Chapitre 8 :
Algorithmes dichotomiques

Le principe général d'un algorithme dichotomique est de diviser en deux le problème à résoudre. On va voir plusieurs exemples d'applications de méthodes dichotomiques.

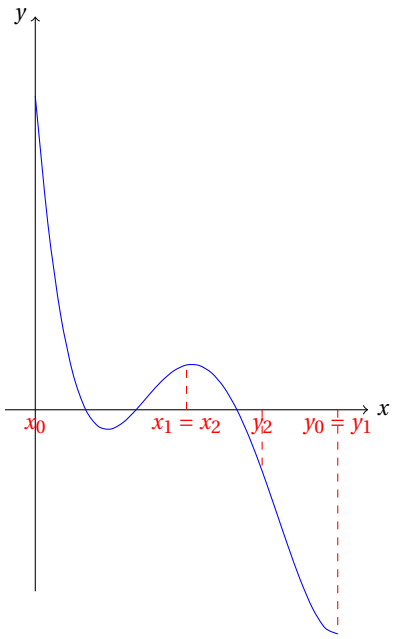
I Recherche d'un zéro d'une fonction

On rappelle le principe de la recherche d'une valeur approchée d'un zéro d'une fonction en utilisant la méthode de dichotomie.
Soient $(a,b) \in \mathbb{R}^2$, avec $a<b$ et $f \in \mathcal{C}^0([a,b],\mathbb{R})$ telle que $f(a)f(b) \leq 0$.
On pose :

$$x_0 = a,\ y_0 = b$$
$$\forall n \in \mathbb{N},\ (x_{n+1}, y_{n+1}) = \begin{cases} \left(x_n, \frac{x_n+y_n}{2}\right) & \text{si } f(x_n) f\left(\frac{x_n+y_n}{2}\right) \leq 0 \\ \left(\frac{x_n+y_n}{2}, y_n\right) & \text{sinon} \end{cases}$$

Alors, pour tout $n \in \mathbb{N}$, f admet au moins un zéro dans l'intervalle $[x_n, y_n]$ et comme cet intervalle est de longueur $\frac{b-a}{2^n}$, on obtient une valeur approchée d'un zéro de f à $\frac{b-a}{2^n}$ près.

Cette méthode ne permet pas d'obtenir tous les zéros d'une fonction mais uniquement l'un d'entre eux.



La fonction suivante prend comme arguments une fonction f, deux réels a et b et un réel strictement positif ε et elle renvoie une liste $[x,y]$ telle que f admette au moins un zéro dans $[x,y]$ et $|x-y| \leq \varepsilon$. On obtient ainsi un zéro de f à une précision de ε.

```
def dichotomie(f,a,b,epsilon) :
    assert f(a)*f(b)<=0
    x=a
    y=b
    while abs(y-x)>epsilon :
        z=(x+y)/2
        if f(x)*f(z)<=0 :
            y=z
        else :
            x=z
    return [x,y]
```

II Recherche d'un élément dans une liste d'entiers

On va écrire une fonction qui renvoie `True` si un entier `x` est dans une liste d'entiers `l` et `False` sinon. On ne travaille qu'avec des entiers car on utilise un test d'égalité.

On a déjà vu une première version de cette fonction qui utilise une boucle `for` :

```
def recherche1(l,x) :
    n=len(l)
    v=False
    for i in range(n) :
        if l[i]==x :
            v=True
    return v
```

On peut améliorer cette fonction en ne parcourant pas tous les éléments de la liste mais en s'arrêtant lorsqu'on a trouvé l'élément x.

- En utilisant du code mort :

```
def recherche2(l,x) :
    n=len(l)
    for i in range(n) :
        if l[i]==x :
            return True
    return False
```

- En utilisant un `break` :

```
def recherche3(l,x) :
    n=len(l)
    v=False
    for i in range(n) :
        if l[i]==x :
            v=True
            break
    return v
```

- En utilisant un `while` :

```
def recherche4(l,x) :
    n=len(l)
    i=0
    while i<n and l[i]!=x :
        i+=1
    if i==n :
        return False
    else :
        return True
```

⚠ L'ordre des conditions est important. Il faut mettre d'abord la condition `i<n` afin que la condition `l[i]!=x` ait un sens.

Toutes ces fonctions sont composées principalement d'une boucle de longueur maximale égale à la longueur de la liste. Elles ont un coût linéaire par rapport à la longueur de la liste.

On va améliorer le coût de la recherche en raisonnant par dichotomie.

Dans toute la suite, on considère une liste l d'**entiers**, triée par **ordre croissant**. Le principe de la recherche par dichotomie est de couper la liste en deux à chaque étape. Par exemple :

- Si l=[0,2,3,8,12] et x=8.
  - La longueur de l est 5, on a 5//2=2, on va donc comparer x=8 et $l[2] = 3$.
    On a $x \geq l[2]$ donc, si x est dans l, il est dans la deuxième moitié de la liste. On considère donc l=[3,8,12].
  - La longueur de l est 3, on a 3//2=1, on va donc comparer x=8 et $l[1] = 8$.
    On a $x \geq l[1]$ donc, si x est dans l, il est dans la deuxième moitié de la liste. On considère donc l=[8,12].
  - La longueur de l est 2, on a 2//2=1, on va donc comparer x=8 et $l[1] = 12$.
    On a $x < l[1]$ donc, si x est dans l, il est dans la première moitié de la liste. On considère donc l=[8].
  - La longueur de l est 1, donc si x est dans l, x est égal à 8, ce qui est le cas.
- Si l=[0,2,3,8,12] et x=1.
  - La longueur de l est 5, on a 5//2=2, on va donc comparer x=1 et $l[2] = 3$.
    On a $x < l[2]$ donc, si x est dans l, il est dans la première moitié de la liste. On considère donc l=[0,2].
  - La longueur de l est 2, on a 2//2=1, on va donc comparer x=8 et $l[1] = 2$.
    On a $x < l[1]$ donc, si x est dans l, il est dans la première moitié de la liste. On considère donc l=[0].
  - La longueur de l est 1, donc si x est dans l, x est égal à 0, ce qui n'est pas le cas.

La fonction est :

```
def recherchedicho(l,x) :
    i=0
    j=len(l)
    while j-i>1 :
        k=(i+j)//2
        if x<l[k] :
            j=k
        else :
            i=k
     if x==l[i] :
         return True
     else :
         return False
```