Université Paul Sabatier
Toulouse

Compte Rendu

Introduction à l'apprentissage automatique

M1 EEA

- Module KEAX7IK1 : Introduction à l'apprentissage automatique
- Réalisé par :
  - KABOU Abdeldjalil

2024 / 2025

TP1 : Analyse des signaux et prétraitement des paramètres

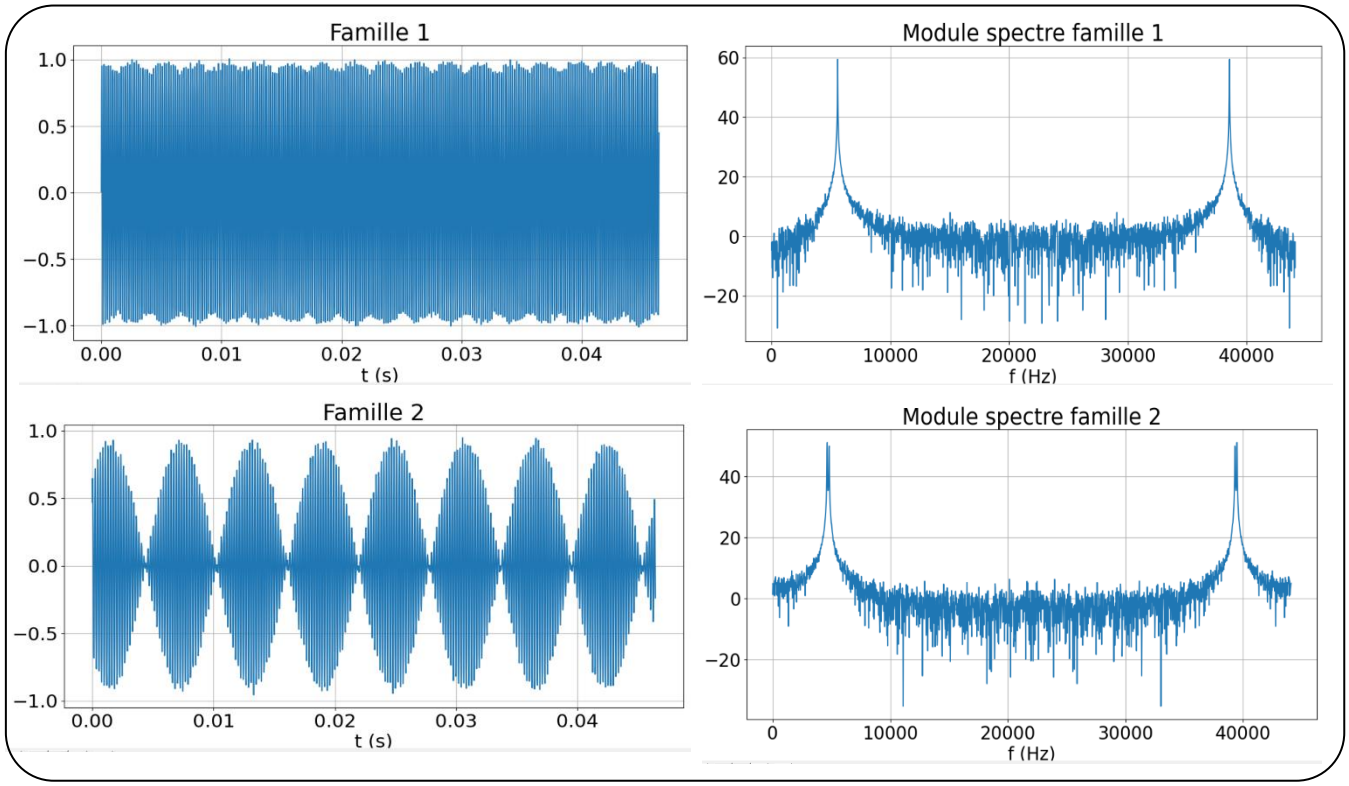
I. *Présentation de la manipulation*

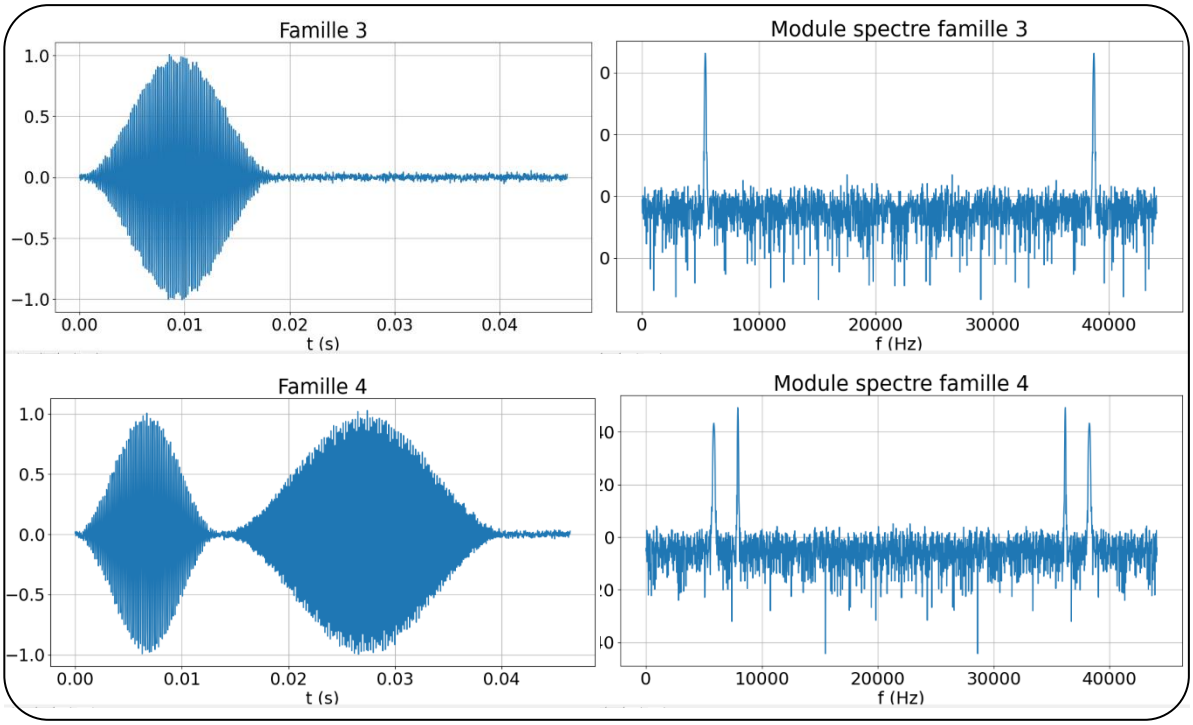
Nous développons un projet de traitement de signaux avec Python. Le programme générateur produit des signaux stockés dans deux fichiers : *signaux.txt* et *signauxTest.txt.* Ces signaux appartiennent à quatre familles distinctes, dont les enregistrements sont regroupés dans *signaux.txt*. La fréquence d'échantillonnage utilisée est de *44100 Hz*. Un programme de lecture et de traitement, *TPClassif2024.py*, est fourni pour analyser ces signaux.

II. *Travail à effectuer*

1. *Analyse du signal :*

L'objectif est d'identifier des caractéristiques distinctives permettant de différencier les familles de signaux. Pour cela, une analyse sera réalisée d'abord dans le domaine temporel, puis dans le domaine fréquentiel. Les signaux, répartis en quatre familles, sont générés par le programme *genere.py* et enregistrés dans le fichier *signaux.txt,* comme illustré dans les figures ci-dessous.





