Travaux Pratiques I

ENSEIRB-MATMECA – ETE5-MATH1
Enzo Iglésis – enzo.iglesis@bordeaux-inp.fr

Novembre 2025

1 Introduction

1.1 Contexte

— Python est aujourd'hui l'un des langages de programmation les plus utilisés au monde (voire le plus populaire selon plusieurs classements en 2025).
— Il offre un vaste champ d'applications : du simple script d'automatisation quotidienne aux bibliothèques logicielles complètes et aux applications industrielles.
— C'est un langage à haut niveau d'abstraction, facile à apprendre et à utiliser, favorisant la lisibilité du code.
— Python peut tirer parti de bibliothèques très performantes développées en C ou C++ grâce à des mécanismes de *binding* (liaisons entre langages), permettant de combiner la simplicité de Python avec la vitesse du code compilé.
— Le développeur est largement libéré de la gestion de la mémoire.
— Python s'impose de plus en plus dans le monde scientifique et technique grâce à un écosystème riche de bibliothèques performantes : `NumPy`, `SciPy`, `TensorFlow`, `pandas`, etc.
— Le langage est multiplateforme et bénéficie d'une excellente portabilité (bien que les mises à jour fréquentes du langage puissent impacter cette portabilité).
— Enfin, Python est gratuit, libre et open-source, à la différence de certains environnements propriétaires comme `Matlab`.

1.2 Objectifs

— S'initier à la syntaxe et aux types de base de Python.
— Savoir écrire des fonctions, manipuler des listes et des dictionnaires.
— Manipuler des fichiers textes et des données tabulaires simples.
— Réaliser des calculs sur des matrices et des données numériques.
— Afficher et interpréter des résultats sous forme de graphiques.
— Appliquer ces connaissances dans de petits projets pratiques.

1.3 Consignes générales

— Écrivez votre code dans des fichiers Python (.py). Chaque exercice doit avoir son fichier nommé clairement (ex : `exo1.py`).
— Commentez votre code et testez-le.
— Rendre : une archive zip contenant les fichiers .py
— Le rendu des travaux se fera par un envoi par mail à `enzo.iglesis@bordeaux-inp.fr` avec pour objet `[ETE5-MATH1] Python`.

2 Exercices

2.1 Exercice 1 : Types et opérations (20 min)

Écrire un script `exo1.py` qui :

1. Demande à l'utilisateur de saisir deux nombres (entiers ou réels).
2. Affiche leur somme, différence, produit et quotient (si le diviseur n'est pas nul).
3. Convertit les nombres en chaînes et affiche la concaténation.
4. Affiche le type Python de chaque variable saisie.

2.2 Exercice 2 : Boucles, tests et fonctions (25 min)

Écrire un script `exo2.py` qui :

1. Permet grace a une fonction `is_prime(n)` de tester si un nombre n est premier.
2. Demande un entier N et liste tous les nombres premiers $\leqslant N$.
3. Affiche le temps d'exécution (utiliser `time.time()`).

2.3 Exercice 3 : Listes, dictionnaires, tuples et compréhensions (20 min)

Écrire un script `exo3.py` qui :

1. À partir d'une liste de noms `names`, construire un dictionnaire qui compte l'occurrence de la première lettre (ex : `{'A':3, 'B':2}`).
2. Trie une liste de tuples `(nom, âge)` par ordre décroissant d'âge.
3. Utilise une "compréhension de liste" qui extrait tous les nombres pairs d'une liste.

2.4 Exercice 4 : Fichiers et traitement de texte (30 min)

1. Écrire un script `exo4_texte.py` qui lit un fichier texte et renvoie :
   — le nombre total de mots,
   — la fréquence de chaque mot (ignorer la casse et la ponctuation simple).
2. Écrire un script `exo4_csv.py` qui lit un CSV simple (trois colonnes : id, nom, score) et calcule la moyenne des scores.

2.5 Exercice 5 : Utilisation de bibliothèques (30–45 min)

Écrire un script `exo5.py` qui :

1. Calcule la moyenne, la variance et la médiane d'un grand tableau initialisé aléatoirement en utilisant `numpy`.
2. Utilise `pandas` pour charger un fichier CSV et effectuer quelques agrégations (par exemple, `groupby`).
3. Trace un histogramme simple avec `matplotlib` (exemple : distribution des scores).

