec les résultats des sous-problèmes précédents



Coefficients pour un enregistrement de Pigeon, avec 5 points par période

Objectifs :

- Calcul des coefficients
- Implémentation dans un réseau de neurones

Complexité algorithme de Transformée de Fourier rapide

On pose $C(N)$ la complexité en temps de calcul de la décomposition en série de Fourier discrète, avec $N = 2^p$ la taille de la liste

Si $N \neq 0$ alors : $C(N) = 2C(\frac{N}{2}) + N = 2C(2^{p-1}) + 2^p$

D'où : $\dfrac{C(2^p)}{2^p} = \dfrac{C(2^{p-1})}{2^{p-1}} + 1 = p + 1$

C'est-à-dire : $C(2^p) = 2^p(p+1)$

Or : $\dfrac{p+1}{\log(2^p)} = \dfrac{1}{\log(2)} + \dfrac{1}{p\log(2)} \underset{p \to +\infty}{\longrightarrow} \dfrac{1}{\log(2)}$

Donc : $C(2^p) = 2^p\mathcal{O}(\log(2^p))$

D'où

$$C(N) = \mathcal{O}(N\log(N))$$

Diviser pour mieux régner

Soit r $\in [\![0; N-1]\!], A_r = \sum_{k=0}^{N-1} X_k W^{rk}$

$$A_r = \sum_{k=0}^{2^{p-1}-1} X_{2k} W^{2rk} + \sum_{k=0}^{2^{p-1}-1} X_{2k+1} W^{r(2k+1)}$$

$$A_r = \sum_{k=0}^{2^{p-1}-1} X_{2k} W^{2rk} + W^r \sum_{k=0}^{2^{p-1}-1} X_{2k+1} W^{2rk}$$

$$A_r = \sum_{k=0}^{2^{p-2}-1} X_{2(2k)} W^{4rk} + \sum_{k=0}^{2^{p-2}-1} X_{2(2k+1)} W^{2r(2k+1)} +$$

$$W^r \left(\sum_{k=0}^{2^{p-2}-1} X_{2(2k)+1} W^{4rk} + \sum_{k=0}^{2^{p-2}-1} X_{2(2k+1)+1} W^{2r(2k+1)} \right)$$

Calcul de la moitié des facteurs

$W^{r2^x} = \exp(-2i\pi\frac{2^x(2^{p-1}+j)}{2^p})$

$W^{r2^x} = \exp(-i\pi 2^x)\exp(-2i\pi\frac{2^x j}{2^p})$

Or $\exp(-i\pi 2^x) = 1$

Donc $W^{2^x r} = W^{2^x j}$

Reconnaissance de chant d'oiseau
Présentation du sujet
Principe
Etude
Plan du calcul
Acquisition et pré-réglages
Séparation
Fourier
Filtrage
Logarithme et cosinus
Application
Problème 1
Problème 2
Filtrage
Problème 3
Coefficients
Conclusion
Annexe
Démonstation
Code

Diviser pour mieux régner

- On pose $K = 2k + 1$
- $2\cos(\frac{K\pi}{2N})\cos(\frac{K\pi(2n+1)}{2N})$
$= 2\cos(\frac{K\pi}{2N})(\cos(\frac{nK\pi}{N})\cos(\frac{K\pi}{2N}) - \sin(\frac{nK\pi}{N})\sin(\frac{K\pi}{2N}))$
$= 2\cos(\frac{K\pi}{2N})^2\cos(\frac{nK\pi}{N}) - 2\cos(\frac{K\pi}{2N})\sin(\frac{nK\pi}{N})\sin(\frac{K\pi}{2N})$
- $\cos(\frac{2nK\pi}{2N}) + \cos(\frac{2K(n+1)\pi}{2N})$
$= \cos(\frac{nK\pi}{N}) + \cos(\frac{2K\pi}{2N})\cos(\frac{nK\pi}{N}) - \sin(\frac{2K\pi}{2N})\sin(\frac{nK\pi}{N})$
$= \cos(\frac{nK\pi}{N}) + (\cos(\frac{K\pi}{2N})^2 - \sin(\frac{K\pi}{2N})^2)\cos(\frac{nK\pi}{N}) -$
$2\cos(\frac{K\pi}{2N})\sin(\frac{nK\pi}{N})\sin(\frac{K\pi}{2N})$
$= 2\cos(\frac{K\pi}{2N})^2\cos(\frac{nK\pi}{N}) - 2\cos(\frac{K\pi}{2N})\sin(\frac{nK\pi}{N})\sin(\frac{K\pi}{2N})$

Reconnaissance de chant d'oiseau

Présentation du sujet
Principe
Etude

Plan du calcul
Acquisition et pré-réglages
Séparation
Fourier
Filtrage
Logarithme et cosinus