L'enveloppe d'un signal est une courbe lisse qui suit les variations d'amplitude du signal dans le domaine temporel. Elle est utile pour analyser les propriétés d'un signal.

La transformée de Hilbert est un outil mathématique qui permet d'obtenir une représentation analytique d'un signal réel. Le module de cette représentation analytique correspond à l'enveloppe.

Donc on va calcule la transformée de Hilbert des signaux. ET le module de cette transformée donne directement l'enveloppe. On le fait avec notre code qui est dans la figure ci-dessous.

a. Etude temporel

```python
import numpy as np
from scipy.signal import hilbert
import TPClassif2024 as tpc
import matplotlib.pyplot as plt

print("Lecture de la base d'apprentissage.\n")
Enveloppe = np.abs(hilbert(tpc.MatriceDonnees))
# Visualiser l'enveloppe temporelle pour quelques signaux de chaque classe
for classe in np.unique(tpc.NoClasse):
    indices_classe = np.where(tpc.NoClasse == classe)[0]
    for i in indices_classe[:3]:  # Choisissez quelques signaux à visualiser
        plt.plot(Enveloppe[i], label=f'Classe {classe}, Signal {i+1}')

    plt.xlabel('Temps')
    plt.ylabel("Amplitude de l'enveloppe")
    plt.title(f"Enveloppe Temporelle des signaux par classe - famille {classe}")
    plt.show()
```