2.6 Exercice 6 : Visualisation de données (30–45 min)

Écrire un script `exo6.py` qui :

1. Lit un fichier CSV contenant deux colonnes numériques, `x` et `y`.
2. Affiche un graphique de `y` en fonction de `x`. Le graphique doit inclure :
   — des marqueurs pour chaque point (`o`) et une ligne reliant les points,
   — un titre,
   — un label pour l'axe X et un label pour l'axe Y,
   — une grille pour faciliter la lecture des valeurs.

2.7 Exercice 7 : Classes et méthodes (30 min)

Écrire un script `exo7.py` qui reprend l'Exercice 5, mais :

1. Contient une classe qui permet :
   — de construire un objet à partir du nom d'un fichier `.csv` et de stocker ses valeurs dans un `DataFrame` (`pandas`),
   — d'avoir une méthode permettant de faire des agrégations (`groupby`),
   — de définir la méthode `__repr__` qui affiche la moyenne, la variance et la médiane des données,
   — d'avoir une méthode permettant de tracer un histogramme simple avec `matplotlib`.
2. Utilise la classe créée avec deux fichiers `.csv` différents.

2.8 Exercice 8 : Opérations matricielles (30 min)

Écrire un script `exo8.py` qui :

1. Crée deux matrices A et B de taille 3×3.
2. Affiche les deux matrices.
3. Calcule et affiche :
   — la somme des deux matrices,
   — la différence,
   — le produit élément par élément,
   — le produit matriciel.
4. Calcule et affiche :
   — la transposée de A,
   — le déterminant de A,
   — l'inverse de A (en gérant les cas où la matrice n'est pas inversible).
5. Affiche la forme (`shape`), le type des éléments (`dtype`) et le nombre total d'éléments de la matrice A.

3 Mini-projet : Transmission d'un signal sur un canal bruité

L'objectif de ce mini-projet est de simuler, à l'aide d'un script Python, la transmission d'un signal sur un canal bruité et d'observer les effets du bruit sur le signal reçu. Le projet doit être structuré autour de plusieurs classes Python représentant les différentes parties du système de communication.

Description du projet

Le programme devra permettre de :
— Générer un signal sinusoïdal représentant le signal émis par la source.
— Faire passer ce signal dans un canal bruité (ajout d'un bruit blanc gaussien).
— Afficher le signal original, le signal bruité et le signal filtré.
— Calculer le rapport signal/bruit (SNR) du canal.
Chaque étape devra être réalisée à l'aide d'une classe spécifique :
— `Signal` : génère le signal sinusoïdal.
— `Canal` : ajoute le bruit blanc gaussien et calcule le SNR.
— `Filtre` : applique un filtre passe-bas pour atténuer le bruit.
Un script Python coordonnera les étapes de la transmission et affichera les résultats finaux.

Rappel théorique

Lorsqu'un signal est transmis sur un canal bruité, il est perturbé par un bruit aléatoire appelé **bruit blanc gaussien**. Le signal reçu peut s'écrire :

$$s_r(t) = s(t) + b(t)$$

où :
— $s(t)$ est le signal émis,
— $b(t)$ est le bruit, de moyenne nulle et de variance σ^2.
En Python, un bruit gaussien peut être généré à l'aide de la bibliothèque `numpy` :

```
import numpy as np
b = amplitude * np.random.normal(moyenne, ecart_type, taille)
```

Le rapport signal/bruit (SNR, *Signal-to-Noise Ratio*) s'exprime en décibels :

$$SNR = 10 \log_{10} \left(\frac{P_s}{P_b} \right)$$

où P_s est la puissance moyenne du signal utile et P_b celle du bruit. En Python, on peut le calculer ainsi :

```
import numpy as np
SNR = 10 * np.log10(puissance_signal / puissance_bruit)
```

Un filtre passe-bas peut ensuite être appliqué pour atténuer le bruit haute fréquence et retrouver le signal utile. Par exemple, pour un filtre de Butterworth :

```
from scipy.signal import butter, filtfilt

b, a = butter(4, freq_coupure / (freq_echantillonnage / 2), btype="low")
signal_filtre = filtfilt(b, a, signal)
```

Classes à implémenter

Classe `Signal` Cette classe permet de générer un signal sinusoïdal :

$$s(t) = A \sin(2\pi f t)$$

avec :
- A : amplitude du signal,
- f : fréquence du signal (en Hz),
- F_e : fréquence d'échantillonnage,
- $T_e = \frac{1}{F_e}$: période d'échantillonnage.