Application
Problème 1
Problème 2
Filtrage
Problème 3
Coefficients

Conclusion

Annexe
Démonstation
Code

Addition des deux sommes

- $m = n + 1$:
$$\sum_{n=0}^{\frac{N}{2}-1} x_{2n+1} \cos\left(\frac{\pi(2k+1)(n+1)}{N}\right) = \sum_{m=1}^{\frac{N}{2}} x_{2m-1} \cos\left(\frac{\pi(2k+1)(m)}{N}\right)$$
- $\cos\left(\frac{\pi(2k+1)\frac{N}{2}}{N}\right) = \cos\left(\frac{\pi(2k+1)}{2}\right) = 0$
- $x_{2n-1}\big|_{n=0} = 0$

Calcul de la moitié des facteurs

Rappel : $$\sum_{n=0}^{\frac{N}{2}-1} x_{2n+1}\cos\left(\frac{\pi(2k+1)(2n+1)}{2N}\right) = \frac{1}{2\cos\left(\frac{\pi(2k+1)}{2N}\right)}\sum_{n=0}^{\frac{N}{2}-1}(x_{2n+1}+x_{2n-1})\cos\left(\frac{\pi(2k+1)n}{N}\right)$$

- $\forall k \in [\![0;\frac{N}{2}-1]\!], \cos\left(\frac{\pi(2(N-1-k)+1)}{2N}\right) = \cos\left(\frac{\pi(2N-2k-1)}{2N}\right) = \cos\left(\pi - \frac{\pi(2k+1)}{2N}\right) = -\cos\left(\frac{\pi(2k+1)}{2N}\right)$
- $\forall k \in [\![0;\frac{N}{2}-1]\!], \cos\left(\frac{\pi(2(N-1-k)+1)n}{N}\right) = \cos\left(\frac{\pi(2N-2k-1)n}{N}\right) = \cos\left(2n\pi - \frac{\pi(2k+1)n}{N}\right) = \cos\left(\frac{\pi(2k+1)n}{N}\right)$

```python
### CALCUL COEFFICIENTS ###

# Import des bibliothèques

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

# Ouverture du document texte

f = open("aigle3.txt", 'r')

### Programme Principal

# Convertir en liste

def OuvrEtConv (f) :

    '''

    entrée : str
    sortie : list

    but : prend en entrée la chaîne de caractères du type :
    0.0066
    0.0095
    qui représente les ordonnées de la fonction tracée par le signal et qui renvoie la liste de ces valeurs

    '''
    c = f.read()
    M = []
    i = 0 #itération des caractères
    z = 0 #compteur de valeurs

    for k in range (len(c)) : # on compte ici le nombre de valeurs
        if c[k] == '\n' :
            z += 1

    for j in range (z) : #j est donc l'indice de la valeur

        P = 0 #valeur entière
        D = 0 #valeur décimale
        V = 0 #valeur complète
        a = 0 #indique si la valeur est négative ou non

        if c[i] == '-' : # valeur négative ou non
            a += 1
            i += 1
        N = [] #liste comportant les caractères composant la partie entière
        while c[i] != '.' : #on cherche la partie entière
            N.append(c[i])
            i += 1
```

```python
            i += 1
        for k in range (len(N)) :
            P += float(N[-k])*(10**(k+1))
        i += 1
        L = [] #même chose avec la partie décimale
        while c[i] != '\n' :
            L.append(c[i])
            i += 1
        for k in range (len(L)) :
            D += int(L[k])*(10**(-k-1))
        if a == 1 :
            V = (-1)*(P + D)
        else :
            V = P + D
        i += 1
        if abs(V) >= 0.0001 : #pour prendre que les valeurs utiles
            M.append(V)

    return(M)

# Valeur moyenne

def VaMoy (L) :

    '''

    entrée : list
    sortie : float

    but : renvoie la valeur moyenne de la liste d'entrée

    '''

    a = 0 #sommme des valeurs
    for x in L:
        a+=x

    return a/len(L)

# Période

def Periode (L) :

    '''

    entrée : list
    sortie : list

    but : renvoie la liste des différentes périodes du signal (liste d'entrée)

    '''
```

```python

    n = len(L)
    f1 = 0 #début de la période
    f2 = 0 #fin de la période
    P = [] #liste des périodes
    a = 0 #nombre de fois que la valeur moyenne est vue (pour une période)
    VM = VaMoy(L)

    for i in range(n-1) :
        if (L[i] <= VM and L[i+1] > VM) or (L[i] >= VM and L[i+1] < VM) : #si la fonction passe par sa valeur moyenne
            if a == 0 :
                a += 1
                f1 = i
            elif a == 1 :
                a += 1
            elif a == 2 :
                f2 = i - 1
                a = 0
                P.append(f2-f1)

    return P

# Échantillonnage

def Echan (L,points) :

    '''

    entrée : list
    sortie : list

    but : renvoie la liste échantillonnée, avec ici 'points' points par période, de la liste d'entrée

    '''

    # on essaie avec 'points' point par périodes

    T = VaMoy(Periode(L))#fréquence moyenne
    LE = [] #signal échantillonné
    NbT = int(len(L)/T) #nombre de périodes
    if NbT*points >= len(L) :
        print("échantillonnage impossible, renvoi de la liste de départ non échantillonnée")

        return L

    else :
        for i in range (NbT*points) :
            LE.append(L[int(i*(T//points))])

        return LE

# Normaliser
```

```python
# Normaliser

def SiNor (L) :

    '''

    entrée : list
    sortie : list

    but : renvoie la liste d'entrée décalée verticalement, de telle sorte que sa valeur moyenne soit nulle

    '''
    LN = [] #nouvelle liste
    VM = VaMoy(L)
    for i in range (len(L)) :
        LN.append(L[i] - VM)

    return LN

# Pré-accentuer

def SiAcc (L,LN) : #on accentue certaines fréquences

    '''

    entrée : list,list
    sortie : list

    but : renvoie la liste d'entrée (LN) où les hautes fréquences sont accentuées

    '''

    K = 0.95 #facteur d'accentuation

    LA = [VaMoy(L)] #on prend comme première valeur, la valeur moyenne du signal non normalisé
    for i in range (len(L)-1) :
        LA.append(LN[i+1]-K*LN[i])

    return LA

# Trames

def Trame (L,LF) : #avec L la liste de base et LF la liste échantillonnée

    '''

    entrée : list,list
    sortie : list(list)

    but : renvoie la liste des différents échantillons de la liste d'entrée (LF)

    '''
```

```
    '''

    T = VaMoy(Periode(L)) #période moyenne
    Q = int(220.5/(len(LF)/len(L))) #longueur d'un trame
    q = int(Q*2) #longueur de l'échantillion
    end = int(len(LF)/Q) #nombre total possible de trames
    LT = [] #liste regroupant toutes les trames
    K = 0

    while K < end - 1 : #jusqu'a avoir fait le nombre maximal de trames
        Ltemp = [] #trame
        for i in range (Q*K,Q*K + q) :
            tra = L[i]*FoncFen(K-i,q)
            'note : si problème par la suite, on peut essayer avec (K*Q+q) à la place de K'
            Ltemp.append(tra)
        LT.append(Ltemp)
        K += 1

    return LT

def FoncFen (n,K) :

    '''

    entrée : int,int
    sortie : float

    '''

    return 0.54-0.46*np.cos(2*np.pi*((n-1)/(K-1)))

# Fourier

def Rajoute_0 (L) :

    '''

    entrée : list
    sortie : list

    but : renvoie une liste dont le début est la liste d'entrée et la fin est des zéros, de telle sorte que la taille de la liste soit une puissance de 2

    '''

    x = 0 #test
    i = 0 #itération dans la liste
    n = len(L)

    while x == 0 : # on cherche la puissance de deux supérieure la plus proche
        if n == 2**i :
            x = 1
        elif 2**i < n and 2**(i+1) > n : # si la puissance de deux supérieure la plus proche est 2 puissance i alors :
```

