Informatique — MPSI/PCSI, 2024–2025

Correction du TP n° 3

On commence par se placer dans le bon répertoire :

```
import os
os.chdir("chemin vers le dossier TP")
```

puis on importe les bibliothèques nécessaires

```
import math as ma
import matplotlib.pyplot as plt
```

Exercice 3.1 (Tracé de fonctions)

1. Pour définir une liste de n valeurs comprises entre a et b, on définit d'abord le pas définissant les $(n-1)$ intervalles $p = \dfrac{b-a}{n-1}$ tel que la fonction `Discretisation` s'écrive alors :

```
def Discretisation(a,b,n):
    p=(b-a)/(n-1)
    return([a+k*p for k in range(n)])
```

2. Le code suivant avec une définition par compréhension de la séquence des images `Y` pour être certain qu'elle soit de même longueur que celle des antécédents `X` convient.

```
def graphe(f,I,n):
    plt.figure()
    X=Discretisation(I[0],I[1],n)
    Y=[f(x) for x in X]
    plt.plot(X,Y)
    plt.grid()
    plt.show()
```

3. Le code suivant convient.

```
def f(x):
    return(3*ma.sin(x))
graphe(f,[0,2*ma.pi],1000)
```

4. En faisant une boucle sur les valeurs de $k \in K$ et ajoutant un `label` pour distinguer les courbes avec une légende incluant une représentation en chaîne de caractères de l'entier k avec `str(k)`, il vient :

```
def graphes(f,K,I,n):
    plt.figure()
    X=Discretisation(I[0],I[1],n)
    for k in K:
        Y=[f(k,x) for x in X]
        plt.plot(X,Y,label="k="+str(k))
    plt.grid()
    plt.legend(loc=1)
    plt.show()
```

5. Le code suivant convient.

```
def g(k,x):
    return(k*ma.sin(x))
graphes(g,[1,2,5],[0,2*ma.pi],1000)
```

Exercice 3.2 (Lecture de fichiers textes)

1. Pour épurer chaque ligne lue des caractères de fin de ligne `"\r"` ou `"\n"`, on utilise la méthode `strip()`. On peut ensuite faire une découpe de la chaîne de caractères sur un caractère particulier avec la méthode `split`. En prenant le séparateur comme argument, ici un tabulation `"\t"`, sous la forme `split('\t')`, il vient le code suivant.

```
X,Y=[],[]
fichier = open('mesures2D.txt','r')
for ligne in fichier:
        champs=ligne.strip().split('\t')
        X.append(float(champs[0]))
        Y.append(float(champs[1]))
fichier.close()
```

   où nous avons pris soin de convertir en flottant avec la fonction `float` les chaînes de caractères lues.

2. Sachant que `[[] for k in range(n)]` construit une liste de n listes vides par compréhension et remplaçant simplement le `'\t'` en argument de la méthode `split` par `sep`, il vient :

```
def Lecture(nomfichier, n, sep):
    L=[[] for k in range(n)]
    fichier = open(nomfichier,'r')
    for ligne in fichier:
        for k,e in enumerate(ligne.strip().split(sep)):
            L[k].append(float(e))
    fichier.close()
    return(L)
```

3. Avec les commandes suivantes

```
L = Lecture("mesures2D.txt",2,'\t')
M = Lecture("mesures3D.txt",3,'\t')
N = Lecture("mesures3D2.txt",3,';')
```

   on forme bien les listes de flottants attendues.

Exercice 3.3 (Ajustement affine)

1. D'après les expressions données dans le sujet, la fonction `Moyenne` s'écrit simplement :

```
def Moyenne(X):
    s=0
    for x in X:
        s+=x
    return(s/len(X))
```

2. En faisant appel à la fonction `Moyenne`, la fonction `Covariance` s'écrit simplement :

```
def Covariance(X,Y):
    XY=[X[i]*Y[i] for i in range(len(X))]
    return(Moyenne(XY)-Moyenne(X)*Moyenne(Y))
```

   La fonction `Variance` s'écrit alors :

```
def Variance(X):
    return(Covariance(X,X))
```

3. En faisant appel aux trois fonctions sus-définies, la fonction `Regression` peut s'écrire :

```
def Regression(X,Y):
    cxy=Covariance(X,Y)
    vx=Variance(X)
    a=cxy/vx
    b=Moyenne(Y)-a*Moyenne(X)
    R=cxy/ma.sqrt(vx*Variance(Y))
    return(a,b,R)
```

   Nous avons pris soin d'affecter les résultats de `Covariance(X,Y)` et `Variance(X)` utilisés deux fois pour rendre le programme plus efficace.

4. L'utilisation des fonctions `Lecture` et `Regression` avec les données du fichier `mesures2D.txt` est immédiate :

```
L=Lecture("mesures2D.txt",2,'\t')
listeX,listeY=L[0],L[1]
a,b,R=Regression(listeX,listeY)
```

Pour afficher la droite sur le nuage de points avec les indications données dans le sujet, le code suivant convient :

```
X=Discretisation(min(listeX),max(listeX),100)
Y=[a*x+b for x in X]
equation='y='+str(round(a,4))+'*x+'+str(round(b,4))
plt.figure()
plt.plot(listeX,listeY,'or',label="Données")
plt.plot(X,Y,'-b',label="Ajustement affine\n" \
       +equation+'\n R^2='+str(round(R**2)))
plt.xlabel('x')
plt.ylabel('y')
plt.legend(loc=2)
plt.grid()
plt.show()
```

Notez simplement que nous avons affecté la chaîne de caractères associée à l'équation pour une meilleure lecture du code et par anticipation de ce qui suit et inséré des marqueurs de fin de ligne `\n` pour une meilleur lisibilité de la légende.

