Concours blanc d'informatique

Devoir surveillé n° 2 du janvier 2015

Durée : 2 heures – documents interdits – calculatrices interdites

– Exercice 1 – et encore un !

On considère la méthode suivante :

```
def Mystere(L):
    ♯ étape 1 :
    m1 = L[0]
    m2 = L[0]
    for i in range(1,len(L)):
        if L[i]<m1: m1 = L[i]
        elif L[i]>m2: m2 = L[i]

    ♯ étape 2 :
    l = m2-m1+1
    T = [0 for i in range(l)]

    ♯ étape 3 :
    for i in range(len(L)): T[L[i]-m1] = T[L[i]-m1] + 1

    ♯ étape 4 :
    L1 = []
    for i in range(l):
        ♯
        ♯
        L1 = L1 + [m1+i for j in range(T[i])]

    return L1
```

1. Appliquez la méthode ci-dessus à la liste $L = [8, 4, 3, 6, 4, 4, 6, 9, 3]$, en détaillant les opérations.
2. Qu'effectue l'algorithme ? Justifiez sommairement votre réponse, instruction par instruction (par exemple en ajoutant des commentaires après les ♯ sur le sujet, sans oublier d'inscrire son nom, son prénom et sa classe sur la feuille !).
3. Déterminez la complexité de cet algorithme, en justifiant.
4. Discutez les avantages et les inconvénients de l'algorithme.

– Exercice 2 – méthode d'Euler implicite

Considérons une équation différentielle du premier ordre avec condition initiale (dit problème de Cauchy) :

$$y'(t) \;=\; f(t, y(t)), \quad t \in [a, b], \quad y(a) = y_0,$$

où f est une fonction de classe $\mathcal{C}^1$ de $I \times \mathbb{R}$ à valeurs dans $\mathbb{R}$.

Notons $(t_i)_{0 \leq i \leq n}$ une subdivision de $[a, b]$. On recherche une approximation y_i de $y(t_i)$, pour $0 \leq i \leq n$. On rapelle que la méthode d'Euler (explicite) consiste à utiliser l'approximation $y'(t_i) \approx \dfrac{y(t_{i+1}) - y(t_i)}{t_{i+1} - t_i}$ pour tout i.

1. Déterminer une relation de récurrence entre y_{i+1} et y_i, pour cette méthode.
2. Illustrer graphiquement le fonctionnement de la méthode d'Euler.

On peut en fait approcher la dérivée en t_{i+1} par la même relation : $y'(t_{i+1}) \approx \dfrac{y(t_{i+1}) - y(t_i)}{t_{i+1} - t_i}$ pour tout i. Ceci donne lieu à la méthode d'Euler implicite.

3. Illustrer graphiquement le fonctionnement de la méthode d'Euler implicite.
4. Déterminer une relation de récurrence reliant y_{i+1} et y_i, pour cette méthode. Que remarque-t-on ? Justifier le nom de cette méthode.

Lorsque la fonction f possède certaines propriétés, on peut se ramener dans la formule obtenue dans la question précédente à une relation de récurrence donnant y_{i+1} en fonction de y_i, c'est-à-dire de la forme $y_{i+1} = \varphi(y_i)$. La suite de l'exercice consiste à appliquer la méthode d'Euler implicite sur un exemple permettant cette simplification.

Considérons l'équation différentielle $y' = \alpha y$ sur le segment $[a, b]$ avec condition initiale $y(a) = y_0$. Supposons le pas h de la subdivision constant.

5. Résoudre (mathématiquement) cette équation différentielle avec condition initiale.
6. Déterminer le pas h et le point t_i, en fonction de a, b et n.
7. Écrire explicitement y_{i+1} en fonction de y_i, puis y_n en fonction de y_0.
8. Quelle est la limite de la suite $(y_n)_n$ ainsi obtenue ? Conclusion ?
9. Écrire un programme en `python` mettant en œuvre la formule trouvée à la question précédente, pour le problème de Cauchy particulier $y' = \alpha y$, $y(a) = y_0$ sur $[a, b]$.
   Plus précisément, on mettra au point un programme :
   – utilisant a, b, α, y_0 et n,
   – qui calculera les (t_i, y_i),
   – qui les affichera,
   – et qui affichera également la solution exacte déterminée dans la question 5.