Reconnaissance de chant d'oiseau

Présentation du sujet
Principe
Etude
Plan du calcul
Acquisition et pré-réglages
Séparation
Fourier
Filtrage
Logarithme et cosinus
Application
Problème 1
Problème 2
Filtrage
Problème 3
Coefficients
Conclusion
Annexe
Démonstration
Code

```
            x = 1
        elif 2**i < n and 2**(i+1) > n : # si la puissance de deux supérieure la plus proche est 2 puissance i alors :
            x = 1
        i += 1
    r = (2**i)-n #le nombre de 0 à rajouter
    L_new = []
    for k in L :
        L_new.append(k)
    for k in range (r) :
        L_new.append(0)

    return L_new

def FFT(x):

    '''


    entrée : list
    sortie : list

    but : renvoie la transformée de Fourier de x, version numpy

    '''


    # on se base sur la symétrie de la transformée de Fourier discrète, pour calculer la transformée de Fourier pour une liste de N valeurs, il suffit que de calculer la moité des termes

    N = len(x)

    if N == 1: #on divise jusqu'à ce que les parties soient de taille 1

        return x

    else:

        #la récursivité va ici pemettre de diviser la somme en plusieurs parties puis de remonter l'algorithme
        X_pair = FFT(x[::2])
        X_impair = FFT(x[1::2])
        facteur = np.exp(-2j*np.pi*np.arange(N)/ N)

        X = np.concatenate([X_pair+facteur[:int(N/2)]*X_impair,
            X_pair+facteur[int(N/2):]*X_impair])

        return X

def FFT2(x) :

    '''


    entrée : list
    sortie : list

    but : renvoie la transformée de Fourier de x, version lisible
```

```python

    but : renvoie la transformée de Fourier de x, version lisible

    '''

    N = len(x)

    if N == 1 : #on divise la liste jusqu'à avoir une liste de taille 1
        return x

    else :
        X_pair = FFT2(x[::2])
        X_impair = FFT2(x[1::2])

        facteurs = [] #on créée nos facteurs
        for i in range (N) :
            facteurs.append(np.exp(-2j*np.pi*i/N))

        X1 = []
        X2 = []
        for i in range (len(X_pair)) :
            X1.append(X_pair[i]+facteurs[:int(N/2)][i]*X_impair[i])
            X2.append(X_pair[i]+facteurs[int(N/2):][i]*X_impair[i])

        return X1 + X2

def ToutesTFD (L) :
    '''

    entrée : list(list)
    sortie : list

    but : renvoie la transformée de Fourier discrète appliquée à chaque liste dans la liste d'entrée

    '''

    TFDS = []
    for x in L :
        tf = FFT2(Rajoute_0(x)) #transformée de Fourier appliquée à une liste
        TFDS.append(tf)

    return TFDS

def TFD(x) :
    '''

    entrée : list
    sortie : list

    but : renvoie la transformée de Fourier de x (version lente)
```

```
    but : renvoie la transformée de Fourier de x (version lente)

    '''

    N = len(x)

    Coefs = []
    for r in range (N) :
        Co = 0
        for k in range (N) :
            Co += x[k]*np.exp(-2j*np.pi*r*k/N)
        Coefs.append(Co)

    return Coefs

def ToutesTFDlentes (L) :

    '''

    entrée : list(list)
    sortie : list

    but : renvoie la transformée de Fourier discrète appliquée à chaque liste dans la liste d'entrée

    '''

    TFDS = []
    for x in L :
        tf = TFD(x) #transformée de Fourier appliquée à une liste
        TFDS.append(tf)

    return TFDS

# Périodogramme

def Per(L) :

    '''

    entrée : list(list)
    sortie : list(list)

    but : renvoie le périodogramme de chaque sous-liste de la liste d'entrée

    '''

    # On calcule le périodogramme pour toutes les sous listes, c'est-à-dire qu'on calcule le module de chaque valeur au carré, et on divise cette valeur par la taille de la sous-liste associée

    Pe = [] #liste des périodogrammes

    for x in L :
        PP = []
```

```python
    for x in L :
        PP = []
        for i in range (len(x)) :
            p = (1/len(x))*((np.abs(x[i]))**2)
            PP.append(p)
        Pe.append(PP)

    return Pe

# Mel filterbank

def Mel(f) :

    '''

    entrée : float
    sortie : float

    but : revoie dans le domaine de fréquences Mel la fréquence f

    '''

    return 1125*np.log(1+(f/700))

def Melinv(m) :

    '''

    entrée : float
    sortie : float

    but : revoie dans le domaine fréquenciel classique la fréquence f (Mel)

    '''

    return 700*(np.exp(m/1125)-1)

def Melfilterbank(L,n,sr) :

    '''

    entrée : list,int,float
    sortie : list(list)

    but : renvoie la liste des filtres du melfilterbank

    '''

    #n le nombre de filtres

    Fmin = 0
    Fmax = len(L)
```

```python
    Fmax = len(L)
    Mmin = Mel(Fmin)
    Mmax = Mel(Fmax)

    M = [] #liste de points entre Mmin et Mmax
    d = Mmax - Mmin
    for i in range (n+2) :
        M.append(Mmin + (i)*(d/(n+1)) )

    H = [] #même chose convertie en Hertz
    for x in M :
        H.append(Melinv(x))

    filterbank = []
    for m in range (n) :
        Temporary = [] #liste temporaire des points d'un filtre
        for k in range (int(H[-1])) : #création des filtres triangulaires
            if k < H[m] :
                Temporary.append(0)
            elif k >= H[m] and k < H[m+1] :
                Temporary.append((k-H[m])/(H[m+1]-H[m]))
            elif k == H[m+1] :
                Temporary.append(1)