**Le code est ci-dessous dans les annexes*

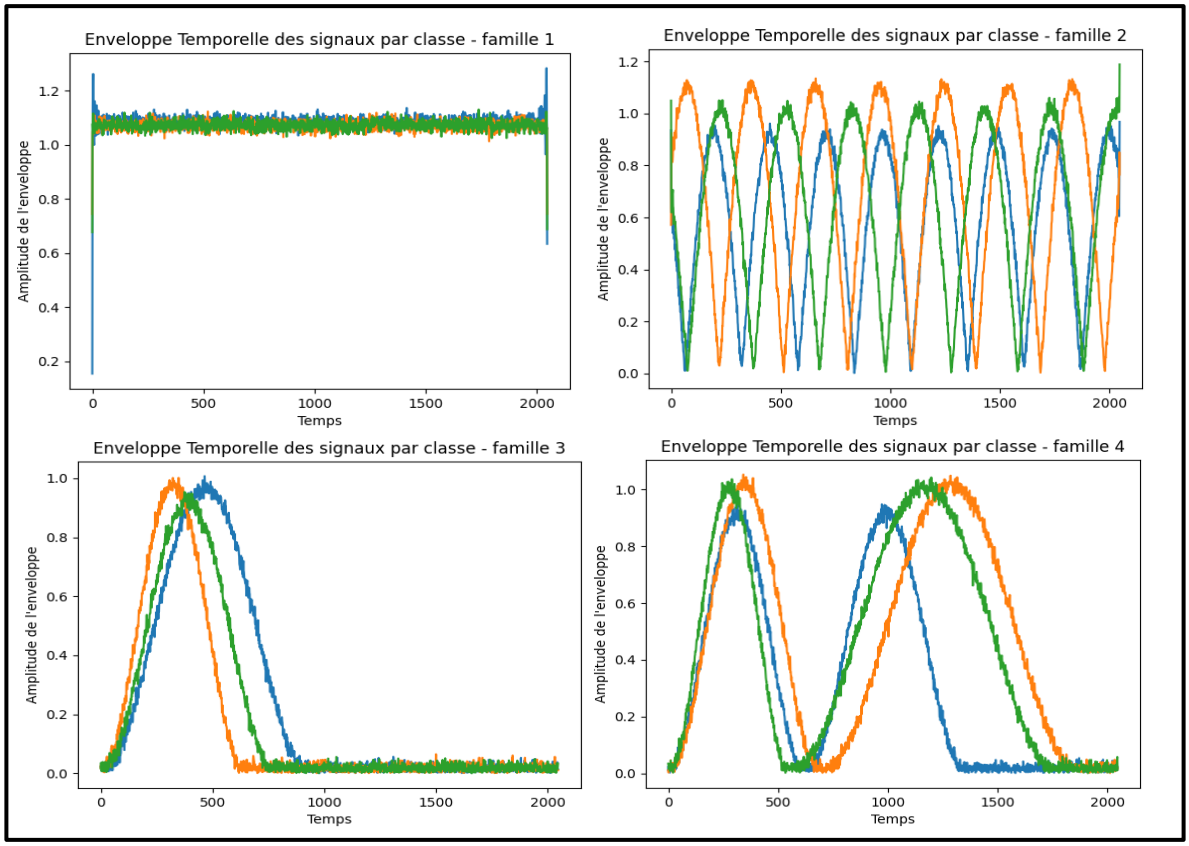
```
Lecture de la base d'apprentissage.

Nombre d'individus de la base d'apprentissage : 1400
Nombre de variables de la base d'apprentissage : 2048


Lecture de la base de test.

Nombre d'individus de la base de test : 1200
Nombre de variables de la base de test : 2048
```

Pour : *Lecture de la base d'apprentissage.*

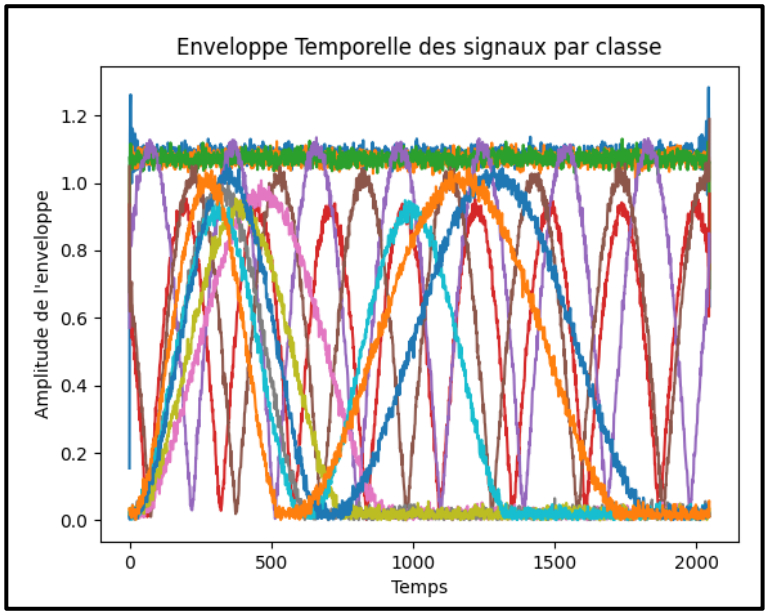
- Nombre d'individus de la base d'apprentissage : 1400
- Nombre de variables de la base d'apprentissage : 2048