Classe `Canal` Cette classe modélise la propagation du signal dans un canal bruité :

$$s_r(t) = s(t) + \sigma\, n(t)$$

où $n(t)$ est un bruit blanc gaussien normalisé. Elle devra également contenir une méthode pour calculer le SNR du canal.

Classe `Filtre` Cette classe applique un filtre passe-bas de Butterworth afin d'atténuer le bruit, en utilisant la fonction `butter()` du module `scipy.signal`.

Remarque sur la méthode `__call__`

En Python, il est possible d'utiliser directement une instance de classe comme une fonction grâce à la méthode spéciale `__call__`. Par exemple, pour appliquer un filtre à un signal :

```
signal = Signal(...)
filtre = Filtre(...)

filtre(signal)  # Appelle automatiquement filtre.__call__(signal)
```

Affichage des résultats

Le programme devra afficher, à l'aide de `matplotlib`, les trois signaux suivants :
- Le signal original (en vert),
- Le signal bruité (en rouge),
- Le signal filtré (en bleu).

Le SNR devra être calculé et affiché dans la console.

Utilisation de `argparse`

Pour passer les paramètres du programme (amplitude, fréquence, durée, bruit, etc.), vous utiliserez le module `argparse`. Voici un exemple de structure à compléter :

```python
# ============================================================
# Mini-projet : Transmission sur canal bruité
# ============================================================
# Objectif :
#   - Utiliser des classes pour modéliser un système de transmission simple
#   - Générer un signal sinusoïdal
#   - Le faire passer dans un canal bruité
#   - Filtrer le signal reçu
# ============================================================

import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import butter, filtfilt
import argparse

# ------------------------------
# Classe 1 : Signal émis
# ------------------------------
class Signal:
    def __init__(self, ...):
        """
        amplitude : amplitude du signal (V)
        frequence : fréquence du signal (Hz)
        duree : durée du signal (s)
        Fe : fréquence d'échantillonnage (Hz)
        """
        self.amplitude = amplitude
        self.frequence = frequence
        self.duree = duree
        self.Fe = Fe
        self.Te = 1 / Fe
        self.t = np.arange(0, duree, self.Te)
        self.signal = self.amplitude * np.sin(2 * np.pi * self.frequence * self.t)

    def __call__(self):
        """Retourne le signal et le vecteur temps"""
        return self.t, self.signal

# ------------------------------
# Classe 2 : Canal bruité
# ------------------------------
class Canal:
    def __init__(self, ...):
        ...
    def __call__(self, signal):
        """Ajoute du bruit blanc gaussien au signal."""
        ...
    def SNR(self, ...):
        """Calcule le rapport signal/bruit (SNR en dB)."""
        ...

# ------------------------------
# Classe 3 : Filtre passe-bas
# ------------------------------
class Filtre:
    def __init__(self, ...):
        ...
    def __call__(self, signal):
        """Applique un filtre passe-bas de Butterworth au signal."""
        ...

# ------------------------------
```

```python
# Programme principal (main)
# ----------------------------
parser = argparse.ArgumentParser(description="Simulation de transmission sur canal bruité ")

parser.add_argument("--amplitude", type=float, default=1.0, help="Amplitude du signal (ex : 1.0)")
parser.add_argument("--frequence", type=float, default=10.0, help="Fréquence du signal en Hz (ex: 10)")
parser.add_argument("--duree", type=float, default=1.0, help="Durée du signal en secondes (ex: 1)")
parser.add_argument("--Fe", type=float, default=1000.0, help="Fréquence d'échantillonnage en Hz (ex: 1000)")
parser.add_argument("--bruit", type=float, default=0.5, help="Intensité du bruit (ex: 0.5)")
parser.add_argument("--fc", type=float, default=20.0, help="Fréquence de coupure du filtre en Hz (ex: 20)")

args = parser.parse_args()

# === Étape 1 : Génération du signal ===
...

# === Étape 2 : Transmission sur canal bruité ===
...

# === Étape 3 : Filtrage du signal ===
...

# === Étape 4 : Affichage des résultats ===
...
```

Exemple d'utilisation du programme:

```
python projet.py --amplitude 1 --frequence 10 --duree 1 --Fe 1000 --bruit 0.5 --fc 20
```

Voici l'affichage attendu avec les paramètres fournis:

