Recherche dichotomique

I) Présentation

On s'intéresse ici à la recherche (s'il existe) un élément e vérifiant une propriété P dans une liste triée S.

Les algorithmes de recherche dichotomique sont une illustration de la maxime "Diviser pour mieux régner". L'idée est de diviser par 2 à chaque étape la plage dans laquelle se situe éventuellement e en comparant e à l'élément médian de la plage.

Nous utiliserons cette technique de recherche dans les deux exemples classiques suivantes:

(1) Calcul approché de la solution d'une équation $f(x) = 0$ dans un segment $[a, b]$

(2) Recherche d'un élément donné dans une liste triée.

II) Calcul approché par dichotomie de la solution d'une équation $f(x) = 0$

1) Principe et mise en œuvre

On suppose que $f : [a, b] \longrightarrow \mathbb{R}$ est une fonction continue qui s'annule en changeant de signe en un unique réel $c \in [a, b]$.
On veut calculer une valeur d approchée de la solution c de l'équation $f(x) = 0$ à une précision ε donnée.

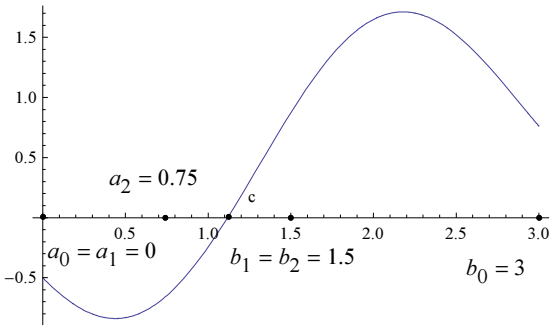
Si $f : [a, b] \longrightarrow \mathbb{R}$ change de signe sur $\left[a, \frac{a+b}{2}\right]$, alors $c \in \left[a, \frac{a+b}{2}\right]$ sinon $c \in \left[\frac{a+b}{2}, b\right]$. Puis on recommence sur le sous intervalle contenant c. On définit (voir cours de maths) deux suites adjacentes (a_n) et (b_n) qui vérifient: $\forall\, n \in \mathbb{N}$, $c \in [a_n, b_n]$ et quiconvergent vers c :

$a_0 = a\,,\ b_0 = b$

$\forall\, n \in \mathbb{N}$ on pose $c_n = \frac{a_n+b_n}{2}$; si $f(a_n)f(c_n) \leqslant 0$, alors $\begin{cases} a_{n+1} = a_n \\ b_{n+1} = c_n \end{cases}$ sinon $\begin{cases} a_{n+1} = c_n \\ b_{n+1} = b_n \end{cases}$

L'algorithme calculera les premiers termes des suites (a_n) et (b_n). Il suffira de s'arrêter lorsque $|b_n - a_n| < \varepsilon$. On devrait pouvoir alors affirmer que $d = \frac{a_n+b_n}{2}$ est une valeur approchée de c à ε près (et même à $\frac{\varepsilon}{2}$ près).

Ici, $f(x) = -\cos\left(2x - \frac{\pi}{3}\right) + \frac{x}{3}$, $[a, b] = [0, 3]$



Q1) Ecrire une fonction dicho(f, a, b, eps) (avec f fonction quelconque, et a, b, eps réels avec a < b) calculant par la méthode décrite ci-dessus le réel $d = \frac{a_n+b_n}{2}$, où n est le plus petit entier tel que $|b_n - a_n| <$ eps. Utiliser une boucle while.

Vérifier qu'avec $f(x) = -\cos\left(2x - \frac{\pi}{3}\right) + \frac{x}{3}$, alors dicho(f, 0, 3, 10**-8) = 1.1180478287860751.

Q2) Calculer $\sqrt{2}$ à 10^{-10} (sans utiliser de racine carrée !) avec la fonction dicho(), en expliquant vos choix.

Q3) Résoudre dans $\mathbb{R}$ l'équation $x^3 = 3x - 1$ avec la fonction dicho(), en expliquant vos choix.