5. Pour déterminer la droite de régression de x en y, il suffit d'intervertir les arguments de la fonction `Regression` puis de calculer les coefficients a' et b'. Le code suivant convient pour comparer les deux approches.

```
alpha,beta,Ry=Regression(listeY,listeX)
ap=1/alpha
bp=-beta/alpha
Yp=[ap*x+bp for x in X]
# affichage
plt.figure()
plt.plot(listeX,listeY,'or',label="Données")
plt.plot(X,Y,'-b',label="Régression de y en x")
plt.plot(X,Yp,'-k',label="Régression de x en y")
plt.xlabel(r'x')
plt.ylabel(r'y')
plt.legend(loc=2)
plt.grid()
plt.show()
```

On peut observer sur la figure que les deux droites ne sont pas superposées mais se croisent au niveau de l'isobarycentre. Les deux approximations ne sont donc pas équivalentes.

Exercice 3.4 (Ajustements « de tableurs »)

1. Pour définir la fonction `Ajustements`, il suffit de commencer par faire un test sur le type de loi `"lin"`, `"log"`, `"exp"` ou `"pow"`, puis de définir les éventuels changements de variables sur les listes avant d'appeler la fonction `Régression` et enfin de renvoyer les paramètres attendus, si besoin après application de la fonction exponentielle. C'est ce que permet de réaliser le code suivant :

```
def Ajustements(X,Y,type):
    if type=="lin":
        a,b,R=Regression(X,Y)
    elif type=="log":
        Xp=[ma.log(x) for x in X]
        a,b,R=Regression(Xp,Y)
    elif type=="exp":
        Yp=[ma.log(y) for y in Y]
        ap,bp,R=Regression(X,Yp)
        a=ma.exp(ap)
        b=ma.exp(bp)
    elif type=="pow":
        Xp=[ma.log(x) for x in X]
        Yp=[ma.log(y) for y in Y]
        a,bp,R=Regression(Xp,Yp)
        b=ma.exp(bp)
    return(a,b,R)
```

2. Pour définir la procédure `Affichage` de la façon la plus compacte possible, il est nécessaire de voir que seuls la liste d'ordonnées `Ya`, le nom du type d'ajustement `typeAj` et l'équation `equation` varient et qu'il est nécessaire de les définir pour chaque type. Le reste peut être commun. En s'inspirant de ce qui a été fait à la question 5 de l'exercice 1, il vient alors le code :

```
def Affichage(X,Y,type):
    a,b,R=Ajustements(X,Y,type)
    Xa=Discretisation(min(X),max(X),100)
    if type=="lin":
        Ya=[a*x+b for x in Xa]
        typeAj="affine"
        equation='y='+str(round(a,4))+'*x+'+str(round(b,4))
    elif type=="log":
        Ya=[a*ma.log(x)+b for x in Xa]
        typeAj="logarithmique"
        equation='y='+str(round(a,4))+'*ln(x)+'+str(round(b,4))
    elif type=="exp":
        Ya=[b*a**x for x in Xa]
        typeAj="exponentiel"
        equation='y='+str(round(b,4))+'*'+str(round(a,4))+'^x'
    elif type=="pow":
        Ya=[b*x**a for x in Xa]
        typeAj="puissance"
        equation='y='+str(round(b,4))+'*x^'+str(round(a,4))
    # affichage
    plt.figure()
    plt.plot(X,Y,'or',label="Données")
    plt.plot(Xa,Ya,'-b',label="Ajustement"+typeAj+'\n'+equation \
        +'\nR^2='+str(round(R**2)))
    plt.xlabel('x')
    plt.ylabel('y')
    plt.legend(loc=2)
    plt.grid()
    plt.show()
```

3. Pour utiliser cette fonction pour chacune des 3 séries de données associée à un type $\alpha \in \{\text{lin, log, exp, pow}\}$ et contenues dans les fichiers $\alpha\beta$`.txt`, avec $\beta \in [\![1, 3]\!]$, il suffit d'écrire le code :

```
for t in ["lin","log","exp","pow"]:
    for i in range(1,4):
        X,Y=Lecture(t+str(i)+'.txt')
        Affichage(X,Y,t)
        plt.pause(2)
```

où l'argument `plt.pause(2)` permet de faire une pause de 2 secondes entre chaque affichage de figure. Après analyse des données, un `plt.close('all')` permet d'éviter 12 clics pour fermer les 12 figures. Dans tous les cas, on peut remarquer que plus les données sont bruitées, plus les ajustements paraissent de mauvaise qualité. En particulier, pour les ajustements de type puissance ou exponentiel, la densité des données pour les faibles valeurs rend caduque l'utilisation du modèle ajusté pour les abscisses les plus élevées.

* *
*