            elif k > H[m+1] and k <= H[m+2] :
                Temporary.append((H[m+2]-k)/(H[m+2]-H[m+1]))
            elif k > H[m+2] :
                Temporary.append(0)
        filterbank.append(Temporary)

    return filterbank

def ValeursMaxListes(L) :
    '''

    entrée : list(list)
    sortie : list

    but : renvoie la liste constituée des valeurs maximales entre chaque liste

    '''

    #on vérifie que les listes de sont de la même taille
    liste_tailles = []
    for i in range (len(L)) :
        liste_tailles.append(len(L[i]))
    for x in liste_tailles :
        if liste_tailles[0] != x : #dès qu'une taille de liste est différente de la première, on revoie rien, en disant que les listes ne sont pas de la même taille
            print("les listes ne sont pas de la même taille")
            return None

    liste_valeursmax = []
```

```
    liste_valeursmax = []
    for k in range (len(L[0])) :
        ValeursIndexK = [] #on créé la liste des valeurs d'indice k
        for i in range (len(L)) :
            ValeursIndexK.append(L[i][k])
        liste_valeursmax.append(max(ValeursIndexK)) #on ajoute à la liste finale le maximum des valeurs d'indice k

    return liste_valeursmax

def Melfiltrage(L,n,sr) :

    '''

    entrée : list(list),int,float
    sortie : list(list)

    but : renvoie la liste des sous-listes de la liste d'entrée filtrées par le melfilterbank

    '''

    #on créée nos filtres

    Filtres = []
    for i in range (len(L)) :
        Filtres.append(Melfilterbank(L[i],n,sr))

    #on prend les valeurs maximales des filtres pour chaque trame, pour pouvoir filtrer chaque trame

    Filtres_max = []
    for i in range (len(Filtres)) :
        Filtres_max.append(ValeursMaxListes(Filtres[i]))

    #on filtre, c'est-à-dire qu'on multiplie chaque fréquence de chaque trame par la valeur correspondante du filtre associé
(valeur entre 0 et 1)

    Listes_filtered = []
    for i in range (len(L)) :
        Liste_filtered_i = []
        for j in range (len(L[i])-1) :
            Liste_filtered_i.append(L[i][j]*Filtres_max[i][j])
        Listes_filtered.append(Liste_filtered_i)

    return Listes_filtered

# Logarithme

def Amplification(L) :

    '''

    entrée : list(list),int
```

```python
    entrée : list(list),int
    sortie : list(list)

    but : renvoie la liste des sous-listes de la liste d'entrée amplifiées

    '''

    #le signal ayant des valeurs trop petites, le logarithme renvoie une valeur négative, on amplifie donc le signal

    Liste_min = []
    for x in L :
        Liste_min.append(min(x))

    if min(Liste_min) >= 1 :

        return L

    else :
        Amp = 1 - min(Liste_min)

        L_ampli = [] #on les amplifie
        for i in range (len(L)) :
            L_ampli_i = []
            for j in range (len(L[i])) :
                L_ampli_i.append(Amp + L[i][j])
            L_ampli.append(L_ampli_i)

        return L_ampli

def PassageLog (L) :
    '''

    entrée : list(list)
    sortie : list(list)

    but : renvoie la liste des sous-listes de la liste d'entrée passées au logarithme

    '''

    #on passe au logarithme une liste de liste

    L_log = []
    for i in range (len(L)) :
        L_temp_log = []
        for j in range (len(L[i])) :
            if L[i][j] != 0 :
                L_temp_log.append(np.log(L[i][j]))
        L_log.append(L_temp_log)

    return L_log

# Transformée en cosinus discrète
```

```python

# Transformée en cosinus discrète

def DCT(x) :

    '''

    entrée : list
    sortie : list

    but : renvoie la transformée en cosinus discrète de x (version lente)

    '''

    N = len(x) #taille de la liste

    Coefs = []
    for r in range (N) :
        Co = 0
        for k in range (N) :
            Co += x[k]*np.cos(np.pi*r*(k+1/2)/N)
        Coefs.append(Co)

    return Coefs

def FDCT(x) :

    '''

    entrée : list
    sortie : list

    but : renvoie la transformée en cosinus discrète de x, version rapide

    '''

    N = len(x) #taille

    if N == 1 :
        return x

    else :

        #on divise la liste
        X1 = []
        X2 = []
        for i in range (int(N/2)) :

            X1.append(x[i]+x[-i-1])
            X2.append((x[i]-x[-i-1])/(np.cos((i+0.5)*np.pi/N)*2))

        X1 = FDCT(X1)
        X2 = FDCT(X2)
```

```
        X1 = FDCT(X1)
        X2 = FDCT(X2)

        X = []
        for i in range (int(N/2)-1) :
            X.append(X1[i])
            X.append(X2[i] + X2[i+1])
        X.append(X1[-1])
        X.append(X2[-1])

        return X


def ToutesDCT(L) :

    '''

    entrée : list(list)
    sortie : list(list)

    but : renvoie la liste des transformées en cosinus discrètes des sous-listes de la liste d'entrée (rapide)

    '''

    L_nouveau = []

    for x in L :
        dct = FDCT(Rajoute_0(x))
        L_nouveau.append(dct)

    return L_nouveau

def ToutesDCTlentes(L) :

    '''

    entrée : list(list)
    sortie : list(list)

    but : renvoie la liste des transformées en cosinus discrètes des sous-listes de la liste d'entrée (lente)

    '''

    L_nouveau = []

    for x in L :
        dct = DCT(Rajoute_0(x))
        L_nouveau.append(dct)

    return L_nouveau

# Coefficients
```