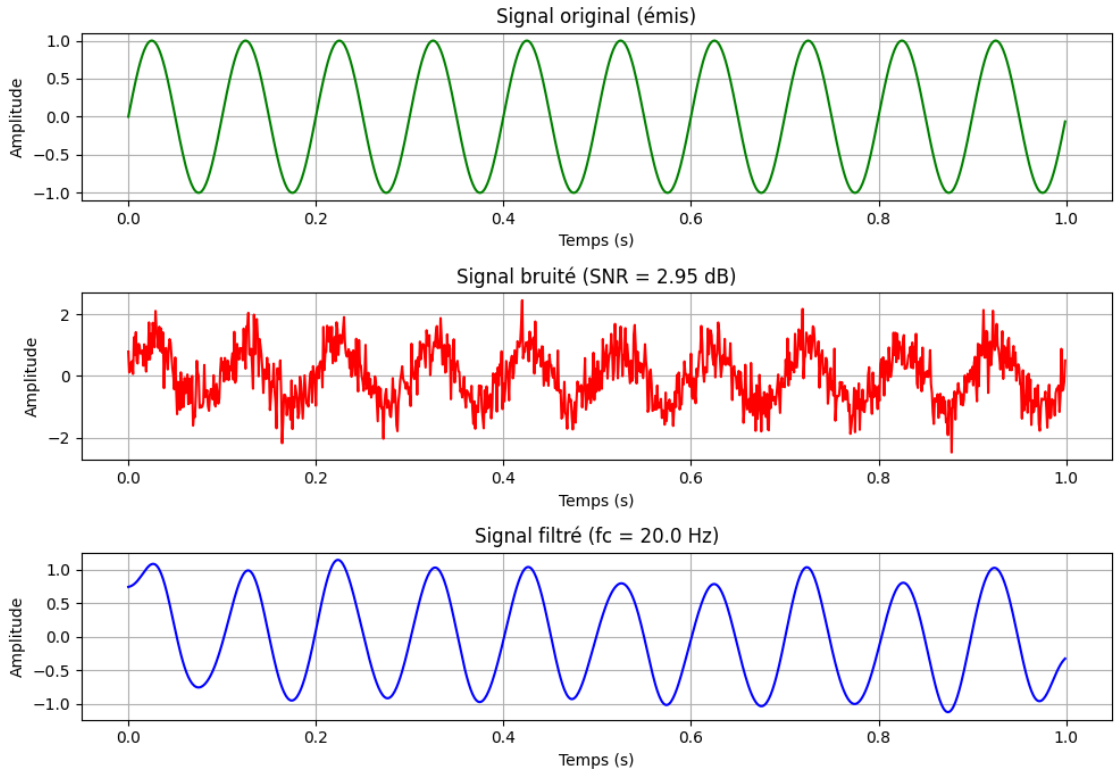

FIGURE 1 – Exemple d'affichage du programme avec les signaux original, bruité et filtré.

A Installation et Utilisation de Python

A.1 Installer Python

Python est un langage libre et gratuit, disponible sur toutes les plateformes (Windows, macOS, Linux). La version recommandée pour ce TP est **Python 3.10** ou supérieure.

- **Sous Windows :** Rendez-vous sur python.org/downloads et téléchargez la dernière version stable adaptée à votre système d'exploitation. Lors de l'installation, cochez la case `Add Python to PATH` pour pouvoir exécuter Python directement depuis le terminal.
- **Sous Linux / macOS :** Python est souvent déjà installé. Vous pouvez vérifier la version avec :

  ```
  python3 --version
  ```

  Si nécessaire, installez ou mettez à jour Python via le gestionnaire de paquets :

  ```
  sudo apt install python3 python3-pip   # Ubuntu/Debian
  brew install python                    # macOS (Homebrew)
  ```

A.2 Créer un environnement Python isolé

Pour travailler efficacement et éviter les conflits entre bibliothèques, il est recommandé d'utiliser un **environnement virtuel** Python. Cela permet d'avoir un espace isolé avec toutes les bibliothèques nécessaires à votre projet.

- **Créer un environnement virtuel** nommé `.venv` dans votre projet :

  ```
  python -m venv .venv
  ```

- **Activer l'environnement virtuel** :
  - **Sous Windows :** `.venv\Scripts\activate`
  - **Sous Linux / macOS :** `source .venv/bin/activate`
- Une fois activé, vous pouvez installer les bibliothèques nécessaires pour votre TP ou projet, par exemple :

  ```
  pip install numpy pandas matplotlib scipy
  ```

- Pour quitter l'environnement virtuel :

  ```
  deactivate
  ```

L'utilisation d'un environnement virtuel assure que chaque projet reste indépendant et que les bibliothèques installées n'affectent pas le reste du système.