2) Comment choisir au mieux eps ?

Pour calculer la solution c de l'équation $f(x) = 0$ avec la plus grande précision possible, il faut choisir la plus petite valeur possible de ε. Mais il ne faut pas prendre une valeur trop petite: vérifier par exemple que dicho(f, 0, 3, 10**-16) ne donne pas de réponse avec $f(x) = -\cos\left(2x - \frac{\pi}{3}\right) + \frac{x}{3}$.

Pourquoi ? La boucle while ne se termine pas car on aura jamais b-a < eps ... Mathématiquement c'est impossible, mais numériquement c'est possible. Lorsque l'écart $b_n - a_n$ devient plus petit qu'une valeur critique alors on pourra avoir $a_{n+1} = a_n$ et $b_{n+1} = b_n$: en effet, a_n et b_n seront alors deux flottants consécutifs et il n'y a plus de flottant disponible entre eux...

Comme les suites (a_n) et (b_n) sont constantes APCR N, si l'écart $b_N - a_N >$ eps, la boucle while ne se termine pas.

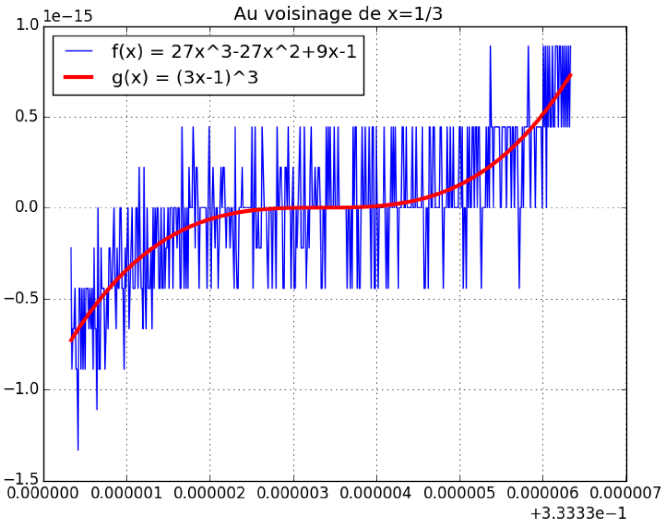
Q4) On veut une solution "sans ε" la meilleure possible: écrire une fonction dicho_meilleure(f, a, b) qui calcule par dichotomie la solution la plus précise possible de l'équation $f(x) = 0$, en sortant de la boucle while lorsque $\frac{a_n+b_n}{2} = a_n$ ou $\frac{a_n+b_n}{2} = b_n$.

Vérifier qu'avec $f(x) = -\cos\left(2x - \frac{\pi}{3}\right) + \frac{x}{3}$, dicho_meilleure(f, 0, 3) = 1.1180478290945293

3) La précision eps n'est pas garantie

Q5) On pose $f(x) = 27x^3 - 27x^2 + 9x - 1$ et $g(x) = (3x - 1)^3$ (Mathématiquent, $f(x) = g(x)$ et $f(x) = g(x) = 0 \Leftrightarrow x = \frac{1}{3}$).
Vérifier que que dicho_meilleure(f, 0, 1) = 0.33333157474589803 , puis que dicho_meilleure(g, 0, 1) = 0.33333333333333326.
On obtient la solution à 10^{-6} près pour f et à 10^{-16} près pour g (Au fait pourquoi 0.33..326 et pourquoi pas 0.33333 pour g ?)

Le problème vient d'erreurs d'éliminations qui font que la fonction $f(x) = 27x^3 - 27x^2 + 9x - 1$ est évaluée numériquement (le plus souvent) à 0 sur un intervalle $\left[\frac{1}{3} - a, \frac{1}{3} + a\right]$ bien plus grand qu'on le pense, ce qui n'est pas le cas pour g.



