DS Informatique n°3 – Partie écrite. Durée : 1 heure

Lorsqu'on écrit un code Python : faire attention à ce que les indentations soient visibles sur la copie ; commenter le code de façon à expliquer les grandes étapes de l'algorithme en ajoutant un commentaire en fin de ligne de code après le symbole ♯.

Exercice 1

On considère la fonction suivante :

```
def algorithme(f,df,x0,n):
    x=float(x0)
    for i in range(1,n+1):
        x=x-f(x)/df(x)
    return(x)
```

1) (a) Déterminer en fonction de `n` la complexité de l'algorithme en supposant que les appels de `f` et `df` valent une opération chacun.

Solution.
- 2 opérations : `x=float(x0)`
- on répète n fois 5 opérations : `x=x-f(x)/df(x)`
- 1 opération `return`

La complexité est : $\boxed{5n+3 \text{ donc } O(n) \text{ donc linéaire.}}$

(b) Quel est le nom de l'algorithme que code la fonction `algorithme` ? A quoi sert-il ?

Solution. $\boxed{\text{C'est la méthode de Newton.}}$ Elle permet de déterminer une approximation d'une solution de l'équation $f(x) = 0$.

(c) Donner un avantage et un inconvénient de cette méthode.

Solution. Avantages : la convergence est très rapide (convergence quadratique) Inconvénient : il faut connaitre f' ; si f' s'annule, l'algorithme peut diverger ou boucler. On n'est pas sûr que le résultat renvoyé est bien une approximation d'une solution.

2) On va se servir de cette fonction pour calculer une approximation de $\sqrt{2}$. On prendra : $f(x) = x^2 - 2$ et $df(x) = 2x$.

(a) Donner en fonction de la valeur de `x0` le comportement de l'algorithme.

Solution.
- si `x0` > 0, l'algorithme va converger vers $\sqrt{2}$.
- si `x0` < 0, l'algorithme va converger vers $-\sqrt{2}$.
- si `x0` $= 0$, $f'(0) = 0$. L'algorithme s'arrête et renvoie un message d'erreur.

(b) Que renvoie `algorithme(f,df,1,0)`, `algorithme(f,df,1,1)`, `algorithme(f,df,1,2)` ? On donnera les résultats sous forme d'une fraction puis sous forme d'un nombre décimal avec 4 chiffres significatifs.

Solution. $x_0 = 1 = 1,000$ $x_1 = \dfrac{3}{2} = 1,500$

$x_2 = \dfrac{17}{12} = 1,416666 \approx 1,417$

3) On veut écrire un algorithme utilisant la méthode de la dichotomie avec comme critère d'arrêt un nombre donné d'itérations.

(a) Écrire en python une fonction `dichotomie` qui prend comme entrée la fonction à étudier, les deux extrémités a et b de l'intervalle de départ et un nombre d'itérations n et qui renvoie c approximation de la solution.

Solution.
```
def dichotomie(f,a,b,n):
for i in range(n):
c=(a+b)/2.
if f(a)*f(c)<0:
b=c
else:
a=c
return((a+b)/2.)
```

(b) Donner un avantage et un inconvénient à cette méthode.

Solution. Avantage : dès que $f(a)$ et $f(b)$ sont de signes opposés, l'algorithme converge toujours ; il n'y a pas à connaitre f'. Inconvénient : la converge est plus lente (convergence linéaire)

4) On reprend $f(x) = x^2 - 2$.

(a) Que renvoie `dichotomie(f,0,2,0)`, `dichotomie(f,0,2,1)`, `dichotomie(f,0,2,2)` ? On donnera les résultats sous forme d'une fraction puis sous forme d'un nombre décimal avec 4 chiffres significatifs.

Solution. `dichotomie(f,0,2,0)` renvoie $1 = 1,000$

`dichotomie(f,0,2,1)` renvoie $\frac{1}{2} = 1,500$

`dichotomie(f,0,2,2)` renvoie $\frac{5}{4} = 1,250$

(b) On donne comme valeur approximative de $\sqrt{2}$ à 11 chiffres significatifs :

$$\sqrt{2} \approx 1,4142135624$$

Pour un nombre d'itérations égal à 0 puis 1 puis 2, donnez le nombre de décimales exactes obtenues avec la fonction `algorithme` et avec la fonction `dichotomie`. Qu'en pensez-vous ?

Solution. Nombre de décimales exactes en fonction du nombre d'itérations :

Nombre d'itérations	avec `algorithme`	avec `dichotomie`
0	0	0
1	0	0
2	2	0

$\boxed{\text{La méthode de Newton est plus rapide}}$

Exercice 2 : pivot de Gauss

1) Soit A une matrice carrée de taille $n \times n$. Donner en français les étapes de l'algorithme du pivot de Gauss permettant de résoudre le système $Ab = c$. On admettra que le système est un système de Cramer.

Solution. Pour i allant de 1 à n(taille de A) :

- dans la colonne i, on cherche la plus grande valeur absolue parmi $a_{i,i}, \cdots, a_{n,i}$: ce sera la valeur du pivot. La ligne du pivot devient ligne de référence.
- on échange la ligne de référence avec la ligne i
- on divise la ligne de référence par la valeur du pivot
- grâce à la ligne de référence, on élimine les termes au-dessus et en dessous du pivot par des opérations de transvections : $L_k \longleftarrow L_k - A[k,i]L_i$

- on obtient la matrice identité
- on effectue dans le même ordre toutes ces opérations sur c. Le vecteur colonne obtenu est le résultat.

2) On considère une matrice carrée de taille n : $A = (a_{i,j})_{1 \leqslant i,j \leqslant n}$. Ecrire en python une fonction `ligneRef` qui prend comme entrée une matrice A et un numéro de ligne i et qui renvoie le numéro de la ligne où se trouve le terme le plus grand en valeur absolue entre $a_{i,i}, \cdots, a_{n,i}$.

   Solution.
   ```
   def ligneRef(A,i):
       n=len(A)
       kmax = i
       for k in range(i+1,n):
           if abs(A[k,i]) > abs(A[kmax,i])
               kmax = k
       return kmax
   ```

3) Le rang d'une matrice est donné par le nombre de pivots non nuls. Ecrire une fonction `rang` qui prend comme entrée une matrice C carrée de taille $n \times n$ et qui renvoie le rang de la matrice. (on pourra utiliser la fonction `ligneRef`)

   Solution. Voir le programme.
   Le principe : de la même façon que dans l'algorithme du pivot de Gauss, on échelonne la matrice. Si on rencontre un pivot nul, le rang diminue.

4) On considère la matrice suivante :
$$C = \begin{pmatrix} \frac{1}{2} & 1 & 1 \\ \frac{1}{3} & 1 & 0 \\ \frac{1}{6} & 0 & 1 \end{pmatrix}$$
   Avec python, `rang(C)` renvoie 3. Qu'en pensez-vous ?

   Solution. Si on calcule à la main la rang de la matrice, on trouve $\boxed{\mathrm{rg} C = 2.}$ On constate que $L_1 = L_2 + L_3$.
   Pourquoi python ne renvoie pas la bonne valeur : python travaille avec des valeurs approchées. Dans ce cas, $\dfrac{1}{2} - \dfrac{1}{3} - \dfrac{1}{6}$ est approximé par un nombre qui n'est pas nul.