A.3 Lancer Python

Plusieurs options sont possibles pour exécuter du code Python :

- **Depuis un terminal** : Tapez `python` ou `python3` pour ouvrir l'interpréteur interactif. Vous pouvez également exécuter un fichier avec :

  ```
  python3 mon_script.py
  ```

- **Depuis un IDE** : Utilisez un environnement de développement comme **Visual Studio Code**, **PyCharm**, ou **Spyder**.
- **Depuis un notebook Jupyter** : Outils qui permettent de combiner code, texte et graphiques :

  ```
  jupyter notebook
  ```

  puis ouvrez le fichier `.ipynb` dans votre navigateur.

A.4 Vérifier l'installation

Une fois Python installé, testez votre environnement avec les commandes suivantes :

```
python3 -m pip show numpy
python3 -c "import matplotlib; import numpy; print('Python fonctionne !')"
```

Si ces commandes s'exécutent sans erreur, votre installation est opérationnelle.

A.5 L'interpréteur interactif Python

En tant que langage interprété, Python peut être lancé directement dans un terminal, ce qui permet de taper les commandes une par une et de voir immédiatement leur résultat.

```
$ python3
Python 3.13.7 (main, Aug 20 2025, 22:17:40) [GCC 15.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello world!")
Hello world!
>>> 3+5
8
>>>
```

Chaque ligne est évaluée immédiatement et la sortie est affichée automatiquement dans le terminal. Pour quitter l'interpréteur, il suffit de taper :

```
>>> exit()
```

Ce mode interactif est très pratique pour tester rapidement de petites commandes, vérifier des calculs ou expérimenter avec des fonctions Python.

A.6 Script Python

Un script Python est un fichier avec l'extension `.py`. Il s'organise généralement de la manière suivante :
- L'importation des bibliothèques, fonctions, classes, etc.
- La définition des fonctions, classes, méthodes, etc.
- Le programme principal, qui peut faire appel aux importations et aux définitions locales.

B Corrigés

B.1 Exercice 1

Correction

Exercice 1 : Types et opérations

```python
# exo1.py
a = input("Entrez le premier nombre: ")
b = input("Entrez le second nombre: ")
# essayer de convertir en float si possible
try:
    af = float(a)
    bf = float(b)
    print("Somme:", af + bf)
    print("Différence:", af - bf)
    print("Produit:", af * bf)
    print("Quotient:", af / bf)
except ZeroDivisionError:
    print("Division impossible : dénominateur est égale à 0.")
except ValueError:
    print("Au moins une saisie n'est pas un nombre.")
# concaténation
print("Concaténation:", a + b)
print("Types:", type(a), type(b))
```

B.2 Exercice 2

Correction

Exercice 2 : Boucles, tests et fonctions

```python
import time


def is_prime(n):
    if n <= 1:
        return False
    if n <= 3:
        return True
    if n % 2 == 0:
        return False
    i = 3
    while i * i <= n:
        if n % i == 0:
            return False
        i += 2
    return True


N = int(input("Entrez N: "))
t0 = time.time()
primes = []
for k in range(2, N + 1):
    if is_prime(k):
        primes.append(k)
t1 = time.time()

print(f"Primes <= {N}: {primes}")
time_str = f"Temps (s): {(t1 - t0):.3f}"
print(time_str)
```

B.3 Exercice 3

Correction

Exercice 3 : Listes, dictionnaires, tuples et compréhensions

```python
# exo3.py
names = ["Alice", "Bob", "Amine", "Claire", "Benoit", "Anne"]
# compter premières lettres
counts = {}
for name in names:
    k = name[0].upper()
    if k in counts:
        counts[k] += 1
    else:
        counts[k] = 1
print(counts)


def age(x):
    return x[1]


# tri par age
people = [("Alice", 30), ("Bob", 25), ("Claire", 35)]
ages = {x[0]: x[1] for x in people}
people_sorted = sorted(people, key=lambda x: x[1], reverse=True)
print(people_sorted)

# comprehension pairs
nums = list(range(1, 21))
pairs = [x for x in nums if x % 2 == 0]

print(pairs)
```