– Exercice 3 – une classe de rationnels

1. À l'aide des propriétés :
   - pour tous entiers a et b, on a $\text{pgcd}(a, b) = \text{pgcd}(a - b, b)$;
   - pour tout entier a, on a $\text{pgcd}(a, 0) = a$;

   écrire une fonction récursive `pgcd` retournant le PGCD de deux entiers.
   J'espère que ceci vous rappelle le TD n°2, exercice 4...
2. Construire une classe `rationnel` avec : constructeur, afficheur, addition et produit.
   - Le constructeur simplifiera la fraction.
   - Le constructeur s'arrangera pour que le signe porte sur le numérateur.
   - Attention, l'algorithme de pgcd fourni en question 1. ne s'applique qu'aux entiers naturels !
3. Surcharger les opérations d'addition et de produit afin de pouvoir écrire `rationnel(3,4)+rationnel(2,3)`.

Bon courage !

Devoir surveillé n° 2 – éléments de correction

– Exercice 1 –

1. Appliquez la méthode ci-dessus à la liste $L = [8, 4, 3, 6, 4, 4, 6, 9, 3]$, en détaillant les opérations.

   L'algorithme commence par déterminer le min et le max. On donne le résultat sans détailler : $m1 = 3$ (le min), $m2 = 9$ (le max), $\ell := 9 - 3 + 1 = 7$ et $T = [0, 0, 0, 0, 0, 0, 0]$. Passons sans plus attendre à la phase de remplissage.

élément correspondant dans L :		3	4	5	6	7	8	9	
départ	$T = [$	0,	0,	0,	0,	0,	0,	0	]
$i = 1$	$T = [$	0,	0,	0,	0,	0,	1,	0	]
$i = 2$	$T = [$	0,	1,	0,	0,	0,	1,	0	]
$i = 3$	$T = [$	1,	1,	0,	0,	0,	1,	0	]
$i = 4$	$T = [$	1,	1,	0,	1,	0,	1,	0	]
$i = 5$	$T = [$	1,	2,	0,	1,	0,	1,	0	]
$i = 6$	$T = [$	1,	3,	0,	1,	0,	1,	0	]
$i = 7$	$T = [$	1,	3,	0,	2,	0,	1,	0	]
$i = 8$	$T = [$	1,	3,	0,	2,	0,	1,	1	]
$i = 9$	$T = [$	2,	3,	0,	2,	0,	1,	1	]

   On sait que L contient 2 fois 3, 3 fois 4, aucun 5, ...

		$L = [$	8,	4,	3,	6,	4,	4,	6,	9,	3]
$i = 1$	$j = 1$	$L = [$	$\underline{3}$,	4,	3,	6,	4,	4,	6,	9,	3]
	$j = 2$	$L = [$	3,	$\underline{3}$,	3,	6,	4,	4,	6,	9,	3]
$i = 2$	$j = 3$	$L = [$	3,	3,	$\underline{4}$,	6,	4,	4,	6,	9,	3]
	$j = 4$	$L = [$	3,	3,	4,	$\underline{4}$,	4,	4,	6,	9,	3]
	$j = 5$	$L = [$	3,	3,	4,	4,	$\underline{4}$,	4,	6,	9,	3]
$i = 3$	$j = 5$	$L = [$	3,	3,	4,	4,	4,	4,	6,	9,	3]
$i = 4$	$j = 6$	$L = [$	3,	3,	4,	4,	4,	$\underline{6}$,	6,	9,	3]
	$j = 7$	$L = [$	3,	3,	4,	4,	4,	6,	$\underline{6}$,	9,	3]
$i = 5$	$j = 7$	$L = [$	3,	3,	4,	4,	4,	6,	6,	9,	3]
$i = 6$	$j = 8$	$L = [$	3,	3,	4,	4,	4,	6,	6,	$\underline{8}$,	3]
$i = 7$	$j = 9$	$L = [$	3,	3,	4,	4,	4,	6,	6,	8,	$\underline{9}$]

2. Qu'effectue l'algorithme ? Justifiez sommairement votre réponse, instruction par instruction.

   Cet algorithme trie la liste L, en comptant le nombre d'occurences de chaque élément de L.

