TP n° 06 – Algorithmes dichotomiques

I Recherche d'un élément dans un tableau trié

La donnée est un tableau L de valeurs réelles **triées**. L'objectif est de trouver une valeur a dans ce tableau.

I.1 Première approche naïve

La première méthode est de parcourir le tableau terme à terme, en commençant par $L[0]$ jusqu'à tomber sur a.

Exercice 1. Ecrire une fonction `recherche_naive` qui prend comme entrée `L` et `a` et renvoie la position de `a` si a est dans le tableau et `False` sinon.

Solution 1.

```
def recherche_naive(L,a):
    i=0
    while L[i]<a:
                i=i+1
        if l[i]==a:
                return(i)
        else:
                return(False)
```

Exercice 2. Combien d'étapes sont nécessaires dans l'algorithme précédent ?

Solution 2. On a fait autant d'étapes que la position renvoyée.

I.2 Résolution par dichotomie

L'algorithme dichotomique permet une recherche de a plus rapide. Le principe en est le suivant :
m est l'indice du milieu du tableau L.

- si $L[m]$ vaut a, on renvoie m ;
- si $L[m] < a$, on recommence dans la moitié supérieure du tableau ;
- si $L[m] > a$, on recommence dans la moitié inférieure.

Exercice 3. A la main.

1. Dans le tableau de taille 15 suivant : $L = [-1, 0, 1, 3, 4, 5, 6, 8, 9, 10, 12, 13, 14, 15, 16]$, déterminer les étapes successives pour trouver la position de $a = 12$, par une recherche dichotomique.
2. Faites de même avec $L = [-2, -1, 0, 2, 3, 4, 8, 9]$ et $a = 8$.

Solution 3. On travaille à chaque fois sur des sous-tableaux de L. On notera d l'indice le plus petit du sous-tableaux et f le plus grand. A chaque étape, voici les valeurs les sous tableaux sur lesquels on travaille :

1. L, $d = 0, f = 14, m = 7, L[m] = 8$.
2. $[9, 10, 12, 13, 14, 15, 16]$, $d = 9, f = 14, m = 11, L[m] = 13$.
3. $[9, 10, 12]$, $d = 8, f = 10, m = 9, L[m] = 10$.
4. $[12]$, $d = 10, f = 10, m = 10, L[m] = 12$.

Pour la deuxième liste :

1. L $d = 0, f = 8, m = 3, L[m] = 2$.
2. $[3, 4, 8, 9]$, $d = 4, f = 8, m = 5, L[m] = 4$.
3. $[8, 9]$, $d = 6, f = 8, m = 6, L[m] = 8$.

Exercice 4. En reprenant le principe de la dichotomie, écrire une fonction `dicho` qui prend comme entrée `L` et `a` et renvoie la position de `a` si a est dans le tableau et `False` sinon.

Solution 4.

```
def dichotomie(L, a):
    debut = 0
    fin = len(L) - 1
    while debut <= fin:
        m = (debut+fin) // 2
        if L[m] == a:
            return m
        elif L[m] < a:
            debut = m + 1
        else:
            fin = m - 1
    return False
```

Remarque. Essayez sans, mais si vous avez des difficultés, reportez-vous au pseudo code en annexe.

Exercice 5. Vérifier que votre fonction est correcte : si i est l'indice renvoyé par `dicho`, que doit retourner `L[i]` ?

Solution 5. `L[dicho(L,a)]=a`

Exercice 6. Comparaison recherche naïve/dichotomie.
Modifier votre fonction `dicho` en une fonction `dicho_comparatif` qui prend comme entrée `L` et `a` et renvoie le nombre d'étapes qu'il a été nécessaire pour trouver la position de `a`.
Comparer sur des mêmes exemples le nombre d'étapes nécessaires pour la recherche naïve et pour la dichotomie.

Solution 6.

```
def dichotomie_comparatif(L, a):
    debut = 0
    fin = len(L) - 1
    compteur=0
    while debut <= fin:
        compteur=compteur+1
        m = (debut+fin) // 2
        if L[m] == a:
            return m,compteur
        elif L[m] < a:
            debut = m + 1
        else:
            fin = m - 1
    return False,compteur
```

Remarque. Complexité des deux algorithmes.
La complexité d'un algorithme donne une idée du nombres d'étapes nécessaires pour qu'un algorithme se termine.
Dans le cas de la recherche naïve, dans le pire des cas, on fera n étapes (si l'élément est en dernier). On dit que la complexité est linéaire, qu'elle est en $O(n)$.
Dans le cas de la recherche par dichotomie, on peut montrer que la complexité est en $O(\log(n))$: elle est beaucoup plus rapide.

II Recherche du zéro d'une fonction

Dans cette section, le tableau L contient des valeurs réelles qui représentent les valeurs prises par une fonction croissante. On cherche à trouver quand est-ce que la fonction s'annule.

Exercice 7. En utilisant le principe de la recherche par dichotomie, écrire une fonction `dicho_zero` qui prend comme entrée un tableau à valeurs triées L et qui renvoie l'indice i tel que $L[i]$ est la valeur la plus proche de zéro dans le tableau L.