Pour : *Lecture de la base de test.*

- Nombre d'individus de la base de test : 1200
- Nombre de variables de la base de test : 2048



Toutes les figures dans une seule fenêtre :

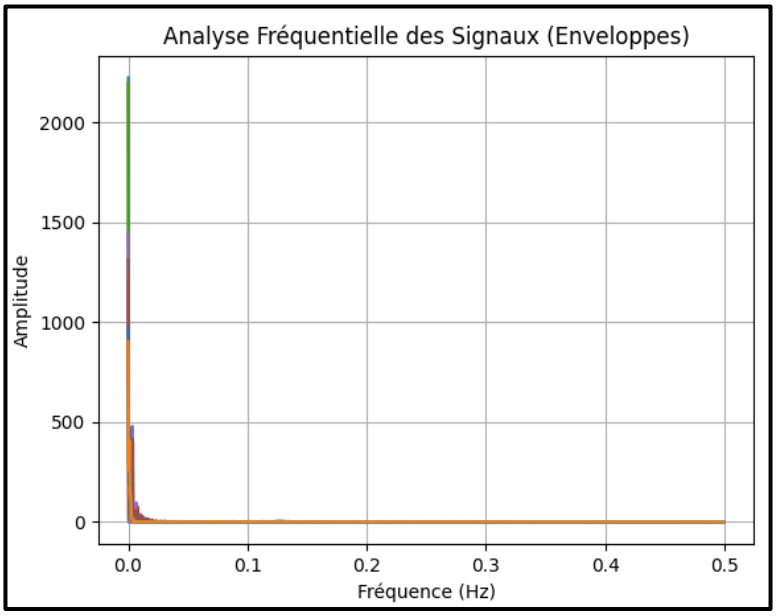


b. *Etude fréquentielle*

```python
import numpy as np
from scipy.signal import hilbert
import TPClassif2024 as tpc
import matplotlib.pyplot as plt
# Lecture de la base d'apprentissage
print("Lecture de la base d'apprentissage.\n")
Enveloppe = np.abs(hilbert(tpc.MatriceDonnees))  # Calcul de l'enveloppe des signaux
# Paramètres de la FFT
N = len(tpc.MatriceDonnees[0])  # Taille du signal
frequencies = np.fft.fftfreq(N)  # Fréquences associées à la FFT
# Afficher l'analyse fréquentielle pour quelques signaux
for classe in np.unique(tpc.NoClasse):
    indices_classe = np.where(tpc.NoClasse == classe)[0]
    for i in indices_classe[:3]:  # Choisir quelques signaux
        # Appliquer la FFT sur l'enveloppe et obtenir le spectre complet
        fft_signal = np.fft.fft(Enveloppe[i])
        plt.plot(frequencies, np.abs(fft_signal), label=f'Classe {classe}, Signal {i+1}')
# Paramétrer le graphique
plt.xlabel('Fréquence (Hz)')
plt.ylabel('Amplitude')
plt.title("Analyse Fréquentielle des Signaux (Enveloppes)")
plt.grid(True)
plt.show()
```

**Le code est ci-dessous dans les annexes*

Toutes les figures dans une seule fenêtre :



2. *Extraction de paramètres pertinents :*

On a utilisé la transformée de Hilbert pour extraire l'enveloppe des signaux. Maintenant on va calculer trois paramètres pour chaque signal : *l'énergie, la moyenne et la variance de l'enveloppe.*

Explication des paramètres :

- *Énergie du signal :* l'énergie est calculée en prenant la somme des carrés des valeurs de l'enveloppe.
- *Moyenne de l'enveloppe* : Il s'agit de la moyenne des valeurs de l'enveloppe.
- *Variance de l'enveloppe* : Cela donne une mesure de la dispersion des valeurs de l'enveloppe autour de la moyenne.

