TD 6 -
Dichotomie pour la :
Recherche d'éléments :
Recherche de racines

Informatique
MPSI - Lycée Thiers

2014/2015

Exercice 1

Exercice 1.

1. Ecrire une fonction `recherche(l,e)` prenant en paramètre un nombre `e` et une liste `l` et qui recherche si `e` apparaît ou non dans la liste `l`. La fonction retournera le booléen `True` ou `False`.
   - 1.1 en utilisant l'instruction `in`.
   - 1.2 sans utiliser l'instruction `in`.
2. Ecrire une fonction `dich_search()` qui recherche par dichotomie si un élément est présent dans une liste de nombres ordonnée dans le sens croissant.
   - 2.1 A l'aide d'un slicing.
   - 2.2 Sans utiliser de slicing. On s'inspirera du code suivant que l'on complétera :

```
def dich_search2(l,e):
    Imin, Imax = 0, len(l)-1
    while Imax - Imin >= 0:
    Imed = ...........
    if ..........:
        return True
    elif ..........:
        Imin = Imed + 1
    else:
        Imax = Imed - 1
    return ...........
```

Exercice 1

3 Tester le temps d'exécution des 4 algorithmes de recherche sur une liste aléatoire ordonnée de 100 000 nombres, grâce aux commandes suivantes (que l'on complétera) :

```
from random import random
L = [random() for k in range(100000)]    # Liste aléatoire
L.sort()    # Tri de la liste

from time import clock
a = clock()
recherche(L,0)
b = clock()
print(b-a, 'secondes')    # Temps d'exécution
# et de même avec les 3 autres fonctions de recherche ...
```

Que constate-t-on ?

Exercice 1 : Réponse

```
# Sans l'instruction in
def recherche(l,e):
    for x in l:
        if x == e:
            return True
    return False


# Avec l'instruction in
def recherche2(l,e):
    return e in l
```

Exercice 1 : Réponse

```
def dich_search(l,e):
# Recherche dichotomique dans une liste croissante
    while len(l)>1:         # tant que liste contient >1 élément
        i = len(l)//2    # indice de l'élément médian
        if l[i] == e:         # Cas de succés
            return True
        elif l[i] < e:
            l=l[i:]        # dichotomie à droite
        else:
            l=l[:i]        # dichotomie à gauche
    if (len(l)==1 and l[0]==e): # lorsqu'il reste 1 élément
        return True
    return False            # Cas d'échec
```

Exercice 1 : Réponse sans slicing

2.2

```
def dich_search2(l,e):
    Imin, Imax = 0, len(l)-1
    while Imax - Imin >= 0:
        Imed = (Imin + Imax)//2
        if l[Imed] == e:
            return True
        elif l[Imed] < e:
            Imin = Imed + 1
        else:
            Imax = Imed - 1
    return False
```

Exercice 1.3

```
from random import random
from time import clock
L = [random() for k in range(100000)]
L.sort()

a = clock()
recherche(L,0)     b = clock()
print("recherche sans in",b-a,"secondes")

a = clock()
recherche2(L,0)
b = clock()
print("recherche avec in",b-a,"secondes")

a = clock()
dich_search(L,0)
b = clock()
print("recherche dichotomique avec slicing",b-a,"secondes")

a = clock()
dich_search2(L,0)
b = clock()
print("recherche dichotomique sans slicing",b-a,"secondes")
```

Exercice 1.3

```
recherche sans in 0.010511999999998523 secondes
recherche avec in 0.004089000000000453 secondes
recherche dichotomique avec slicing 0.0011019999999994923 secondes
recherche dichotomique sans slicing 1.2000000001677336e-05 secondes
```

On constate qu'une recherche par dichotomie est considérablement plus rapide !

• Sans dichotomie la plus rapide est celle utilisant l'instruction `in` : l'algorithme est le même ; mais il est implémenté à plus bas niveau (langage C), et donc plus rapide : une fonction prédéfinie même si elle fait la même chose qu'un algorithme que l'on écrira le fait mieux et plus vite (en python !).

• Avec dichotomie la recherche est de 4 (avec slicing) à 400 fois (sans slicing) plus rapide !

