Informatique : Devoir Surveillé (CORRECTION)

1 Déplacement d'un cavalier sur un échiquier

Un échiquier est un plateau avec 8 lignes et 8 colonnes. Ces lignes et ces colonnes seront dans cet exercice numérotées de 0 à 7. Une position sur un échiquier est un couple (i, j) d'entiers compris entre 0 et 7 inclus, avec i le numéro de la ligne et j le numéro de colonne.
Un cavalier placé sur l'échiquier se déplace en bougeant de 2 cases dans une direction (verticale ou horizontale) et de 1 case perpendiculairement. Le dessin ci-dessous à gauche illustre les 8 possibilités de déplacement d'un cavalier situé loin des bords de l'échiquier. Comme le cavalier ne peut pas sortir du plateau, lorsqu'il est près des bords, il a moins de possibilités de se déplacer, comme l'illustre le dessin ci-dessous à droite.
Les positions seront codées sous la forme d'un tuple de deux éléments : ainsi `p=(1,6)` *définit une position valide.*

Question 1
Écrire une fonction `Valide` *prenant en argument un tuple t de deux éléments et qui vérifie que celui-ci désigne bien une position de l'échiquier.* `Valide` *renvoie un booléen.*

Une position est valide si t[0] et t[1] sont des entiers compris entre 0 et 7 :

```
def Valide(t):
    return type(t[0])==int and type(t[1])==int and 0<=t[0]<=7 and 0<=t[1]<=7
```

Question 2
Écrire une fonction `CoupSuivant` *prenant en argument un tuple t désignant une position valide (il n'est pas nécessaire de vérifier ce point) et renvoyant la liste des positions valides que peut atteindre un cavalier placé en (i, j) avec i=t[0] et j=t[1] en un seul coup. Cette fonction renverra donc une liste de tuples de deux éléments.*

Il y a ici plusieurs façons de procéder. Le plus simple est de parcourir une tuple (ou une liste) de tuples (ou de listes) contenant les différents déplacements possibles et de vérifier leur validité à l'aide de la fonction précédente. Il est important de s'interroger un jour de concours sur la possibilité de réutiliser les fonctions codées précédemment, sur la structure des fonctions, et sur l'enchaînement logique des questions !

```
def CoupSuivant(t) :
    i, j, L = t[0], t[1], []
    D = ((-2,-1),(-2,1),(-1,-2),(-1,2),(1,-2),(1,2),(2,-1),(2,1))
    for dep in D :
        coup = (i + dep[0], j + dep[1])
        if Valide(coup) :
            L.append(coup)
    return L
```

L'objectif des questions suivantes est de trouver un algorithme qui permet de savoir quel est le nombre de coups minimal pour qu'un cavalier atteigne les positions de l'échiquier depuis une position initiale donnée.
Nous allons utiliser un algorithme récursif pour cela. Cet algorithme utilisera une matrice M, qui représente la meilleure distance (en terme de nombre de coups de cavalier) trouvée jusqu'à présent depuis la position initiale. On admet que toutes les positions sont atteignables en moins de 64 coups, ainsi on initialisera M avec des 65, pour toutes les positions sauf celle du départ. La matrice M pourra au choix être codée comme une liste de listes ou comme un numpy.array

Question 3
Écrire une fonction `Initialise` *prenant en argument deux entiers i0 et j0 tels que $(i0, j0)$ est une position et qui renvoie un tableau M de taille 8×8 tel que :*

$$(i, j) \neq (i0, j0), M[i][j] = 65 \text{ et } M[i0][j0] = 0$$

```
def Initialise(i0,j0):
    M=[[65]*8 for _ in range(8)]
    M[i0][j0] = 0
    return M
```

Question 4

Écrire une fonction récursive ***PositionsRecursives*** *d'arguments M, t et c où :*

- *M est la matrice des meilleurs distances provisoires,*
- *t est le tuple désignant la position en cours,*
- *c est le nombre de coups qui aura été nécessaire pour parvenir à t.*

La condition d'arrêt sera M[t[0]][t[1]] < c.