Pour chaque signal de la base de données, on obtient les trois paramètres, voici des exemples dans la figure ci-dessous :

```
Signal 1:
  Energie: 2413.299156304717
  Moyenne de l'enveloppe: 1.0851176410047678
  Variance de l'enveloppe: 0.0008884338446606122

Signal 2:
  Energie: 2348.7543934609957
  Moyenne de l'enveloppe: 1.0707627370065196
  Variance de l'enveloppe: 0.000319892220434154

Signal 3:
  Energie: 2353.493667406524
  Moyenne de l'enveloppe: 1.0718206615149568
  Variance de l'enveloppe: 0.0003672993379821686

Signal 4:
  Energie: 1867.7292456806153
  Moyenne de l'enveloppe: 0.9548122000981969
  Variance de l'enveloppe: 0.0003108332861289627

Signal 5:
  Energie: 1681.770977556855
  Moyenne de l'enveloppe: 0.9059536048272179
  Variance de l'enveloppe: 0.00042530103575238917
```

```
Signal 1229:
  Energie: 436.0731058990918
  Moyenne de l'enveloppe: 0.30530835455961103
  Variance de l'enveloppe: 0.11971312987589378

Signal 1230:
  Energie: 706.467295405381
  Moyenne de l'enveloppe: 0.4480682189443186
  Variance de l'enveloppe: 0.1441896052567249

Signal 1231:
  Energie: 740.4529550117092
  Moyenne de l'enveloppe: 0.44917371181154364
  Variance de l'enveloppe: 0.15979227105675137

Signal 1232:
  Energie: 552.6611491240235
  Moyenne de l'enveloppe: 0.345237387084685
  Variance de l'enveloppe: 0.15066522327965398

Signal 1233:
  Energie: 522.7745162520606
  Moyenne de l'enveloppe: 0.3723116934195586
  Variance de l'enveloppe: 0.11664499720676202
```

**Le code est ci-dessous dans les annexes*

Pour appliquer les paramètres à chaque classe, On peut regrouper les résultats par classe et calculer les paramètres (l'énergie, la moyenne de l'enveloppe et la variance de l'enveloppe) par classe.

Les signaux sont regroupés selon leurs classes, voici dans la figure ci-dessous :

```
Classe 1 :
Nombre de signaux : 350
  Moyenne des energies : 2046.16
  Moyenne des moyennes d'enveloppe : 1.00
  Moyenne des variances d'enveloppe : 0.00
  Energie totale : 716156.96

Classe 2 :
Nombre de signaux : 300
  Moyenne des energies : 1031.55
  Moyenne des moyennes d'enveloppe : 0.64
  Moyenne des variances d'enveloppe : 0.09
  Energie totale : 309464.47
```

```
Classe 3 :
Nombre de signaux : 400
  Moyenne des energies : 299.49
  Moyenne des moyennes d'enveloppe : 0.21
  Moyenne des variances d'enveloppe : 0.10
  Energie totale : 119796.63

Classe 4 :
Nombre de signaux : 350
  Moyenne des energies : 603.11
  Moyenne des moyennes d'enveloppe : 0.39
  Moyenne des variances d'enveloppe : 0.14
  Energie totale : 211087.93
```

**Le code est ci-dessous dans les annexes*

3. *Prétraitement des données :*

Pour effectuer une *Analyse Factorielle Discriminante* (AFD) et une *Analyse en Composantes Principales* (ACP), Les données doivent être normalisées (centrées-réduites) si les variables ont des échelles différentes.

- L'AFD maximise la séparabilité entre les classes en cherchant les directions qui maximisent le rapport entre la variance inter-classe et intra-classe.
- L'ACP cherche des directions d'inertie maximale (variance) sans tenir compte des classes.

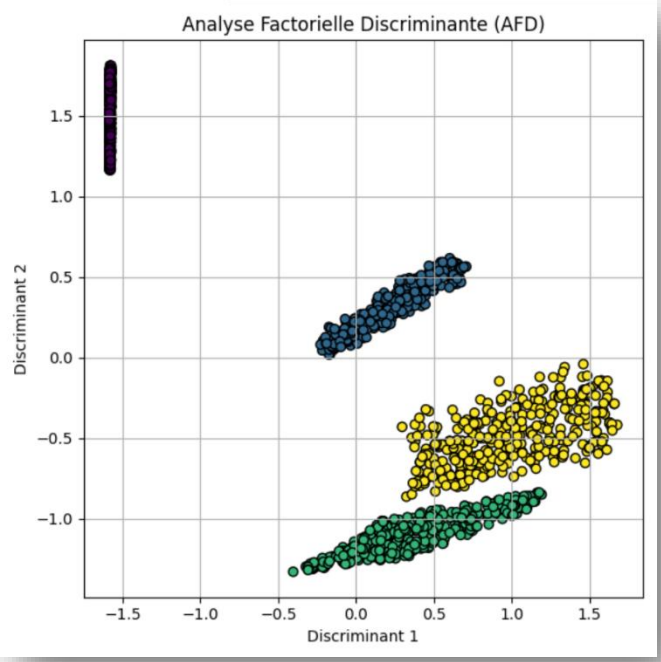
Donc on va calculer les matrices de variance et les matrices de covariance

```
[31mAttention pb de calcul pour les variances (VT<>VA+VE)!
 VT=938834.1740677891, VA=1389967.2004078976, VE=452446.51259140397, difference=-903579.5389315125[0m
[31mAttention pb de calcul sur la matrice de covariance totale CT!
[0m
[31mAttention pb de calcul sur la matrice de covariance intraclasses CA!
[0m
[31mAttention pb de calcul sur la matrice de covariance interclasses CE!
[0m
Centres de gravite :
 [[9.68932852e+02 5.43618356e-01 8.36625496e-02]
 [2.04616275e+03 9.97702597e-01 5.24274582e-04]
 [1.03154823e+03 6.37810523e-01 9.46977958e-02]
 [2.99491582e+02 2.05786939e-01 1.02589751e-01]
 [6.03108365e+02 3.94891018e-01 1.35710955e-01]]
Matrices de variance (Total, Intra, Inter) :
 938834.1740677891 1389967.2004078976 452446.51259140397
Matrices de covariance (Total, Intra, Inter) :
 [[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]] [[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]] [[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
```