5 - Annexe
Code


```python
# Coefficients

def MFCC(L,c) :

    '''

    entrée : list(list)
    sortie : list(list)

    but : renvoie et affiche les c premiers mfcc en foncton du temps

    '''

    Liste_coefs = [] #on créée la liste des coefficients en fonction du temps
    for i in range (c) :
        Liste_coef_c = []
        for j in range (len(L)) :
            Liste_coef_c.append(L[j][i])
        Liste_coefs.append(Liste_coef_c)

    plt.imshow(Liste_coefs,origin = 'lower',cmap='jet',aspect="auto")
    plt.colorbar(label="Valeur coefficients")
    plt.gcf().set_size_inches(plt.gcf().get_size_inches()*1.5)
    plt.xlabel("Fenêtre",fontsize='xx-large')
    plt.ylabel("Numéro Coefficient",fontsize='xx-large')
    plt.show()

    return Liste_coefs

# Exécution

def Exe(f,c,e) :

    '''

    entrée : str,int,int
    sortie : list(list))

    but : execution du programme, f étant l'entrée, c le nombre de coefficients et e le nombre de points par période pour l'échantillonnage

    '''

    L = OuvrEtConv(f)
    if L != [] :
        LE = Echan(L,e)
        print("signal échantilloné à ", len(LE),"éléments au lieu de ", len(L))
        LN = SiNor(LE)
        print("signal normalisé")
        LA = SiAcc(LE,LN)
        print("signal pré accentué")
        LT = Trame(L,LA)
        print("signal séparé")
```

```python
        LT = Trame(L,LA)
        print("signal séparé")
        TFDS = ToutesTFD(LT)
        print("transformés calculées")
        Pe = Per(TFDS)
        print("périodogramme calculé")
        M = Melfiltrage(Pe,10,44150)
        print("fréquences filtrées")
        Log = PassageLog(Amplification(M))
        print("signal passé au logarithme")
        dct = ToutesDCT(Log)
        print("transformé en cos")
        mfcc = MFCC(dct,c)
        print("coefficients calculés")

    return mfcc

### Programme Annexe

# Test log

def test_log(L) :

    '''

    entrée : list(list)
    sortie : int,int

    but : renvoie le nombre de valeurs interdites pour le logarithme

    '''

    zeros = 0
    negatifs = 0

    for sous_liste in L :
        for x in sous_liste :
            if x == 0 :
                zeros += 1
            if x < 0 :
                negatifs += 1

    return zeros,negatifs

# Temps comparatif

def ListeMoins (L,d) :

    '''

    entrée : list,int
    sortie : list
```

```
    sortie : list

    but : renvoie la liste L avec les d derniers termes enlevés

    '''

    if d < len(L) :
        return L[0:len(L)-d]
    else :
        return L

def ExeC (E,L) :

    '''

    entrée : bool,list

    but : execution du programme avec ou sans fft

    '''

    if E == True : #execution avec fft
        if L!= [] :
            LN = SiNor(L)
            LA = SiAcc(L,LN)
            LT = Trame(L,LA)
            TFDS = ToutesTFD(LT)
            Pe = Per(TFDS)
            M = Melfiltrage(Pe,10,44150)
            Log = PassageLog(M)
            dct = ToutesDCT(Log)


    else : #execution sans fft
        if L!= [] :
            LN = SiNor(L)
            LA = SiAcc(L,LN)
            LT = Trame(L,LA)
            TFDS = ToutesTFDlentes(LT)
            Pe = Per(TFDS)
            M = Melfiltrage(Pe,10,44150)
            Log = PassageLog(M)
            dct = ToutesDCT(Log)

def ExeE (E,L) :

    '''

    entrée : bool,list

    but : execution du programme avec ou sans échantillonnage

    '''
```