B.4 Exercice 4

Correction

Exercice 4 : Fichiers et traitement de texte (texte)

```python
# exo4_texte.py
import re
from collections import Counter

fname = input("Nom du fichier texte: ")
with open(fname, "r", encoding="utf-8") as f:
    text = f.read().lower()
# enlever ponctuation simple
words = re.findall(r"\b[\w']+\b", text, flags=re.UNICODE)
print("Nombre de mots:", len(words))
freq = Counter(words)
for w, c in freq.most_common(20):
    print(w, c)

counter = {}
for word in words:
    if word in counter:
        counter[word] += 1
    else:
        counter[word] = 1

counter = dict(sorted(counter.items(), key=lambda x: x[1]))
```

Correction

Exercice 4 : Fichiers et traitement de texte (CSV)

```python
# exo4_csv.py
import csv

fname = input("Nom du fichier texte: ")
nb_lignes = 0
total = 0

with open(fname, newline="", encoding="utf-8") as fichier:
    lecteur = csv.reader(fichier, delimiter=",")
    next(lecteur)  # sauter l'en-tête

    for ligne in lecteur:
        score = float(ligne[2])  # la 3e colonne est le score
        total += score
        nb_lignes += 1

print(f"Moyenne des scores : {total / nb_lignes}")
```

B.5 Exercice 5

Correction

Exercice 5 : Utilisation de bibliothèques

```python
# exo5.py
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# --- Partie 1 : Numpy ---
# Création d'un grand tableau de nombres aléatoires (ex : scores entre 0 et 20)
scores = np.random.randint(0, 21, size=1000)

# Calculs statistiques
moyenne = np.mean(scores)
variance = np.var(scores)
mediane = np.median(scores)

print(
    "=== Statistiques avec NumPy ===\n"
    + f"Moyenne : {moyenne}\n"
    + f"Variance : {variance}\n"
    + f"Médiane : {mediane}\n\n"
)

# --- Partie 2 : Pandas ---
# Exemple : lecture d'un fichier CSV avec des colonnes : id, nom, groupe, score
# Exemple de fichier CSV :
# id,nom,groupe,score
# 1,Alice,A,15
# 2,Bob,A,12
# 3,Charlie,B,18
# 4,Diane,B,14

# Chargement du fichier CSV
fname = input("Nom du fichier csv: ")
df = pd.read_csv(fname, delimiter=";")

print("=== Aperçu du fichier CSV ===")
print(df.head(), "\n")
```

```python

# Agrégations simples avec groupby
moyennes_groupes = df.groupby("limite")["mesure"].mean()
print("Moyenne des scores par groupe :")
print(moyennes_groupes)

# --- Partie 3 : Matplotlib ---
# Tracer un histogramme de la distribution des scores
for name, group in df.groupby("limite"):
    plt.hist(
        group["mesure"],
        bins=20,
        alpha=0.5,
        label=f"limite = {name}",
        edgecolor="black",
    )

plt.title("Distribution des vitesse par limite")
plt.xlabel("Vitesse")
plt.ylabel("Nombre")
plt.legend()
plt.show()

df.boxplot(column="mesure", by="limite", grid=False)
plt.title("Distribution des mesures par limite")
plt.suptitle("")  # remove automatic title
plt.xlabel("Limite")
plt.ylabel("Mesure")
plt.show()
```

B.6 Exercice 6

Correction

Exercice 6 : Visualisation de données

```python
# exo6.py
import matplotlib.pyplot as plt
import csv


def read_csv_data(filename):
    """
    Lit un fichier CSV contenant deux colonnes (x et y) et retourne deux listes.
    """
    x = []
    y = []
    with open(filename, newline="", encoding="utf-8") as csvfile:
        reader = csv.reader(csvfile)
        header = next(reader, None)  # Ignore l'en-tête si présent
        for row in reader:
            if len(row) >= 2:
                try:
                    x.append(float(row[0]))
                    y.append(float(row[1]))
                except ValueError:
                    print(f"Ignored invalid row: {row}")
    return x, y


def plot_data(x, y, title="Graphique", xlabel="X", ylabel="Y"):
    """
    Affiche les données x et y dans un graphique simple.
```

```
    """
    _, ax = plt.subplots()
    ax.plot(x, y, marker="o", linestyle="-", color="blue")
    ax.set_title(title)
    ax.set_xlabel(xlabel)
    ax.set_ylabel(ylabel)
    ax.grid(True)
    plt.show()


fname = input("Nom du fichier csv: ")
x, y = read_csv_data(fname)
plot_data(x, y, title="Données issues du CSV", xlabel="X", ylabel="Y")
```