**Le code est ci-dessous dans les annexes*

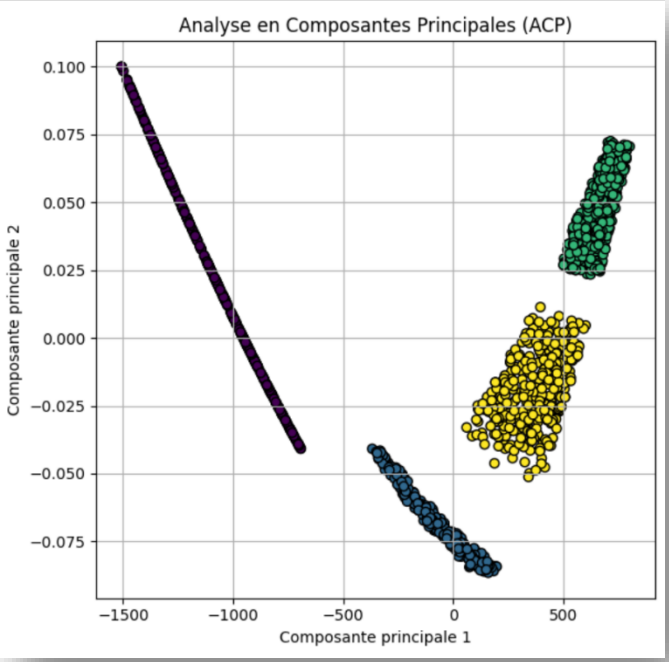
a. *Analyse Factorielle Discriminante (AFD) :*

Pour l'AFD, la matrice de covariance intra-classe est inversée, puis multipliée par la matrice de covariance inter-classe pour obtenir les valeurs propres et les vecteurs propres. Les individus centrés-réduits sont ensuite projetés sur les vecteurs propres associés aux plus grandes valeurs propres, ce qui maximise la séparation entre les classes



b. Analyse en Composantes Principales (ACP) :

Pour l'ACP, les vecteurs propres de la matrice de covariance des paramètres sont extraits, et les données sont projetées sur les deux premiers vecteurs propres correspondant aux directions de la plus grande variance. Les résultats montrent une séparation nette des classes, mettant en évidence les relations structurelles dans les données.



Pourquoi centrer-réduire les données ?

Les données doivent être normalisées (centrées-réduites) si les variables ont des échelles différentes pour éviter que certaines dominent l'analyse. La centration-réduction est justifiée pour les données ayant des unités différentes ou des distributions déséquilibrées. Donc si les données ne sont pas centrées-réduites, les variables avec de grandes amplitudes auront un poids excessif dans l'ACP et l'AFD, faussant les résultats.

4. *Conclusion :*

Dans cette partie de TP, on a exploré plusieurs étapes pour analyser et classifier des données issues de signaux

- Extraction des paramètres des signaux : À partir des signaux bruts, nous avons extrait trois paramètres significatifs (l'énergie, la moyenne et la variance de l'enveloppe).
- Analyse des matrices de variance et de covariance : On a calculé les matrices intra-classes, inter-classes et totales pour évaluer la dispersion des données. Ces matrices ont permis d'analyser la séparation entre les classes et de poser les bases des analyses discriminantes.
- Analyse Factorielle Discriminante (AFD) : L'AFD a été utilisée pour maximiser la séparation entre les classes. Cela a permis de visualiser et de confirmer la capacité des paramètres extraits à discriminer efficacement les différentes classes.
- Analyse en Composantes Principales (ACP) : L'ACP a fourni une autre méthode de réduction dimensionnelle, basée sur la maximisation de la variance totale des données. Bien qu'elle ne tienne pas compte des classes, elle a offert une perspective complémentaire sur la structure des données.

