Feuille de TP Python 01

Révisions premiers algorithmes : Manipulation de listes, de `ndarray`; Statistiques descriptives

L'objet de ce TP est d'une part de revenir sur les algorithmes élémentaires de calculs de sommes, minima, maxima, etc... de liste et d'autre part d'y revoir les fondamentaux de statistique descriptive de première année. Le TP suivant sera consacré à la mise en place d'une méthodologie de traitement des données.

Pour démarrer, créez un *répertoire/dossier de travail* `td01` où vous voulez et repérez l'emplacement de ce répertoire. Ouvrez Spyder et dans l'éditeur, tapez (on peut ausi faire un copier-coller du .pdf vers l'éditeur, à condition de remettre les tabulations/indentations correctes, de gérer correctement les espacements et quelques autres détails)

```
# -*- coding: utf-8 -*-
"""
algos_base.py : algorithmes de somme, produit, minimum, maximum,
(On peut laisser les autres identifications mises par défaut par Spyder)
"""
#Zone (optionnelle) d'importation des modules externes
# ....

#Zone de définition des constantes et fonctions communes
# ....
```

et sauver ce fichier sous le nom `algos_base.py` dans le répertoire de travail `td01`, ce qui nomme le script présent dans l'éditeur. On y met la `docstring` (entre `"""`) décrivant ce que fait le script et des commentaires (débutant par `#`) signalant les différentes zones du script.

> Il est important de systématiquement documenter (`docstrings`) et commenter les scripts (`#`) ! !
> Autant prendre de bonnes habitudes !

1 Algorithmes à accumulateur

1.1 Sommes, produits

Pour calculer la somme des éléments d'une liste `L`, on se base sur une boucle et un « accumulateur » `S`. L'algorithme se décrit en mots de la façon suivante :

1. Initialiser l'accumulateur à 0
2. Boucler sur les éléments de la liste `L` et, dans la boucle, ajouter l'élément courant de la liste à l'accumulateur `S`.

Des codes Python typiques pour cela sont donc

```
#La liste L est composée de nombres, préexiste
S = 0 #Accumulateur pour la somme
for i in range(len(L)): #indices pour L
    S = S+L[i] #ou S+=L[i]
#A ce point S= la somme des éléments de L
```

```
S = 0
for elt in L: #Possibilité Pythonesque
    S += elt
```

```
S = 0
i = 0 #indices pour L
while i < len(L):
    S += L[i]
    i += 1 #incrementation de l'indice
```

```
S = 0
i = len(L) #indices pour L
while i > 0: #En décrémentant
    i = i-1 #décrementation de l'indice
    S += L[i]
```

Travail demandé

1. (a) Ecrire, dans le script `algos_base.py`, dans la zone des fonctions communes, une fonction `Somme(L)` prenant en argument une liste Python `L` de nombres et retournant la somme.
   (b) Ecrire dans la zone de script principal les lignes suivantes

```
liste_entiers = [2,1,4,3,6,7,5]
print("Test Somme de",liste_entiers,"=",Somme(liste_entiers))
```

(c) Exécuter le script, le résultat est-il correct ?

2. (a) Ecrire, dans le script `algos_base.py`, dans la zone des fonctions communes, une fonction

`Denombrement(L,mini=1,maxi=5)`

prenant en argument une liste Python `L` de nombres, deux bornes `mini` et `maxi`, passées en *arguments nommés avec valeurs par défaut* et retournant le nombre d'éléments de la liste `L` qui sont compris (au sens strict) entre `mini` et `maxi`.

(b) Ecrire dans la zone de script principal à la suite des lignes de code déja écrites

```
print("Test Denombrement entre 1 et 5 de",liste_entiers,"=",
      Denombrement(liste_entiers))
```

(c) Exécuter le script, le résultat est-il correct ?

3. (a) Ecrire, dans le script `algos_base.py`, dans la zone des fonctions communes, une fonction

`SommeFiltree(L,mini=1,maxi=5)`

prenant en argument une liste Python `L` de nombres, deux bornes `mini` et `maxi`, passées en arguments nommés avec valeurs par défaut et retournant la somme des éléments de la liste `L` qui sont compris (au sens strict) entre `mini` et `maxi`.

(b) Ecrire dans la zone de script principal à la suite des lignes de code déja écrites

```
print("Test SommeFiltree entre 1 et 5 de",liste_entiers,"=",
      SommeFiltree(liste_entiers))
```

(c) Exécuter le script, le résultat est-il correct ?

4. (a) Ecrire, dans le script `algos_base.py`, dans la zone des fonctions communes, une fonction

`SommeRiemann(f,a=0,b=1,n=10)`

prenant en argument une fonction Python `f` d'argument un nombre réel et retournant un nombre réel, deux bornes `a` et `b` et un entier `n`, passés en arguments nommés et retournant la somme de RIEMANN (à droite)

$$\Delta \sum_{k=0}^{n-1} f(a+k.\Delta) \text{ avec } \Delta = \frac{b-a}{n}$$

(b) Ecrire dans la zone de script principal à la suite des lignes de code déja écrites

```
def X2(x):
    """ X2(x): Retourne le carré de x """
    return x*x
def X3(x):
    """ X3(x): Retourne le cube de x """
    return x*x*x
print("Test SommeRiemann de X2 entre 0 et 1, n=10 =", SommeRiemann(X2))
print("Test SommeRiemann de X3 entre 1 et 2, n=100 =",
      SommeRiemann(X3,a=1,b=2,n=100))
```

(c) Exécuter le script, les résultats sont-ils raisonnables ?