Reconnaissance de chant d'oiseau
Présentation du sujet
Principe
Etude
Plan du calcul
Acquisition et pré-réglages
Séparation
Fourier
Filtrage
Logarithme et cosinus
Application
Problème 1
Problème 2
Filtrage
Problème 3
Coefficients
Conclusion
Annexe
Démonstration
Code

```python
    '''

    if E == True : #execution avec échantionnage
        if L!= [] :
            LE = Echan(L,4)
            LN = SiNor(LE)
            LA = SiAcc(LE,LN)
            LT = Trame(L,LA)
            TFDS = ToutesTFD(LT)
            Pe = Per(TFDS)
            M = Melfiltrage(Pe,10,44150)
            Log = PassageLog(M)
            dct = ToutesDCT(Log)


    else : #execution sans échantillonage
        if L!= [] :
            LN = SiNor(L)
            LA = SiAcc(L,LN)
            LT = Trame(L,LA)
            TFDS = ToutesTFD(LT)
            Pe = Per(TFDS)
            M = Melfiltrage(Pe,10,44150)
            Log = PassageLog(M)
            dct = ToutesDCT(Log)

def ExeTfft(E,L) :
    '''

    entrée : bool,list

    but : exécution de la tfd ou de la fft

    '''

    if E == False : #tfd
        if L!=[] :
            LN = SiNor(L)
            LA = SiAcc(L,LN)
            fft = TFD(LA)
    else : #fft
        if L!=[] :
            LN = SiNor(L)
            LA = SiAcc(L,LN)
            fft = FFT2(Rajoute_0(LA))

def Comparaisonfft_sanstrames (f,n) :
    '''
```