Comme les opérations sur les listes ou tableaux sont globales, il n'y a pas lieu de retourner M si la condition d'arrêt est satisfaite. Il faut par contre remettre à jour le nombre de coups nécessaires pour atteindre la position en cours, et explorer récursivement (parcours en largeur) les coups possibles en incrémentant c.

```
def PositionsRecursives(M, t, c):
    i, j = t[0], t[1]
    if M[i][j] >= c:
        M[i][j] = c
        Pos = CoupSuivant(t)
        for couple in Pos:
            PositionsRecursives(M, couple, c+1)
```

Question 5

À l'aide des fonctions précédentes, écrire une fonction ***Cavalier*** *prenant en argument une position t0 et renvoyant un tableau M de taille* 8×8 *tel que M[i][j] est le nombre minimum de coups nécessaires à un cavalier situé en (t0[0],t0[1]) pour arriver à la position* (i, j)

Les trois dernières questions ont en réalité fait l'objet d'une seule question au concours e3a, certes en option Info pour les MP. Il aurait fallu penser tout seul aux techniques de wrapping, séparant l'initialisation de la récursivité de la fonction récursive à proprement parler, techniques que nous avons étudiées à l'occasion des suites de Fibonacci entre autres. Cette question seule paraît quand même très délicate... Ici, il suffit de réunir les fonctions précédentes.

```
def Cavalier(i0, j0):
    A = Initialise(i0, j0)
    t = (i0, j0)
    PositionsRecursives(A, t, 0)
    return A
```

Par exemple :

```
In [38]: Cavalier(2,3)
Out[38]:
[[3, 4, 1, 2, 1, 4, 3, 2],
 [2, 1, 2, 3, 2, 1, 2, 3],
 [3, 2, 3, 0, 3, 2, 3, 2],
 [2, 1, 2, 3, 2, 1, 2, 3],
 [3, 4, 1, 2, 1, 4, 3, 2],
 [2, 3, 2, 3, 2, 3, 2, 3],
 [3, 2, 3, 2, 3, 2, 3, 4],
 [4, 3, 4, 3, 4, 3, 4, 3]]
```

2 Charge de voitures électriques

On s'intéresse à la charge de différents modèles de voitures électriques, tous alimentés par une batterie d'accumulateurs. Celle-ci se comporte comme un condensateur.
Une expérience consiste à relever l'intensité aux bornes de celui-ci à différents temps.
On rappelle que la charge q est donnée par :

$$q = \int_0^{t_f} i(t)\,dt$$

2.1 Intégration numérique

Question 6
Écrire une fonction **`Trapeze`** *d'arguments T et I, où T est une liste de temps triée dans l'ordre croissant et I une liste d'intensités correspondant aux différents temps de T, qui réalise le calcul de l'intégrale numérique de la fonction i(t) entre T[0] et T[len(T)-1] par la méthode des trapèzes. On rappelle que la méthode des trapèzes consiste à approcher l'intégrale d'une fonction par l'aire des trapèzes sous la courbe comme indiqué par la figure ci-dessous :*

Il suffit ici d'écrire correctement la formule de l'aire d'un trapèze et d'effectuer une sommation. Comme il s'agit pratiquement d'une question de cours sur le programme de première année, l'indulgence du correcteur n'est pas acquise... Notez que la boucle s'arrête à l'avant-dernière position de I et T, et que le pas d'intégration n'est pas nécessairement constant.

```
def Trapeze(T,I):
    somme = 0
    for i in range(len(T)-1) :
        somme += 0.5 * (I[i+1] + I[i]) * (T[i+1] - T[i])
    return somme
```

