Algorithmes dichotomiques | 5

5.1 Introduction

Les méthodes de résolutions par un algorithme dichotomique font partie des algorithmes basés sur le principe de « diviser pour régner ». Elles utilisent la définition du terme **dichotomie** qui signifie diviser un tout en deux parties « opposées ». Certains algorithmes de tris sont basés sur ce principe de diviser pour régner.

Ce cours vous présente deux algorithmes dichotomiques :

- la recherche d'un élément dans une liste triée ;
- la détermination de la racine d'une fonction quand elle existe.

5.2 Recherche dichotomique dans une liste triée

Lorsque vous cherchez le mot « hippocampe » dans le dictionnaire, vous ne vous amusez pas à parcourir chaque page depuis la lettre a jusqu'à tomber sur le mot « hippocampe »...
Dans une liste triée, il y a plus efficace ! Par exemple dans le dictionnaire, vous ouvrez à peu près au milieu, et suivant si le mot trouvé est « inférieur » ou « supérieur » à « hippocampe » (pour l'ordre alphabétique), vous poursuivez votre recherche dans l'une ou l'autre moitié du dictionnaire.

Propriété –

On se donne une liste `L` de nombres de longueur `n`, triée dans l'ordre croissant, et un nombre `x0`.
Pour chercher `x0`, on va couper la liste en deux moitiés et chercher dans la moitié intéressante et ainsi de suite.
On appelle `g` l'indice de l'élément du début de la sous-liste dans laquelle on travaille et `d` l'indice de l'élément de fin.
Au début, `g = 0` et `d = n-1`
On souhaite construire un algorithme admettant l'invariant suivant :

si `x0` est dans `L` alors `x0` est dans la sous-liste `L[g:d]` (`g` inclus et `d` exclu).

On va utiliser la méthode suivante.

- On compare `x0` à « l'élément du milieu » : c'est `L[m]` où `m = (g+d)//2` son indice est `m` = `n//2` (division euclidienne)
- Si `x0 = L[m]`, on a trouvé `x0`, on peut alors s'arrêter.
- Si `x0` < `L[m]`, c'est qu'il faut chercher dans la première moitié de la liste, entre `L[g]` et `L[m-1]` (`L[m]` exclu).
- Si `x0` > `L[m]`, c'est qu'il faut chercher dans la seconde moitié de la liste, entre `L[m+1]` et `L[d]` (`L[m]` exclu).

On poursuit jusqu'à ce qu'on a trouvé `x0` ou lorsque l'on a épuisé la liste `L`.

5.2.1 Exemples d'application

Indiquer pour les deux exemples suivants les valeurs successives de `g` et `d` :

1. `x0 = 5` et `L` =

−3	5	7	10	11	14	17	21	30

2. `x0 = 11` et `L` =

−2	1	2	7	8	10	13	16	17

Cas 1 : $\begin{cases} g = 0 \\ d = 8 \\ m = 4, L[m] > x0 \end{cases}$ $\begin{cases} g = 0 \\ d = 3 \\ m = 1, L[m] = x0 \end{cases}$.
C'est fini, on a bien trouvé x_0 dans la liste.

Cas 2 : $\begin{cases} g = 0 \\ d = 8 \\ m = 4, L[m] < x0 \end{cases}$, $\begin{cases} g = 5 \\ d = 8 \\ m = 6, L[m] > x0 \end{cases}$ $\begin{cases} g = 5 \\ d = 5 \\ m = 5, L[m] < x0 \end{cases}$ $\begin{cases} g = 6 \\ d = 5 \end{cases}$.
C'est fini, on a épuisé la liste `L` et on n'a pas trouvé $x0$.