5. (a) Adapter l'algorithme de somme pour calculer le produit des éléments d'une liste de nombres.

(b) Ecrire, dans le script `algos_base.py`, dans la zone des fonctions communes, une fonction `Produit(L)` prenant en argument une liste Python `L` de nombres et retournant le produit des éléments de la liste `L`.

(c) Ecrire dans la zone de script principal à la suite des lignes de code déja écrites

```
print("Test Produit de",liste_entiers,"=",Produit(liste_entiers))
```

(d) Exécuter le script, le résultat est-il correct ?

1.2 Minima, maxima, minimant, maximant

Pour calculer le minimum des éléments d'une liste de nombre `L`, on se base là encore sur une boucle et un « accumulateur » `m`. L'algorithme se décrit en mots de la façon suivante :

1. initialiser `m` l'accumulateur à l'un des éléments de la liste ;
2. boucler sur les éléments de la liste `L` et, dans la boucle, tester si l'élément courant de la liste est inférieur à l'accumulateur `m`, si c'est le cas, affecter cette valeur à l'accumulateur.

Des codes Python typiques pour cela sont donc

```
#La liste L est composée de nombres, préexiste
m = L[0] #Accumulateur pour le minimum
for i in range(len(L)): #indices pour L
    if L[i] < m:
        m = L[i]
#A ce point m = minimum des éléments de L
```

```
m = L[-1] #Accumulateur pour le minimum
i = len(L)#indices pour L
while i>0:
    i = i-1#On décrémente l'indice
    if L[i] <= m:
        m = L[i]
```

Travail demandé

1. (a) Ecrire, dans le script `algos_base.py`, dans la zone des fonctions communes, une fonction `Minimum(L)` retournant le minimum des éléments de la liste numériqueL.
   (b) Ecrire de même une fonction `Maximum(L)` retournant le maximum de `L`.
   (c) Ecrire dans la zone de script principal à la suite des lignes de code déja écrites

```
print("Test Minimum de",liste_entiers,"=",Minimum(liste_entiers))
print("Test Maximum de",liste_entiers,"=",Maximum(liste_entiers))
```

   (d) Exécuter le script, le résultat est-il correct ?
2. (a) Ecrire, dans le script `algos_base.py`, dans la zone des fonctions communes, une fonction `IndiceMinimum(L)` retournant le *premier* indice d'un élément minimum de la liste numériqueL.
   (b) Ecrire de même une fonction `IndiceMaximum(L)` retournant le *dernier* indice d'un élément maximum de la liste numériqueL.
   (c) Ecrire dans la zone de script principal à la suite des lignes de code déja écrites

```
print("Test IndiceMinimum de",liste_entiers,"=",IndiceMinimum(liste_entiers))
print("Test IndiceMaximum de",liste_entiers,"=",IndiceMaximum(liste_entiers))
```

   (d) Exécuter le script, le résultat est-il correct ?

2 `numpy` :Utilisation des tableaux et de fonctions prédéfinies

Vous devez connaitre les principes de calcul présentés précédemment. Cependant, dès que les calculs envisagés sont plus « volumineux », il peut être beaucoup plus efficace d'utiliser des fonctions ayant les mêmes fonctionnalités mais écrites par des professionnels, beaucoup plus rapides et efficaces. Ces fonctions « add-ons » du langage sont regroupées dans des *modules*.

Les modules que nous utiliserons principalement cette année sont

1. `numpy` qui effectue des calculs sur des vecteurs, matrices de grande taille et implémente les fonctions mathématiques de haut niveau (ln, exp,..., arcsin, la fonction d'erreur de GAUSS (primitive de $x \mapsto e^{-x^2}$),..
2. `matplotlib` et plus précisémment le sous-module `matplotlib.pyplot` pour les sorties graphiques.

Ces modules donnent à Python des fonctionnalités et une syntaxe très proches du logiciel de calcul scientifique professionnel MATLAB. L'utilisation d'un module par un script se se fait, en écrivant en tête de script des lignes d'importation.

```
import numpy as np #importe numpy avec l'alias np
import matplotlib.pyplot as plt #importe ce module avec l'alias plt
```

Il y a d'autres manières d'importer toutes les fonctions d'un module sans alias, une seule fonction,...à voir à l'usage.

L'efficacité des fonctions de ces modules externes provient des faits suivants :

— L'*interpréteur* Python « lit » les instructions décrites dans le script et, au fûr et à mesure, traduit ces instructions dans le langage compréhensible par le processeur de la machine. Lors d'une boucle, il effectue cette opération de traduction en même temps qu'il effectue les opérations elles-mêmes [1]. Traduire en simultané prend du temps et de l'énergie de calcul.