   Étape 1 : détermination du min et du max de L.

   Étape 2 : construction et initialisation du tableau des occurences des éléments de L.

   Étape 3 : détermination des occurences des éléments de L. En d'autres termes on compte combien de fois apparaît min, combien de fois min+1, ..., combien de fois max.

   Étape 4 : construction de la liste triée. La première valeur de L apparaît T[1] fois, la seconde T[2]...

   Enfin on renvoie la liste triée.

3. Déterminez la complexité de cet algorithme, en justifiant.

   La recherche du minimum et du maximum s'effectue en $O(n)$ (on peut même modifier le programme pour effectuer ceci en $3E(n/2)$ opérations, exercice !).

La construction de T et sa mise à zéro nécessite ℓ affectations, donc une complexité en $O(\ell)$.

Le remplissage de T suppose de parcourir A, donc une complexité en $O(n)$.

Le vidage de T demande de parcourir T, d'où une complexité en $O(\ell)$.

On a finalement une complexité en $O(n+\ell)$, où n est le nombre d'éléments de A et ℓ est la différence entre le plus grand et le plus petit élément de A.

4. Discutez les avantages et les inconvénients de l'algorithme.

   Ce tri, dit « tri par casiers », a le bon goût d'être linéaire.

   Il est extrêmement performant pour les tableaux contenant peu de valeurs différentes, toutes dans un intervalle restreint.

   Par contre, il n'est réellement efficace que lorsque les valeurs du tableau sont dans un intervalle assez restreint par rapport à leur nombre ($m \ll n$). En outre ce tri a besoin de la création d'un tableau intermédiaire contenant les occurences des éléments de L, d'où une allocation de mémoire en fonction de l'écart entre le min et le max de L (même si L contient peu d'éléments différents).

   L'algorithme serait catastrophique sur le tableau à deux éléments $[-10^6, 10^6]$, pourtant déjà trié. L'algorithme serait particulièrement intéressant pour le tri de la population d'un pays en fonction de l'année de naissance (des millions d'éléments à trier, une centaine de valeurs différentes).

– Exercice 2 –

Considérons une équation différentielle du premier ordre avec condition initiale (dit problème de Cauchy) :

$$y'(t) \;=\; f(t, y(t)), \quad t \in [a, b], \quad y(a) = y_0,$$

où f est une fonction de classe $\mathcal{C}^1$ de $I \times \mathbb{R}$ à valeurs dans $\mathbb{R}$.

Notons $(t_i)_{0 \leq i \leq n}$ une subdivision de $[a, b]$. On recherche une approximation y_i de $y(t_i)$, pour $0 \leq i \leq n$. On rapelle que la méthode d'Euler (explicite) consiste à utiliser l'approximation $y'(t_i) \approx \dfrac{y(t_{i+1}) - y(t_i)}{t_{i+1} - t_i}$ pour tout i.

1. Déterminer une relation de récurrence entre y_{i+1} et y_i, pour cette méthode.

   Sans détour : $y_{i+1} = y_i + (t_{i+1} - t_i) f(t_i, y_i)$.

2. Illustrer graphiquement le fonctionnement de la méthode d'Euler.

   `Cf. cours.`

On peut en fait approcher la dérivée en t_{i+1} par la même relation : $y'(t_{i+1}) \approx \dfrac{y(t_{i+1}) - y(t_i)}{t_{i+1} - t_i}$ pour tout i. Ceci donne lieu à la méthode d'Euler implicite.

3. Illustrer graphiquement le fonctionnement de la méthode d'Euler implicite.

   Un joli dessin.

4. Déterminer une relation de récurrence reliant y_{i+1} et y_i, pour cette méthode. Que remarque-t-on ? Justifier le nom de cette méthode.