Reconnaissance de chant d'oiseau

Présentation du sujet
- Principe
- Etude

Plan du calcul
- Acquisition et pré-réglages
- Séparation
- Fourier
- Filtrage
- Logarithme et cosinus

Application
- Problème 1
- Problème 2
- Filtrage
- Problème 3
- Coefficients

Conclusion

Annexe
- Démonstration
- Code


```
    entrée : str,int

    but : créer un graphe comparatif des méthodes fft et tfd

    '''

    LtempsE = [] #liste des temps avec fft
    LtempsS = [] #liste des temps sans fft
    Ltaille = [] #liste des tailles de liste
    L = OuvrEtConv(f)

    for i in range (n) :
        d = int(len(L)/n)*i #distance à retirer chaque fois
        LMod = ListeMoins(L,d) #liste avec cette distance en moins

        Ltaille.append(len(LMod)) #on ajoute à la liste des abscisses la taille de la liste

        s1 = time.time() #temps d'execution avec échantillion
        ExeTfft(True,LMod)
        LtempsE.append(time.time() - s1)
        print("Bon pour E", i+1, "fois")

        s2 = time.time() #temps d'execution sans échantillion
        ExeTfft(False,LMod)
        LtempsS.append(time.time() - s2)
        print("Bon pour S", i+1, "fois")

    plt.plot(Ltaille,LtempsE)
    plt.plot(Ltaille,LtempsS)
    plt.legend(['Transformée de Fourier rapide','Décomposition en série de Fourier finie'],fontsize='xx-large')
    plt.title("Temps comparatif de calcul de la transformée de Fourier rapide et discrète",fontsize='xx-large')
    plt.xlabel("Taille de liste (n)",fontsize='xx-large')
    plt.ylabel("Temps d'exécution (s)",fontsize='xx-large')
    plt.show()

def Comparaisonfft (f,n) :
    '''

    entrée : str,int

    but : créer un graphe comparatif des méthodes fft et tfd

    '''

    LtempsE = [] #liste des temps avec fft
    LtempsS = [] #liste des temps sans fft
    Ltaille = [] #liste des tailles de liste
    L = OuvrEtConv(f)

    for i in range (n) :
        d = int(len(L)/n)*i #distance à retirer chaque fois
```

```
    for i in range (n) :
        d = int(len(L)/n)*i #distance à retirer chaque fois
        LMod = ListeMoins(L,d) #liste avec cette distance en moins

        Ltaille.append(len(LMod)) #on ajoute à la liste des abscisses la taille de la liste

        s1 = time.time() #temps d'execution avec échantillion
        ExeC(True,LMod)
        LtempsE.append(time.time() - s1)
        print("Bon pour E", i+1, "fois")

        s2 = time.time() #temps d'execution sans échantillion
        ExeC(False,LMod)
        LtempsS.append(time.time() - s2)
        print("Bon pour S", i+1, "fois")

    plt.plot(Ltaille,LtempsE)
    plt.plot(Ltaille,LtempsS)
    plt.legend(['Transformée de Fourier rapide','Décomposition en série de Fourier finie'],fontsize='xx-large')
    plt.title("Temps comparatif de calcul avec et sans transformée rapide",fontsize='xx-large')
    plt.xlabel("Taille de liste (n)",fontsize='xx-large')
    plt.ylabel("Temps d'exécution (s)",fontsize='xx-large')
    plt.show()

def ComparaisonE (f,n) :
    '''

    entrée : str,int

    but : créer un graphe comparatif des méthodes avec échantillonnage et sans

    '''

    LtempsE = [] #liste des temps avec échantillonnage
    LtempsS = [] #liste des temps sans échantillonnage
    Ltaille = [] #liste des tailles de liste
    L = OuvrEtConv(f)

    for i in range (n) :
        d = int(len(L)/n)*i #distance à retirer chaque fois
        LMod = ListeMoins(L,d) #liste avec cette distance en moins

        Ltaille.append(len(LMod)) #on ajoute à la liste des abscisses la taille de la liste

        s1 = time.time() #temps d'execution avec échantillion
        ExeE(True,LMod)
        LtempsE.append(time.time() - s1)
        print("Bon pour E", i+1, "fois")

        s2 = time.time() #temps d'execution sans échantillion
        ExeE(False,LMod)
```