Solution 7.

```
def dicho_zero(L):
    debut = 0
    fin = len(L) - 1
    while debut <= fin:
        m = (debut+fin) // 2
        if L[m] == 0:
            return m
        elif L[m] < 0:
            debut = m + 1
        else:
            fin = m - 1
    return(m)
```

Exercice 8. Testez la fonction précédente sur $L = [-2, -1, 2, 3]$. Proposer une façon de vérifier le résultat.

Solution 8. Si `dicho(L)` renvoie `i`, on peut afficher `L[i-1],L[i],L[i+1]`.

Exercice 9. On considère la fonction suivante : $f(x) = x + \exp(x)$ (qui est croissante).

1. Construire un tableau X qui contient 1000 réels équirépartis entre -1 et 1.
2. Construire un tableau L qui contient les valeurs de f pour chaque $x \in X$.
3. Avec la fonction `dicho_zero`, déterminer une valeur approchée de x tel que $f(x) = 0$.
4. A l'aide de la notice du concours ci-dessous, tracez le graphe de la fonction f.
5. Vérifiez graphiquement le résultat trouvé à la question 3.

Solution 9.

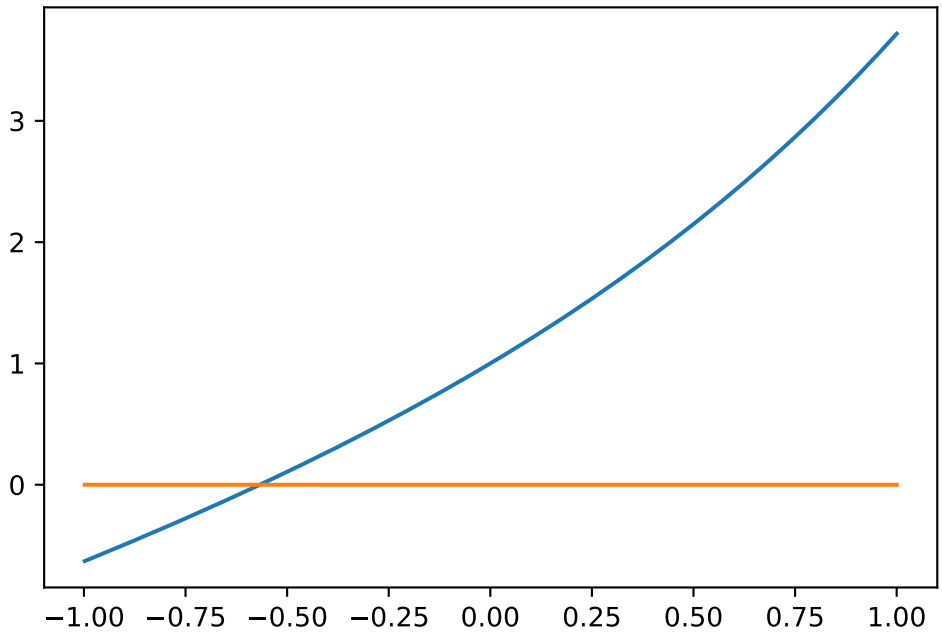
1.
```
X=[]
x=-1
for i in range(1000):
    x=x+2/1000.
    X.append(x)
```

2.
```
def f(x):
    return(exp(x)+x)

L=[]
for x in X:
    L.append(f(x))
```

3. `X[dicho_zero(L)]`.

4.
```
import matplotlib.pyplot as plt
plt.plot(X,Y)
plt.plot(X,[0]*len(X))
plt.show()
```



5.

```
import matplotlib.pyplot as plt   → charge le module pyplot sous le nom plt
plt.figure('titre')   → crée une fenêtre de tracé vide
plt.plot(LX,LY,'o-b')   → trace le graphique défini par les listes LX et LY (abscisses et ordonnées)
                          couleur : 'b' (blue), 'g' (green), 'r' (red), 'c' (cyan), 'm' (magenta), 'y' (yellow), 'k' (black)
                          type de ligne : '-' (trait plein), '--' (pointillé), '-.' (alterné)...
                          marque : 'o' (rond), 'h' (hexagone), '+' (plus), 'x' (croix), '*' (étoile)...
plt.xlim(xmin,xmax)   → fixe les bornes de l'axe x
plt.ylim(ymin,ymax)   → fixe les bornes de l'axe y
plt.axis('equal')   → change les limites des axes x et y pour un affichage avec des axes orthonormés (le tracé d'un cercle donne un cercle)
plt.show()   → affichage de la fenêtre
plt.savefig(fichier)   → sauve le tracé dans un fichier
                         (le suffixe du nom fichier peut donner le format ; par exemple, 'image.png')
```

FIGURE 1 – Documentation sur le tracé de fonctions

III Annexe

Algorithme 1 : Recherche par dichotomie.

Données : L une liste triée de n éléments et a un élément

```
1  debut ←premier indice de L ;
2  fin ←dernier indice de L ;
3  Tant que debut inférieur ou égal à fin
4      m ← quotient de fin+debut par 2
5      si L[m] == a
6          retourner m
7      ou si L[m] < a
8          debut←m+1
9      sinon
10         fin←m-1
11 retourner False
```