Chaque slicing nécessite une copie de la moitié de la liste qui est couteuse en temps (et en mémoire).

Sans slicing on apprécie pleinement les avantages de rapidité de la recherche par dichotomie :

• Sans dichotomie, dans le pire des cas (celui où l'élément ne figure pas dans liste), on doit parcourir toute la liste pour s'en rendre compte : 100000 élément à parcourir (nombre d'itérations de la boucle `for`).

• Avec dichotomie, dans le pire des cas, la boucle `while` est itérée $\log_2(100000) < 17$ fois.

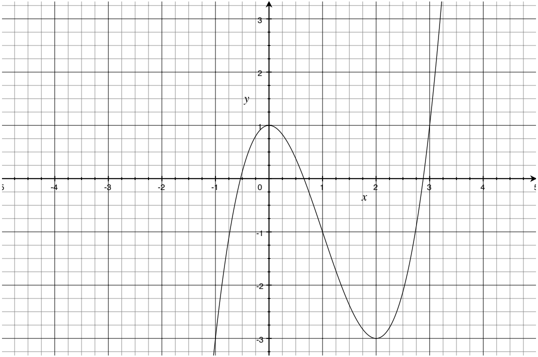
Exercice 2

Soit la fonction

$$f : x \mapsto x^3 - 3x^2 + 1$$

1. Vérifier que $f(0) = 1 > 0$ et $f(2) = -3 < 0$, et en déduire l'existence d'une solution unique dans $[0, 2]$ à l'équation $f(x) = 0$.
2. Ecrire une fontion `dich_solve()` prenant en paramètre une fonction f, un entier n, des réels (float) `a<b` tels que `f(a)f(b)`< 0 et qui retourne une valeur approchée à 10^{-n} près de la solution. La recherche s'effectuera *par dichotomie sur l'intervalle* `[a,b]`.
3. Combien de passages dans la boucle `while` sont nécessaires pour obtenir une valeur approchée à 10^{-5} près ?

Exercice 2



L'application est polynomiale et donc dérivable sur $\mathbb{R}$, de dérivée $f' : x \mapsto 3x(x-2)$ qui prend des valeurs strictement négatives dans $]0,2[$, l'application est donc strictement monotone sur $[0,2]$, et réalise donc une bijection de $[0,2]$ sur $f([0,2]) = [-3,1]$. Elle y admet donc une unique racine.

Exercice 2

```
def dich_solve(f,n,a=0,b=2):
    while (b-a)/2 >= 10**(-n):
    # Tant que précision non atteinte
        m=(a+b)/2                # Prendre le milieu de [a,b]
        if f(a) * f(m) < 0
                a, b = a, m        # dichotomie à gauche
            else:
                a, b = m, b        # dichotomie à droite
    return (a+b)/2               # Retourner le milieu
```

```
>>> f = lambda x:x**3-2*x**2+1     % lambda pour définir fonction
>>> dich_solve(f,5)
```

Exercice 2

(3) La question devient : combien de fois faut-il découper en deux parties égales un intervalle d'amplitude 2 pour obtenir des intervalles d'amplitude 10^{-5}.

Après n dichotomie l'amplitude de l'intervalle est :

$$2 \times \left(\frac{1}{2}\right)^n = 2^{1-n}$$

On cherche donc le plus petit entier n tel que $2^{1-n} \leqslant 10^{-5}$:

$$\begin{aligned}
& \iff \exp((1-n)\ln(2) \leqslant \exp(-5\ln(10)) \\
(\exp \text{ est croissante}) \quad & \iff (1-n)\ln(2) \leqslant -5\ln 10 \\
& \iff (n-1)\ln(2) \geqslant 5\ln(10) \\
(\ln(2) > 0) \quad & \iff n-1 \geqslant 5\frac{\ln(10)}{\ln(2)} \\
& \iff n \geqslant 1 + 5\log_2(10)
\end{aligned}$$

Le plus petit entier n vérifiant cela est $\lceil 1 + 5\log_2(10) \rceil = \lfloor 5\log_2(10) \rfloor + 2$ puisque $5\log_2(10)$ n'est pas un entier.