B.7 Exercice 7

Correction

Exercice 7 :

```
# exo7.py
import pandas as pd
import matplotlib.pyplot as plt


class AnalyseCSV:
    def __init__(self, fichier_csv):
        """
        Initialise l'objet en chargeant le fichier CSV dans un DataFrame.
        """
        self.fichier_csv = fichier_csv
        try:
            self.df = pd.read_csv(fichier_csv)
        except FileNotFoundError:
            raise FileNotFoundError(f"Le fichier {fichier_csv} est introuvable.")

    def agreger(self, colonne_groupe, colonne_valeur, agg_func="mean"):
        """
        Effectue une agrégation groupby sur le DataFrame.
        colonne_groupe : nom de la colonne pour le groupby
        colonne_valeur : nom de la colonne à agréger
        agg_func : fonction d'agrégation ('mean', 'sum', 'count', etc.)
        """
        if (
            colonne_groupe not in self.df.columns
            or colonne_valeur not in self.df.columns
        ):
            raise ValueError("Les colonnes spécifiées ne sont pas dans le DataFrame
                ↪ .")
        return self.df.groupby(colonne_groupe)[colonne_valeur].agg(agg_func)

    def __repr__(self):
        """
        Affiche la moyenne, la variance et la médiane des colonnes numériques du
            ↪ DataFrame.
        """
        stats = self.df.describe().loc[["mean", "50%", "std"]]
        stats.rename(index={"50%": "median", "std": "variance"}, inplace=True)
        return str(stats)

    def tracer_histogramme(self, colonne, bins=10, titre=None):
        """
        Trace un histogramme simple pour une colonne spécifique.
```

```
        """
        if colonne not in self.df.columns:
            raise ValueError(f"La colonne {colonne} n'existe pas dans le DataFrame.
                ↪ ")
        plt.hist(self.df[colonne], bins=bins, edgecolor="black")
        plt.xlabel(colonne)
        plt.ylabel("Fréquence")
        if titre:
            plt.title(titre)
        plt.grid(True)
        plt.show()


# -----------------------------
# Exemple d'utilisation
# -----------------------------

# Remplacer 'fichier1.csv' et 'fichier2.csv' par vos fichiers réels
fichiers = ["fichier1.csv", "fichier2.csv"]

for f in fichiers:
    print(f"\nAnalyse du fichier : {f}")
    try:
        analyse = AnalyseCSV(f)
        print(analyse)  # Affiche moyenne, variance, médiane
    except FileNotFoundError as e:
        print(f"error: {e}")

    # Exemple d'agrégation
    # Assurez-vous que les colonnes existent dans vos CSV
    # print(analyse.agreger('groupe', 'valeur', 'mean'))

    # Exemple d'histogramme
    # Remplacez 'valeur' par le nom d'une colonne numérique de votre CSV
    # analyse.tracer_histogramme('valeur', bins=15, titre=f'Histogramme {f}')
```

B.8 Exercice 8

Correction

Exercice 8 : Opérations matricielles

```
# exo8.py
import numpy as np

# 1. Création de deux matrices à partir de listes
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])

# 2. Affichage des matrices
print("Matrice A :\n", A)
print("Matrice B :\n", B)

# 3. Opérations de base
print("\nA + B =\n", A + B)
print("A - B =\n", A - B)
print("A * B (produit élément par élément) =\n", A * B)
print("A @ B (produit matriciel) =\n", A @ B)

# 4. Autres opérations
print("\nTransposée de A :\n", A.T)
print("Déterminant de A :", np.linalg.det(A))

```

```python
# Vérification avant inversion
if np.linalg.det(A) != 0:
    print("Inverse de A :\n", np.linalg.inv(A))
else:
    print("A n'est pas inversible (déterminant nul).")

# 5. Informations sur la matrice
print("\nShape de A :", A.shape)
print("Type des éléments :", A.dtype)
print("Nombre total d'éléments :", A.size)
```