Reconnaissance de chant d'oiseau

Présentation du sujet
Principe
Etude

Plan du calcul
Acquisition et pré-réglages
Séparation
Fourier
Filtrage
Logarithme et cosinus

Application
Problème 1
Problème 2
Filtrage
Problème 3
Coefficients

Conclusion

Annexe
Démonstration
Code

```python
        ExeE(False,LMod)
        LtempsS.append(time.time() - s2)
        print("Bon pour S", i+1, "fois")

    plt.plot(Ltaille,LtempsE)
    plt.plot(Ltaille,LtempsS)
    plt.legend(['échantillonné','non échantillonné'],fontsize='xx-large')
    plt.title("Temps comparatif de calcul avec et sans échantillonnage",fontsize='xx-large')
    plt.xlabel("Taille de liste (n)",fontsize='xx-large')
    plt.ylabel("Temps d'exécution (s)",fontsize='xx-large')
    plt.show()

# Extraction

def Extrac(L) :
    '''

    entrée : list(list)

    but : créer un fichier texte avec les coefficients de fréquence mel

    '''

    Fichier = open("Pigeon3.txt",'a')
    for x in L :
        for k in x :
            Fichier.write(str(k))
            Fichier.write('\n')
        Fichier.write('\n')
    Fichier.close()

# Complexité

def N_carre(n) :
    '''

    entrée : int

    but : simule une fonction de complexité n carré, analogue à la décomposition en série de Fourier finie

    '''

    Simu = []
    for i in range (n) :
        Valeur = 0
        for k in range (n) :
            Valeur += k*np.exp(-2j*np.pi*k*i)
        Simu.append(Valeur)

def N_logN(n) :
```

5 - Annexe
Code


```python
def N_logN(n) :

    '''

    entrée : int

    but : simule une fonction de complexité n*log(n), analogue à la transformée de Fourier rapide

    '''

    Simu = []
    for i in range (n) :
        Valeur = 0
        for k in range (int(np.log(n))) :
            Valeur += k*np.exp(-2j*np.pi*k*i)
        Simu.append(Valeur)

def Liste_entiers(n) :

    '''

    entrée : int
    sortie : list

    but : renvoie la liste des entiers de 1 à n

    '''

    Liste = []
    for i in range (n) :
        Liste.append(i+1)

    return Liste

def ComparaisonComp(N,n) :

    '''

    entrée : int,int

    but : affiche un graphique comparatif entre les temps d'exécution de fonctions ayant pour complexité N² et Nlog(N)

    '''

    L = Liste_entiers(N)
    Ltaille = []
    LtempsC = []
    LtempsL = []

    for i in range (n) :
        d = int(len(L)/n)*i #distance à retirer chaque fois
        LMod = ListeMoins(L,d) #liste avec cette distance en moins
```

5 - Annexe
Code


```python
    LtempsC = []
    LtempsL = []

    for i in range (n) :
        d = int(len(L)/n)*i #distance à retirer chaque fois
        LMod = ListeMoins(L,d) #liste avec cette distance en moins

        Ltaille.append(len(LMod)) #on ajoute à la liste des abscisses la taille de la liste

        s1 = time.time() #temps d'execution de complexité en carré
        N_carre(len(LMod))
        LtempsC.append(time.time() - s1)
        print("Bon pour C", i+1, "fois")

        s2 = time.time() #temps d'execution de complexité en nlog(n)
        N_logN(len(LMod))
        LtempsL.append(time.time() - s2)
        print("Bon pour L", i+1, "fois")


    plt.plot(Ltaille,LtempsL)
    plt.plot(Ltaille,LtempsC)
    plt.legend(['Nlog(N)','N carré'],fontsize='xx-large')
    plt.title("Temps comparatif de complexité N carré et Nlog(N)",fontsize='xx-large')
    plt.xlabel("Taille de liste (N)",fontsize='xx-large')
    plt.ylabel("Temps d'exécution (s)",fontsize='xx-large')
    plt.show()

# Echelle de Mel

def Echelle_Mel(Hz) :

    '''

    entrée : int

    but : affiche le graphe de la perception auditive

    '''

    Liste_Mel = []
    Liste_Hz = Liste_entiers(int(Hz))

    for x in Liste_Hz :
        Liste_Mel.append(Mel(x))

    plt.plot(Liste_Hz,Liste_Mel)
    plt.title("Perception auditive",fontsize='xx-large')
    plt.xlabel("Hertz (Hz)",fontsize='xx-large')
    plt.ylabel("Mel (M)",fontsize='xx-large')
    plt.show()
```