Annexes

1. *Analyse du signal :*

   a. *Etude temporel (la* transformée de Hilbert des signaux et l'enveloppe)

```
import numpy as np
from scipy.signal import hilbert
import TPClassif2024 as tpc
import matplotlib.pyplot as plt

print("Lecture de la base d'apprentissage.\n")
Enveloppe = np.abs(hilbert(tpc.MatriceDonnees))
# Visualiser l'enveloppe temporelle pour quelques signaux de chaque classe
for classe in np.unique(tpc.NoClasse):
    indices_classe = np.where(tpc.NoClasse == classe)[0]
    for i in indices_classe[:3]:  # Choisissez quelques signaux à visualiser
        plt.plot(Enveloppe[i], label=f'Classe {classe}, Signal {i+1}')

    plt.xlabel('Temps')
    plt.ylabel("Amplitude de l'enveloppe")
    plt.title(f"Enveloppe Temporelle des signaux par classe - famille {classe}")
    plt.show()
```

   b. *Etude fréquentielle (la* transformée de Hilbert des signaux et l'enveloppe)

```
import numpy as np
from scipy.signal import hilbert
import TPClassif2024 as tpc
import matplotlib.pyplot as plt
# Lecture de la base d'apprentissage
print("Lecture de la base d'apprentissage.\n")
Enveloppe = np.abs(hilbert(tpc.MatriceDonnees))  # Calcul de l'enveloppe des
signaux
N = len(tpc.MatriceDonnees[0])  # Taille du signal
frequencies = np.fft.fftfreq(N)  # Fréquences associées à la FFT
# Afficher l'analyse fréquentielle pour quelques signaux
for classe in np.unique(tpc.NoClasse):
    indices_classe = np.where(tpc.NoClasse == classe)[0]
    for i in indices_classe[:3]:  # Choisir quelques signaux
        # Appliquer la FFT sur l'enveloppe et obtenir le spectre complet
        fft_signal = np.fft.fft(Enveloppe[i])
        plt.plot(frequencies, np.abs(fft_signal), label=f'Classe {classe}, Signal
{i+1}')
# Paramétrer le graphique
```

```
plt.xlabel('Fréquence (Hz)');plt.ylabel('Amplitude')
plt.title("Analyse Fréquentielle des Signaux
(Enveloppes)");plt.grid(True);plt.show()
```

2. *Extraction de paramètres pertinents :*

a. Trois paramètres pour chaque signal : *l'énergie, la moyenne et la variance de l'enveloppe.*

```
import numpy as np
from scipy.signal import hilbert
import TPClassif2024 as tpc
import matplotlib.pyplot as plt
Enveloppe = np.abs(hilbert(tpc.MatriceDonnees))  # Calcul de l'enveloppe des
signaux
# Extraire les paramètres pour chaque signal
params = []
for classe in np.unique(tpc.NoClasse):
    indices_classe = np.where(tpc.NoClasse == classe)[0]
    for i in indices_classe:  # Pour chaque signal dans la classe
        # Calculer l'énergie, la moyenne et la variance de l'enveloppe
        Energie = np.sum(np.square(Enveloppe[i]))  # Calcul de l'énergie
        Moyenne = np.mean(Enveloppe[i])  # Moyenne de l'enveloppe
        Variance = np.var(Enveloppe[i])  # Variance de l'enveloppe
        # Ajouter les paramètres dans une liste
        params.append([Energie,Moyenne,Variance])
params = np.array(params)
# Affichage des résultats
for i, param in enumerate(params,start=1):
    print(f"Signal {i}:");print(f"  Energie: {param[0]}")
    print(f"  Moyenne de l'enveloppe: {param[1]}")
    print(f"  Variance de l'enveloppe: {param[2]}");print()
```

b. Application des paramètres à chaque classe

```
import numpy as np
from scipy.signal import hilbert
import TPClassif2024 as tpc
Enveloppe = np.abs(hilbert(tpc.MatriceDonnees))  # Calcul de l'enveloppe des
signaux
# Initialiser un dictionnaire pour stocker les paramètres par classe
params_by_class = {}
# Calcul des paramètres pour chaque classe
for classe in np.unique(tpc.NoClasse):
```

```
    indices_classe = np.where(tpc.NoClasse == classe)[0]
    class_params = []  # Liste des paramètres pour cette classe
    for i in indices_classe:
        # Calcul des paramètres pour chaque signal
        Energie = np.sum(np.square(Enveloppe[i]))
        Moyenne = np.mean(Enveloppe[i])
        Variance = np.var(Enveloppe[i])
        # Ajouter les paramètres du signal à la classe
        class_params.append([Energie, Moyenne, Variance])
    # Convertir en numpy array pour une manipulation plus facile
    params_by_class[classe] = np.array(class_params)
# Affichage des résultats par classe
for classe, params in params_by_class.items():
    print(f"\nClasse {classe} :");print(f"Nombre de signaux : {len(params)}")
    print(f"  Moyenne des energies : {np.mean(params[:, 0]):.2f}")
    print(f"  Moyenne des moyennes d'enveloppe : {np.mean(params[:, 1]):.2f}")
    print(f"  Moyenne des variances d'enveloppe : {np.mean(params[:, 2]):.2f}")
    print(f"  Energie totale : {np.sum(params[:, 0]):.2f}")
```