Question 7
Écrire une ligne de commande qui permet de calculer la charge totale pour la série de mesures suivante :

temps(s)	*intensité(A)*
0	*0*
2	*0.01*
8	*0.03*
11	*0.025*
18	*0.025*
20	*0.01*
26	*0*

```
Trapeze([0,2,8,11,18,20,26],[0,0.01,0.03,0.025,0.025,0.01,0])
```

2.2 Fichier externe

Dans le cas d'un grand nombre de mesures, rentrer les données à la main est trop fastidieux. Il est possible de lire les valeurs mesurées dans un fichier texte.
Le stockage dans le fichier texte contient les différentes valeurs numériques sous forme d'une unique colonne. Celle-ci contient :

- *le nombre de mesures n sur la première ligne*
- *l'instant t1,*
- *l'intensité i1,*
- *...*
- *l'instant tn,*
- *l'intensité in*

Question 8

On supposera dans cette question que les temps sont rangés dans l'ordre croissant. En utilisant la documentation fournie en annexe et la fonction ***Trapeze****, écrire la fonction* ***Int_fichier(s)*** *de paramètre s, qui est une chaîne de caractères contenant le nom du fichier où sont stockés les mesures, et qui renvoie la charge totale calculée à l'aide de la méthode des trapèzes pour les données stockées dans ce fichier.*

Nous allons créer deux fonctions distinctes, afin de pouvoir les réutiliser dans la question 10. La première fonction servira à lire le fichier et renverra une liste de temps et une liste d'intensités. La seconde fonction appellera la première fonction et calculera la charge totale.

```
def lire_fichier(s):
     fichier = open(s, 'r')
     n = int(fichier.readline())
     T,I = [], []
     for i in range(n):
          T.append(float(fichier.readline()[:-1]))
          I.append(float(fichier.readline()[:-1]))
     fichier.close()
     return T,I

def Int_fichier(s):
     T,I = lire_fichier(s)
     return Trapeze(T,I)
```

On obtient alors :

```
In [62]: lire_fichier('mesures.txt')
Out[62]:
([0.0, 2.0, 8.0, 11.0, 18.0, 20.0, 26.0],
 [0.0, 0.01, 0.03, 0.025, 0.025, 0.01, 0.0])

In [63]: Int_fichier('mesures.txt')
Out[73]: 0.4525000000000001
```

Question 9

On souhaite faire une représentation graphique l'intensité en fonction du temps. Dans cette question, on ne supposera plus que les données ne sont pas triées par temps croissant dans le fichier précédent.
Écrire une fonction ***Tri(T,I)*** *de paramètres T et I qui sont deux listes de même longueur, qui réalise le tri de la liste T par ordre croissant, en prenant garde que les temps T[i] associées aux intensités I[i] doivent le rester après que la liste T ait été triée. On pourra utiliser n'importe quel algorithme de tri vu dans le cours, en l'adaptant au problème qui se pose ici, mais il n'est pas question d'utiliser des fonctions préprogrammées d'une bibliothèque ou du langage Python. Cette fonction renverra deux listes TT et IT.*

Le choix du tri n'est pas anodin. Utiliser un tri récursif serait ici plus délicat à écrire. Comme la question n'impose pas d'obtenir une bonne complexité, nous allons nous contenter d'implémenter un tri par insertion. Lorsque deux éléments de la liste T seront permutés, nous en ferons de même sur la liste I.

```
def Tri(T, I) :
    for p in range(1, len(T)) :
        i = p
        while i > 0 :
            if T[i] < T[i-1]:
                T[i],T[i-1] = T[i-1],T[i]
                I[i],I[i-1] = I[i-1],I[i]
                i -= 1
            else:
                i = 0
```

Question 10

Écrire alors une fonction ***Graphe(s)****, de paramètre s, qui est une chaîne de caractères contenant le nom du fichier où sont stockés les mesures, et qui réalise la représentation graphique de l'intensité en fonction du temps.*