B.9 Projet

Correction

Mini-projet : Transmission d'un signal sur un canal bruité

```python
# ============================================================
# Mini-projet : Transmission sur canal bruité
# ============================================================
# Objectif :
#   - Utiliser des classes pour modéliser un système de transmission simple
#   - Générer un signal sinusoïdal
#   - Le faire passer dans un canal bruité
#   - Filtrer le signal reçu
# ============================================================

import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import butter, filtfilt
import argparse


# ------------------------------
# Classe 1 : Signal émis
# ------------------------------
class Signal:
    def __init__(self, amplitude=1.0, frequence=10.0, duree=1.0, Fe=1000):
        """
        amplitude : amplitude du signal (V)
        frequence : fréquence du signal (Hz)
        duree : durée du signal (s)
        Fe : fréquence d'échantillonnage (Hz)
        """
        self.amplitude = amplitude
        self.frequence = frequence
        self.duree = duree
        self.Fe = Fe
        self.Te = 1 / Fe
        self.t = np.arange(0, duree, self.Te)
        self.signal = self.amplitude * np.sin(2 * np.pi * self.frequence * self.t)

    def __call__(self):
        """Retourne le signal et le vecteur temps"""
        return self.t, self.signal


# ------------------------------
# Classe 2 : Canal bruité
# ------------------------------
class Canal:
    def __init__(self, intensite_bruit=0.5):
        """
```

```
        intensite_bruit : écart-type du bruit gaussien ajouté au signal
        """
        self.intensite_bruit = intensite_bruit

    def __call__(self, signal):
        """
        Ajoute du bruit blanc gaussien au signal.
        Retourne le signal bruité.
        """
        bruit = self.intensite_bruit * np.random.normal(0, 1, len(signal))
        signal_bruite = signal + bruit
        return signal_bruite, bruit

    def SNR(self, signal, bruit):
        """Calcule le rapport signal/bruit (SNR en dB)."""
        puissance_signal = np.mean(signal**2)
        puissance_bruit = np.mean(bruit**2)
        return 10 * np.log10(puissance_signal / puissance_bruit)


# ------------------------------
# Classe 3 : Filtre passe-bas
# ------------------------------
class Filtre:
    def __init__(self, Fe, fc=20):
        """
        Fe : fréquence d'échantillonnage (Hz)
        fc : fréquence de coupure du filtre (Hz)
        """
        self.Fe = Fe
        self.fc = fc

    def __call__(self, signal):
        """Applique un filtre passe-bas de Butterworth au signal."""
        b, a = butter(4, self.fc / (self.Fe / 2), btype="low")
        signal_filtre = filtfilt(b, a, signal)
        return signal_filtre


# ------------------------------
# Programme principal (main)
# ------------------------------
# === Définition des arguments via argparse ===
parser = argparse.ArgumentParser(
    description="Simulation de transmission sur canal bruité"
)

parser.add_argument(
    "--amplitude", type=float, default=1.0, help="Amplitude du signal (ex: 1.0)"
)
parser.add_argument(
    "--frequence",
    type=float,
    default=10.0,
    help="Fréquence du signal en Hz (ex: 10)",
)
parser.add_argument(
    "--duree", type=float, default=1.0, help="Durée du signal en secondes (ex: 1)"
)
parser.add_argument(
    "--Fe",
    type=float,
    default=1000.0,
    help="Fréquence d'échantillonnage en Hz (ex: 1000)",
)
```

```python
parser.add_argument(
    "--bruit", type=float, default=0.5, help="Intensité du bruit (ex: 0.5)"
)
parser.add_argument(
    "--fc",
    type=float,
    default=20.0,
    help="Fréquence de coupure du filtre passe-bas en Hz (ex: 20)",
)

args = parser.parse_args()

# === Étape 1 : Génération du signal ===
signal_source = Signal(args.amplitude, args.frequence, args.duree, args.Fe)
t, signal = signal_source()

# === Étape 2 : Transmission sur canal bruité ===
canal = Canal(args.bruit)
signal_bruite, bruit = canal(signal)
SNR = canal.SNR(signal, bruit)
print(f"SNR du canal : {SNR:.2f} dB")

# === Étape 3 : Filtrage du signal ===
filtre = Filtre(args.Fe, args.fc)
signal_filtre = filtre(signal_bruite)

# === Étape 4 : Affichage des résultats ===
plt.figure(figsize=(10, 7))

plt.subplot(3, 1, 1)
plt.plot(t, signal, "g")
plt.title("Signal original (émis)")
plt.xlabel("Temps (s)")
plt.ylabel("Amplitude")
plt.grid()

plt.subplot(3, 1, 2)
plt.plot(t, signal_bruite, "r")
plt.title(f"Signal bruité (SNR = {SNR:.2f} dB)")
plt.xlabel("Temps (s)")
plt.ylabel("Amplitude")
plt.grid()

plt.subplot(3, 1, 3)
plt.plot(t, signal_filtre, "b")
plt.title(f"Signal filtré (fc = {args.fc} Hz)")
plt.xlabel("Temps (s)")
plt.ylabel("Amplitude")
plt.grid()

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