   On a $y_{i+1} = y_i + h f(t_{i+1}, y_{i+1})$: le terme y_{i+1} intervient dans les deux membres ! On n'a pas explicitement y_{i+1} en fonction de y_i, mais seulement une relation implicite entre ces deux approximations.

Lorsque la fonction f possède certaines propriétés, on peut se ramener dans la formule obtenue dans la question précédente à une relation de récurrence donnant y_{i+1} en fonction de y_i, c'est-à-dire de la forme $y_{i+1} = \varphi(y_i)$. La suite de l'exercice consiste à appliquer la méthode d'Euler implicite sur un exemple permettant cette simplification.

Considérons l'équation différentielle $y' = \alpha y$ sur le segment $[a, b]$ avec condition initiale $y(a) = y_0$. Supposons le pas h de la subdivision constant.

5. Résoudre (mathématiquement) cette équation différentielle avec condition initiale.

   `Sans détour :` $y(t) = y_0 \times e^{\alpha(t-a)}$ `pour tout` $t \in [a, b]$`.`

6. Déterminer le pas h et le point t_i, en fonction de a, b et n.
   Il vient $h = \dfrac{b-a}{n}$ et $t_i = a + ih = a + i\dfrac{b-a}{n}$ pour $i \in [\![0, n]\!]$.
7. Écrire explicitement y_{i+1} en fonction de y_i, puis y_n en fonction de y_0.
   Fixons i. Alors : $y_{i+1} = y_i + h\alpha y_{i+1}$ si et seulement si $(1-\alpha h)y_{i+1} = y_i$, soit $y_{i+1} = \dfrac{1}{1-\alpha h} y_i$.
   Par récurrence immédiate : $y_n = \dfrac{1}{(1-\alpha h)^n} y_0$ donc $y_n = \left(1 - \dfrac{\alpha(b-a)}{n}\right)^{-n} y_0$.
8. Quelle est la limite de la suite $(y_n)_n$ ainsi obtenue ? Conclusion ?
   Sans détour : la limite est $y_0 e^{\alpha(b-a)}$. Cette limite permet d'affirmer que plus le pas h diminue, meilleure est l'approximation y_n de $y(b)$.
   On dit que la méthode est consistante.
9. Écrire un programme en python mettant en œuvre la formule trouvée à la question précédente, pour le problème de Cauchy particulier $y' = \alpha y$, $y(a) = y_0$ sur $[a, b]$.

```
import numpy as np
import matplotlib.pyplot as plt
plt.clf()
def EulerImplicite(a,b,alpha,y0,n):
    h = (b-a) / n
    x = [a]
    y = [y0]
    for i in range(1,n):
        xi = x[i-1] + h
        x = x + [xi]
        yi = y[i-1] / (1 - alpha * h)
        y = y + [yi]
    plt.plot(x,y,'bo',label = 'Euler implicite')
a = 0
b = 1
alpha = 1
y0 = 1
n = 20
EulerImplicite(a,b,alpha,y0,n)
x=np.linspace(a,b,n)
plt.plot(x,np.exp(x),label='solution')
plt.legend(loc='upper left')
plt.show()
```

– Exercice 3 –

```
classrationnel:
     def __init__(self,numerateur,denominateur):
         if denominateur == 0:
            return "dénominateur nul !"
         if type(numerateur) != int or type(denominateur) != int:
            return "il faut des entiers !"
         p = pgcd(abs(numerateur),abs(denominateur))
         if denominateur>0:
            self.n = numerateur // p
            self.d = denominateur // p
         else:
               self.n = -numerateur // p
               self.d = -denominateur // p

     def __repr__(self):
         return str(self.n) + '/' + str(self.d)

     def __add__(self,r):
         return rationnel(self.n*r.d+self.d*r.n,self.d*r.d)

     def __mul__(self,r):
         return rationnel(self.n*r.n,self.d*r.d)

n1 = 10
d1 = 14
r1 = rationnel(n1,d1)
n2 = 33
d2 = -24
r2 = rationnel(n2,d2)
print("r1=",r1,"r2=",r2,"r1+r2=",str(r1+r2))
```