Nous supposerons dans cette question que nous avons effectivement choisi d'écrire deux fonctions distinctes à la question 8, de sorte à ne pas avoir à implémenter la lecture du fichier à nouveau.

```
import matplotlib.pyplot as plt
def Graphe(s):
    T,I = lire_fichier(s)
    Tri(T,I)
    plt.plot(T,I)
    plt.show()
```

3 Diffusion thermique

L'objectif de ce problème est de faire une étude informatique de certains aspects liés à la diffusion thermique. On considérera l'évolution de la température en fonction du temps pour un fil conducteur unidimensionnel : la température de ce fil sera donc une fonction de deux variables : $T(z,t)$

3.1 Méthode des moindres carrés pour déterminer la température initiale

Une étude théorique a permis d'affirmer que la température à l'instant initial pour le fil unidimensionnel est de la forme $f(z) = A + B\exp(-10z)$ avec A et B deux paramètres à déterminer.
L'objectif de cette partie est de trouver les paramètres A et B qui correspondent le mieux à des valeurs expérimentales mesurées. Nous allons procéder à un ajustement par la méthode des moindres carrés, c'est à dire trouver les paramètres A et B qui minimisent pour des données mesurées (z_i, T_i) l'erreur quadratique commise en remplaçant les (z_i, T_i) par les $(z_i, f(z_i))$.
Il s'agit donc de minimiser la quantité suivante :

$$E(A,B) = \sum_{i=0}^{i=n-1} \left(T_i - (A + B\exp(-10z_i))\right)^2$$

L'algorithme de Gauss-Newton fournit une solution à ce problème. C'est un algorithme itératif, à savoir qu'il va partir d'une solution approchée rentrée par l'utilisateur, et l'améliorer à chaque itération.
Dans le détail, on note $E_0 = \begin{pmatrix} A_0 \\ B_0 \end{pmatrix}$ l'estimation initiale des paramètres A et B.
On a alors la relation de récurrence suivante :

$$E_{s+1} = E_s - \left(J_r^T J_r\right)^{-1} J_r^T R \text{ où } r_i = \left(T_i - (A + B\exp(-10z_i))\right)^2 \text{ et } R = \begin{pmatrix} r_0 \\ \dots \\ r_{n-1} \end{pmatrix}$$

L'exposant T désigne la transposée et où J_r est la matrice jacobienne $n \times 2$ des dérivées de r_i par rapport à A et B :

$$J_r = \begin{pmatrix} \frac{\partial r_0}{\partial A} & \frac{\partial r_0}{\partial B} \\ \dots & \dots \\ \frac{\partial r_{n-1}}{\partial A} & \frac{\partial r_{n-1}}{\partial B} \end{pmatrix}$$

Question 11
*Écrire une fonction **`residu(Z,T,a,b)`** qui prend en arguments deux listes Z et T, contenant les données mesurées, et deux valeurs a et b contenant les estimations de A et B et renvoie le vecteur colonne (numérique) R.*

```
import numpy as np

def residu(Z,T,a,b):
    R = []
    for i in range(len(Z)):
        R.append( [ ( T[i] - ( a + b * np.exp(-10*Z[i]) ) )**2 ] )
    return R
```

Question 12

Vérifier que la matrice J_r peut s'écrire :

$$J_r = \begin{pmatrix} -2(T_0 - a - b\exp(-10z_0)) & -2\exp(-10z_0)(T_0 - a - b\exp(-10z_0)) \\ \dots & \dots \\ -2(T_i - a - b\exp(-10z_i)) & -2\exp(-10z_i)(T_i - a - b\exp(-10z_i)) \\ \dots & \dots \\ -2(T_{n-1} - a - b\exp(-10z_{n-1})) & -2\exp(-10z_{n-1})(T_{n-1} - a - b\exp(-10z_{n-1})) \end{pmatrix}$$

Il suffit de calculer les dérivées partielles de r_i par rapport à a et à b.