3. *Prétraitement des données :*

a. Les matrices de variance et les matrices de covariance

```
import numpy as np
from scipy.signal import hilbert
import TPClassif2024 as tpc
from ModuleTPClassif import (CalculerIndividusCentresReduits,CalculerVariances,
                             CalculerMatricesCovariance,CalculerCentresGravite)
# Calcul de l'enveloppe des signaux
Enveloppe = np.abs(hilbert(tpc.MatriceDonnees))
# Calcul des paramètres pour chaque signal
params = np.array([
    [np.sum(np.square(Enveloppe[i])), np.mean(Enveloppe[i]),
np.var(Enveloppe[i])]
    for i in range(len(Enveloppe))
])
# Calcul des centres de gravité
CentresGravite = CalculerCentresGravite(params, tpc.NoClasse)
# Centrage et réduction des individus
IndividusCentresReduits = CalculerIndividusCentresReduits(params, CentresGravite)
# Calcul des matrices de variance
VT, VA, VE = CalculerVariances(IndividusCentresReduits, tpc.NoClasse,
CentresGravite)
# Calcul des matrices de covariance
```

```
CT, CA, CE = CalculerMatricesCovariance(IndividusCentresReduits, tpc.NoClasse,
CentresGravite)
# Affichage des résultats
print("Centres de gravite :\n", CentresGravite)
print("Matrices de variance (Total, Intra, Inter) :\n", VT, VA, VE)
print("Matrices de covariance (Total, Intra, Inter) :\n", CT, CA, CE)
```

b. Analyse en Composantes Principales (ACP)/ Analyse Factorielle Discriminante (AFD) :

```
import numpy as np
from scipy.signal import hilbert
import matplotlib.pyplot as plt
import TPClassif2024 as tpc
from ModuleTPClassif import (CalculerIndividusCentresReduits, CalculerVariances,
                             CalculerMatricesCovariance, CalculerCentresGravite)
# Calcul de l'enveloppe des signaux
Enveloppe = np.abs(hilbert(tpc.MatriceDonnees))
params = np.array([
    [np.sum(np.square(Enveloppe[i])), np.mean(Enveloppe[i]),
np.var(Enveloppe[i])]
    for i in range(len(Enveloppe))
])
CentresGravite = CalculerCentresGravite(params, tpc.NoClasse)
IndividusCentresReduits = CalculerIndividusCentresReduits(params, CentresGravite)
VT, VA, VE = CalculerVariances(IndividusCentresReduits, tpc.NoClasse,
CentresGravite)
CT, CA, CE = CalculerMatricesCovariance(IndividusCentresReduits, tpc.NoClasse,
CentresGravite)

# **1. Analyse Factorielle Discriminante (AFD)**
# Régularisation de CA
epsilon = 1e-6
CA_reg = CA + epsilon * np.eye(CA.shape[0])
# Calcul de S_W^-1 * S_B
Sw_inv = np.linalg.inv(CA_reg) @ CE
eigvals_afd, eigvecs_afd = np.linalg.eig(Sw_inv)
# Trier les vecteurs propres selon les valeurs propres décroissantes
sorted_indices = np.argsort(eigvals_afd)[::-1]
eigvecs_afd = eigvecs_afd[:, sorted_indices]
# Projeter les données sur les deux premières composantes discriminantes
X_AFD = IndividusCentresReduits @ eigvecs_afd[:, :2]

# **2. Analyse en Composantes Principales (ACP)**
# Centrer les données pour l'ACP
params_centered = params - np.mean(params, axis=0)
```

```
# Décomposition en valeurs singulières
_, _, Vt = np.linalg.svd(params_centered)
W_ACP = Vt[:2, :].T  # Deux premières composantes principales
X_ACP = params_centered @ W_ACP

# Plot AFD
plt.figure(figsize=(12, 6));plt.subplot(1, 2, 1)
plt.scatter(X_AFD[:, 0], X_AFD[:, 1], c=tpc.NoClasse, cmap='viridis',
edgecolors='k')
plt.title("Analyse Factorielle Discriminante (AFD)")
plt.xlabel("Discriminant 1");plt.ylabel("Discriminant 2");plt.grid()

# Plot ACP
plt.subplot(1, 2, 2)
plt.scatter(X_ACP[:, 0], X_ACP[:, 1], c=tpc.NoClasse, cmap='viridis',
edgecolors='k')
plt.title("Analyse en Composantes Principales (ACP)")
plt.xlabel("Composante principale 1");plt.ylabel("Composante principale
2");plt.grid()

plt.tight_layout();plt.show()
```