5.2.2 Implémentation en Python

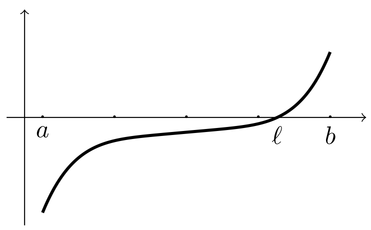
La fonction `recherche dichotomie` d'arguments une liste `L` et un élément `x0` renvoyant un booléen disant si `x0` est dans la liste `L` est proposée :

```python
def recherche_dichotomie(L:list, x0:int)-> bool:
    n = len(L)
    g = 0 # c'est l'indice de gauche
    d = n - 1 # c'est l'indice de droite
    rep = False
    while g <= d and rep == False :
        # si x0 est dans L alors L[g] <= x0 <= L[d]     {invariant}
        m = (g+d) // 2
        if x0 == L[m]:
            rep = True
        elif x0 < L[m]:
            d= m - 1
        else:
            g = m + 1
            # si x0 est dans L alors L[g] <= x0 <= L[d]     {invariant}
    return(rep)
```

Remarque : La terminaison de l'algorithme est obtenue avec $d - g$ qui est un entier positif qui décroît strictement à chaque passage dans la boucle **`while`** et joue le rôle de variant.

5.3 Détermination de la racine d'une fonction par dichotomie

5.3.1 Principe théorique de la méthode par dichotomie



On considère une fonction f vérifiant :

$$f \text{ continue sur } [a, b] \,;\, f(a) \text{ et } f(b) \text{ de signes opposés.}$$

Le théorème des valeurs intermédiaires nous assure que f possède au moins un zéro ℓ entre a et b. La preuve, vue en cours de mathématiques, repose sur la méthode de dichotomie. Prenons le cas $f(a) < 0$ et $f(b) > 0$ et posons $g_0 = a$, $d_0 = b$.

On considère $m_0 = \dfrac{g_0 + d_0}{2}$ et on évalue $f(m_0)$:

- Si $f(m_0) \geq 0$, on va poursuivre la recherche d'un zéro dans l'intervalle $[g_0, m_0]$
  On pose donc : $g_1 = g_0 \quad ; \quad d_1 = m_0$
- Sinon, la recherche doit se poursuivre dans l'intervalle $[m_0, d_0]$
  On pose donc : $g_1 = m_0 \quad ; \quad d_1 = d_0$

- On recommence alors en considérant $m_1 = \dfrac{g_1 + d_1}{2}$...

5.3.2 Implémentation en Python et avec scipy

Écrivons une fonction `zero_dichotomie(f:callable, a:float, b:float, epsilon:float) -> float` d'arguments une fonction `f`, des flottants `a` et `b` (tels que `a < b`), et la précision voulue `epsilon` (flottant strictement positif). Cette fonction renverra une valeur approchée à `epsilon` près d'un zéro de `f`, compris entre `a` et `b`, obtenue par la méthode de dichotomie.

```python
def zero_dichotomie(f:callable, a:float, b:float, epsilon:float):
    g = a # c'est un flottant
    d = b # c'est un flottant
    while d-g > 2*epsilon :
        m = (g + d) / 2
        if f(g)*f(m) <= 0:
            d = m
        else:
            g = m
    return ((g + d)/2)
```

Effectuons un test avec la fonction $f : x \mapsto x^2 - 2$ sur l'intervalle $[1, 2]$, avec une précision de 10^{-6} :

```python
def f(x):
     return(x ** 2 - 2)
print (zero_dichotomie(f, 1, 2, 10**(-6)))

# il s'affichera : 1.4142141342163086
```

Une telle fonction est déjà prédéfinie dans la bibliothèque `scipy.optimize`, la fonction `bisect`[1] :

```python
import scipy.optimize as spo
print (spo.bisect(f, 1, 2))
# il s'affichera : 1.4142135623724243
```

La précision est un argument optionnel (à mettre après `f`, `a` et `b`) et vaut 10^{-12} par défaut.

1: La méthode de dichotomie s'appelle aussi la méthode de la *bisection*.