Question 13

Écrire une fonction ***Jacobienne(Z,T,a,b)*** *qui prend en arguments deux listes Z et T, contenant les données mesurées, et deux valeurs a et b contenant les estimations de A et B et qui renvoie la matrice (numérique) J_r évaluée en (a,b)*

```
def jacobienne(Z, T, a, b):
    n = len(Z)
    J = np.zeros((n, 2))
    for i in range(n):
        expo = np.exp(-10*Z[i])
        coeff =  -2* (T[i] - a - b*expo)
        J[i] = [coeff, expo * coeff]
    return J
```

Question 14

Écrire une fonction ***transitionGN(Z,T,E)*** *qui prend en argument les listes Z et T et un vecteur colonne E 1×2 constitué des estimations de A et B et qui renvoie un autre vecteur colonne, correspondant aux nouvelles estimations de A et B, à l'aide de la méthode de Gauss-Newton*

Le produit de matrices à l'aide de numpy s'écrit **np.dot**. Il est important de retenir ce point. La transposée s'obtient à l'aide de **np.transpose**. Il était bien sûr possible de la redéfinir à la main.

```
def transition(Z, T, E):
     a,b = E[0][0], E[1][0]
     Jr = jacobienne(Z, T, a, b)
     R = np.array(residu(Z, T, a, b))
     return E - np.dot(
         np.linalg.inv( np.dot(np.transpose(Jr), Jr) ),
         np.dot(np.transpose(Jr), R)
     )
```

Question 15

On décide de réitérer la méthode de Gauss-Newton tant que $\|E_{s+1} - E_s\| > 10^{-6}$.
Implémenter une fonction ***solveGN(Z,T,E0)*** *qui prend en argument les listes Z et T, et un vecteur colonne E0 contenant l'estimation initiale de A et B et qui renvoie la solution approchée calculée à l'aide de la méthode de Gauss-Newton.*

Il est ici possible de définir la norme à la main, ou d'utiliser directement **np.linalg.norm**. L'algorithme est itéré dans une boucle conditionnelle.

```
def solveGN(Z, T, E0):
     U = transition(Z, T, E0)
     while np.linalg.norm(U-E0) > 10**(-6) :
          E0 = U
          U = transition(Z,T,E0)
     return U
```

Question 16

Une première série de mesures expérimentales nous a donné les valeurs suivantes :

$z(m)$	0	0,05	0,1	0,15	0,2	0,25	0,3	0,35	0,4	0,45	0,5
$T(z,0)$ (K)	400	360	337	322	313	308	305	303	302	301	300

Écrire une ligne de commande qui permet d'obtenir des valeurs approchées de A et de B à l'aide de la méthode de Gauss-Newton avec les estimations initiales a_0=400 et b_0 = 50

```
Z = [0,0.05,0.1,0.15,0.2,0.25,0.3,0.35,0.4,0.45,0.5]
T = [400,360,337,322,313,308,305,303,302,301,300]
solveGN(Z, T, [[400],[50]])
```

4 Tris par paquets

De manière plus détaillée dans le cadre choisi, le tri par paquets prend en entrée un tableau T de n nombres dans [0,1[. Ensuite :

1. *on crée n listes L_0, ..., L_{n-1} appelées paquets. Le paquet L_k est associé à l'intervalle $I_k = \left[\frac{k}{n}, \frac{k+1}{n}\right[$*
2. *Pour chaque élément $T[i]$:*
   - *calculer l'intervalle I_k dans lequel il se trouve.*
   - *ajouter $T[i]$ à L_k*
3. *Trier chaque liste L_k à l'aide de l'algorithme de tri auxiliaire (tri par fusion dans notre cas).*
4. *Renvoyer la concaténation des listes L_0, ..., L_{n-1}.*