— La fonction `np.sum(L)` de `numpy` [2] comporte elle aussi une boucle mais *codée directement dans le langage machine*. Il n'y a donc pas de traduction simultanée et un gain de temps d'un facteur pouvant aller de 10 à 1000.

Les fonctions décrites dans la partie précédente ont pour *alter-ego* `numpy`

`Somme(L)`	`np.sum(L)` ou `L.sum()`	`Produit(L)`	`np.prod(L)` ou `L.prod()`
`Minimum(L)`	`np.min(L)` ou `L.min()`	`Maximum(L)`	`np.max(L)` ou `L.max()`

1. Le vrai processus n'est pas exactement celui-là mais est assez similaire
2. qui retourne la somme des éléments du `ndarray` (les vecteurs/matrices de `numpy`) `L`

Pour le travail demandé après, on rappelle les formules classiques du cours des statistiques descriptives. Si $x=(x_0,x_1,\ldots,x_n)$, $y=(y_0,y_1,\ldots,y_n)$ sont deux séries [3] statistiques, *i.e.* deux suites, de $n+1$ nombres alors,

1. en notant m_x la moyenne de x, on a
$$m_x=\frac{1}{n+1}\sum_{k=0}^{n}x_k,$$
2. en notant σ_x^2 la variance de x, on a
$$\sigma_x^2=\frac{1}{n+1}\sum_{k=0}^{n}x_k^2-m_x^2=\frac{1}{n+1}\sum_{k=0}^{n}(x_k-m_x)^2,$$
3. en notant s_{xy} la covariance de x et y, on a
$$s_{xy}=\frac{1}{n+1}\sum_{k=0}^{n}x_k.y_k-m_x.m_y=\frac{1}{n+1}\sum_{k=0}^{n}(x_k-m_x).(y_k-m_y),$$
4. en notant $y_r=a.x+b$ l'équation de la droite de régression de y sur x, alors $a=\frac{s_{xy}}{\sigma_x^2}$, $b=m_y-a.m_x$,
5. en notant μ_x la médiane de x alors, si n est pair, μ_x est la valeur d'indice $\frac{n}{2}$ du vecteur $\tilde{x}$ obtenu en ordonnant les valeurs de x par ordre croissant (Quelle est la convention en cas de nombre impair d'observations ?).

Travail demandé

Ouvrir une nouvelle fenêtre de l'éditeur et commencer un nouveau script que l'on nommera `stats-numpy.py` en tapant puis sauvant l'entête suivant

```
# -*- coding: utf-8 -*-
""" stats-numpy.py : essai de statistiques en Python/Numpy """
#Zone (optionnelle) d'importation des modules externes
import numpy as np #importe numpy avec l'alias np
import matplotlib.pyplot as plt #importe ce module avec l'alias plt

#Zone de définition des constantes et fonctions communes

#Script principal
x=np.linspace(0,1,100) # ndarray de 100 points equidistribués
                       # sur l'intervalle [0,1], bornes comprises
y=2*np.sin(2*np.pi*x) # ndarray de meme taille que x contenant
                      # les 2.sin(2\pi.x)
```

1. Sachant que `matplotlib.pyplot` est importé avec l'alias `plt`, que `x` est un `ndarray` unidimensionnel et que la fonction `plt.hist(x,density=True)` du module `matplotlib.pyplot` trace l'histogramme de la série statistique `x`, afficher l'histogramme de `y`. Tester ! ! (`plt.show()` montre le graphique produit.)
2. Sachant que `numpy` est importé avec l'alias `np`, que `x` est un `ndarray` et que la fonction `numpy` `np.mean(x)` ou `x.mean()` calcule la moyenne des éléments de `x`, écrire à la suite du script :
   (a) Les lignes Python imprimant la moyenne des éléments de `x`, `y` et plaçant ces quantités dans les variables `mx` et `my`. Ordonnancer de façon à ne pas faire deux fois le même calcul. Tester ! !
   (b) Les lignes Python imprimant les écart-types des éléments de `x`, `y` et plaçant ces quantités dans les variables `sx` et `sy`. Ordonnancer de façon à ne pas faire deux fois le même calcul. Tester ! !
   (c) Les lignes Python imprimant la covariance des deux listes `x`, `y` et plaçant cette quantité dans la variable `sxy`. Tester ! !
   (d) Les lignes Python imprimant les éléments a et b de la droite de régression de `y` sur `x`. Tester ! !
3. Sachant que `matplotlib.pyplot` est importé avec l'alias `plt`, que `x`, `y` sont des `ndarray` unidimensionnels de même taille et que la fonction `plt.plot(x,y,'x')` trace le nuage de points (x_i,y_i) dans la fenêtre graphique, écrire à la suite du script :
   (a) Les lignes Python graphant `y` en fonction de `x`. Tester ! ! (`plt.show()` montre le graphique produit.)
   (b) Les lignes Python calculant $z=a.x+b$, placé dans le `ndarray` `z` puis graphant `z` en fonction de `x`. Tester ! !

3. Attention ! une *série* au sens mathématique de ce terme, n'est pas exactement une suite de ce type, *cf.* cours de fin d'année.