Dichotomie - Recherche dichotomique

Algorithmie

1 La méthode de la dichotomie

1.1 Méthode mathématique de recherche de zéro de fonction

On se place dans la situation où on dispose des données suivantes :

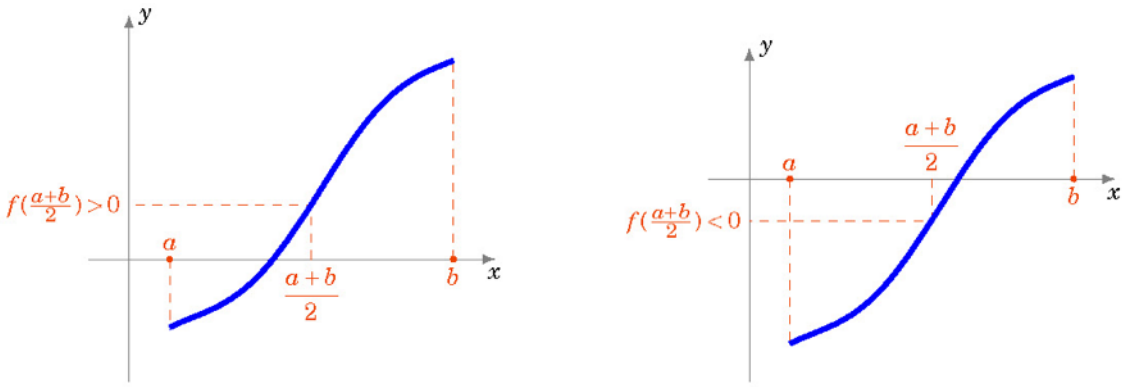
— Une fonction f continue et un intervalle sur lequel elle s'annule une fois.

— Une précision souhaitée pour la valeur approchée de la solution.

On part d'une fonction f : $[a, b] \to \mathbb{R}$ continue, avec a < b, et $f(a).f(b) \leq 0$. On construit une suite d'intervalles emboîtés, dont la longueur tend vers 0, et contenant chacun une solution de l'équation f (x) = 0.

— Si $f(a).f(\frac{a+b}{2}) \leq 0$ alors il existe c ∈ [a,$\frac{a+b}{2}$] tel que f(c) = 0

— Si $f(a).f(\frac{a+b}{2}) > 0$ alors on a $f(\frac{a+b}{2}).f(b) \leq 0$, et donc il existe c ∈ [$\frac{a+b}{2}$,b] tel que f(c) = 0



On a obtenu un intervalle de longueur moitié dans lequel l'équation f (x) = 0 admet une solution. On itère alors le procédé pour diviser de nouveau l'intervalle en deux.

Même si ce n'est pas nécessaire mathématiquement, on prendra une précaution informatique qui consistera à interrompre le processus itératif si trop d'itérations ont été effectuées. Ce nombre est choisi de manière subjective.

1.2 Implémentation informatique de la méthode de la dichotomie

On désire trouver la racine de l'équation $x^2 = 2$, ce qui revient à chercher la valeur de x annulant la fonction $f(x) = x^2 - 2$.

1.2.1 Les attendus

En plus de la simple méthode de la dichotomie et sa programmation, on désire :

— tracer la fonction f(x) pour vérifier que l'intervalle [a,b] choisi est approprié. Il s'agit ici, visuellement, de s'assurer que la fonction étudiée contient bien une racine sur l'intervalle étudié.

— compter le nombre d'itérations nécessaires à l'obtention de la solution, avec la précision désirée.

1.2.2 Ecriture et utilisation des fonctions utiles

Un fichier (incomplet), intitulé `Dichotomie_a_completer.py` vous est proposé pour mener à bien l'écriture du programme. Il permet de se concentrer sur l'écriture des fonctions utiles.

Activité 1 **S'approprier le code d'une fonction**

nom : `trace`

arguments : x_a, x_b, f, n (float, float, function, int)

effet : les arguments sont respectivement les deux bornes inférieure et supérieure de l'intervalle de recherche, la fonction f, et le nombre de points que l'on veut utiliser pour le tracé.

- S'approprier le code de la fonction, en comprenant bien les instructions utilisées.
- En utilisant la fonction `trace`, tracer la fonction $f(x) = x^2$ sur l'intervalle [1 , 2], pour vérifier qu'une racine unique se trouve bien sur l'intervalle.

Activité 2 **Ecriture d'une fonction**

nom : `dicho`

arguments : x_a, x_b, f, precision, N (float, float, function, float, int)

effet : les arguments sont respectivement les deux bornes inférieure et supérieure de l'intervalle de recherche, la fonction f, la précision à atteindre pour s'assurer de la convergence du processus, et le nombre maximum d'itérations que l'on s'autorise. On utilisera l'instruction `while`.

Cette fonction doit retourner un tuple :

— en première valeur la valeur de la solution de l'équation f(x) = 0.

— en seconde valeur le nombre d'itérations effectuées.

Pour rappel, le cœur de la technique est la suivante :

- On calcule m le milieu de l'intervalle [a , b].
- Si $f(a).f(m) < 0$ alors le nouvel intervalle contenant la solution est [a , m]. On reprend le processus de calcul du milieu du segment.
- Sinon le nouvel intervalle contenant la solution est [m , b]. On reprend le processus de calcul du milieu du segment.

2 Recherche dichotomique

2.1 But

Soit une liste L, contenant plusieurs éléments. **On désire savoir à quel endroit se trouve un élément particulier x dans L.**

Lorsque l'on recherche un élément dans un tableau, si celui-ci est dans un ordre aléatoire, il faut examiner chaque élément.

Par contre, si le tableau est trié, on peut en tirer profit pour aller beaucoup plus vite : c'est l'algorithme de recherche dichotomique qui fait l'objet de ce chapitre.

Pour rappel, la méthode `.sort` permet de trier les éléments d'une liste. Il existe également des algorithmes de tri à votre programme. Ils seront vus ultérieurement.

```
>>> L=[2,7,1,8,5,18,0]
>>> L.sort()
>>> L
[0, 1, 2, 5, 7, 8, 18]
```

2.2 Recherche de l'algorithme

Au démarrage, on cherche x dans toute la liste. Si on note `n = len(L)`, alors l'intervalle de recherche commence à l'indice `debut = 0` et finit à l'indice `fin = n-1`.

L'idée est de diviser par deux l'intervalle de recherche, en regardant la valeur prise par l'élément en milieu d'intervalle (d'indice `milieu = (debut+fin)//2`).

Pour trouver l'algorithme, posez-vous la question de ce qu'il faut faire pour définir le nouvel intervalle de recherche si :

- `x = L[milieu]` ?
- `x < L[milieu]` ?
- `x > L[milieu]` ?

2.3 Écriture de l'algorithme

Activité 3 **Ecriture d'une fonction**

nom : `recherche_dicho`

arguments : L,x (list,type non précis)

effet : renvoie l'indice auquel on trouve l'élément x dans la liste L. On utilisera l'instruction `while`. Pour information, si L est une liste, L.sort() permet de trier automatiquement la liste, et de replacer le résultat dans la variable L.

Si l'élément n'est pas présent, l'utilisateur doit être averti.

Utiliser la fonction avec l'exemple suivant :

```
Liste=[-1,3.14,-23.8,2,14.84,7,19.45,4,8,10,3267,89]
Liste.sort()
a=recherche_dicho(Liste,4) ; print(a)
b=recherche_dicho(Liste,5) ; print(b)
```