Question 17

Proposer une implémentation du tri par fusion en Python. On pourra définir deux fonctions, comme vu dans le cours.

```
def interlac(xs, ys) :
    """ Version impérative de la fusion de deux listes """
    i, j, T = 0, 0, []
    while i < len(xs) and j < len(ys) :
        if xs[i] <= ys[j]:
            T.append(xs[i])
            i += 1
        else :
            T.append(ys[j])
            j += 1
    if i == len(xs) :
        T += ys[j:]
    else :
        T += xs[i:]
    return T

def interlac_r(xs, ys) :
    """ Version récursive de la fusion de deux listes """
    if xs == [] or ys == [] :
        return xs + ys
    if xs[0] < ys[0] :
        return [xs[0]] + interlac_r(xs[1:], ys)
    return [ys[0]] + interlac_r(xs, ys[1:])

def fusion(L) :
    """ Fonction principale """
    if len(L) >= 2 :
        i = len(L)//2
        return interlac( fusion(L[:i]), fusion(L[i:]) )
    else :
        return L
```

Question 18

Écrire une fonction qui effectue le tri par paquets.

Il est important de comprendre ici que les L_i sont individuellement des listes et que ces n listes doivent être gardées en mémoire : la structure choisie est une liste de listes.

```
def tri_paquet(L) :
    N = len(L)
    Tab = [[] for _ in range(N)]
    for nombre in L :
        no_paquet = int(nombre*N) # On calcule le numéro du paquet où le nombre est affecté
        Tab[no_paquet].append(nombre)
    for paquet in Tab :
        paquet = fusion(paquet) # Chaque paquet est trié
    Liste_triee = [] # On crée la liste finale
    for paquet in T:
        Liste_triee += paquet # Et les paquets sont fusionnés
    return Liste_triee
```

Question 19

Quelle est la complexité en mémoire du tri par paquets ? On admettra que le tri par fusion a une complexité en mémoire en $O(n)$

Les n listes crées lors du tri par paquets vont se voir affecter les n éléments de L : la phase de répartition par paquets a un coût mémoire en $O(n)$. Comme le tri par fusion a une complexité en mémoire qui est également en $O(n)$, le tri par paquets a un coût en mémoire en $O(n)$.

Question 20

Dans quel cas est-on dans le pire des cas en terme de complexité en temps ? (On ne vous demande pas de justifier cette affirmation). Quelle est la complexité dans le pire des cas du tri par paquets ?

Le pire des cas pour la complexité en temps pour le tri par paquets est lorsque tous les éléments de L sont dans le même paquet (quel qu'il soit). En effet, dans cette situation c'est la complexité en temps du tri auxiliaire (par fusion dans notre cas) qui domine : ici il s'agit d'un $O(n \log n)$. La phase de répartition a un coût total en $O(n)$ qui est négligeable devant celui du tri par fusion.

Question 21

Montrer que la complexité en temps dans le meilleur des cas du tri par paquets en $O(n)$. On pourra démontrer que le tri par paquets nécessite forcément $O(n)$ opérations élémentaires puis proposer un cas où cette complexité est atteinte.

Le tri par paquets nécessite obligatoirement n opérations élémentaires lors de sa phase d'affectations des nombres dans les paquets : en effet, on doit déterminer pour chaque nombre à quelle liste il appartient : un calcul par nombre est au minimum nécessaire et donc n calculs sont nécessaires au total.
Lorsque chaque nombre est dans une liste différente (il s'agit effectivement du meilleur des cas pour le tri par paquets mais il n'est pas nécessaire de le démontrer...), chacune des listes L_i est de longueur 1, ce qui est une condition terminale pour le tri auxiliaire. Ainsi le tri de chaque liste L_i ne coûte aucune opération, et la concaténation des L_i est en $O(n)$, donc au final le tri par paquets est en $O(n)$ dans ce cas.
Puisqu'on a au final besoin de $O(n)$ opérations et que dans ce cas cette complexité est atteinte, on peut dire que le tri par paquets est en $O(n)$ opérations dans le meilleur des cas (et que cette domination est optimale).