Dès qu'on "tombe" sur un $c_n = \frac{a_n+b_n}{2}$ tel que $f(c_n) = 0$ numériquement, alors $a_{n+1} = c_n$, mais il se peut que $c < c_n$. Du coup la solution c cherchée n'est plus dans l'intervalle $[a_{n+1}, b_{n+1}]$ de recherche et l'algorithme renvoie une valeur c' bidon. avec $c' > c$. Quand ce nombre c' est loin de la vraie solution c (ici c'est le cas pour les raisons exposées ci-dessus), la fonction dicho() renvoie la valeur qu'elle doit renvoyer, mais qui n'est pas celle que l'on espérait...

Hélas ici aucun patch systématique ! Il faut organiser le calcul de $f(x)$ au voisinage de la solution cherchée de manière à ce qu'il y ait le moins possible d'erreurs d'élimination. Dans notre exemple, $f(x) = (3x - 1)^3$ est la meilleure façon de calculer $f(x)$.

III) Recherche dichotomique d'un élément dans une liste triée

1) Objectif

On s'intéresse ici au problème de la recherche d'un élément e dans une liste triée s. On se pose les questions suivantes:

(1) Cet élément e est-il présent dans la liste s ?

(2) Si oui, quelle est la (une) position de e dans la liste s ?

(3) Si non, à quelle position de la liste s l'élément e devrait-il être inséré pour en conserver l'ordre ?

Les positions (comme en Python), sont indexées à partir de 0.

Avec s = [2, 3, 5, 7, 7, 11, 13, 17], une liste de n = 8 éléments écrite s = [s[0], s[1], ..., s[n-1]].

L'élément e = 5 est présent dans s à position = 2

L'élément e = 4 n'est pas présent dans s. Il devrait être inséré à position = 2

L'élément e = 1 n'est pas présent dans s. Il devrait être inséré à position = 0

L'élément e = 20 n'est pas présent dans s. Il devrait être inséré à position = 8

2) Méthode et notations

Nous travaillerons sur des listes de nombres entiers ou flottants triées par ordre croissant, mais les algorithmes proposés s'adaptent à toute liste ordonnée d'objets.

On cherche un entier e dans la liste d'entiers s = [s[0], s[1], ..., s[n-1]] triée par ordre croissant de n entiers. Voilà le principe:

Les entiers a et b ci-dessous "pointent" sur des positions (des indices) de la liste s dans laquelle on cherche e.
(1) On initialise les entiers a et b aux valeurs a = 0 et b = n - 1
(2) Tant que a ⩽ b , on compare e à s[c] avec c = (a+b) // 2 (division entière) Alors, dans l'ordre:
    2a) Si s[c] = e, alors e est présent à position = c et on termine
    2b) Si e < s[c], alors b = c - 1 # La plage de recherche de e devient [s[a], s[c-1]]
    2c) Si e > s[c], alors a = c + 1 # La plage de recherche de e devient [s[c+1], s[b]]
(3) On n'a pas terminé avec 2a), alors a > b . L'entier e n'est pas présent dans s et doit être inséré à position = a

3) Exemples manuels de recherche dichotomique

On pose s = [2, 3, 5, 7, 7, 9, 11, 13, 13, 13, 17, 19, 23, 29]. La liste s est une liste de longueur n = 14.

On note a_k, b_k, c_k les valeurs de a, b, c après la $k^{\text{ième}}$ itération de la boucle "tant que" (On pose $a_0 = 0$, $b_0 = n - 1$, c_0 n'existe pas)

Q6) Indiquer dans un unique tableau clair, les valeurs a_k, b_k, c_k ($k \geqslant 0$) successives pour la recherche des éléments:
e = 1 ; e = 5 ; e = 13 ; e = 20 ; e = 30 . Encadrer parmi les valeurs la position (réelle ou d'insertion) de l'élément e cherché.

4) Algorithme

Q7) Ecrire une fonction recherche_dichotomique(e, s) (avec e ∈ ℤ et s liste triée de nombres) calculant le couple (B, k) (B est un booléen et k est un entier) définis par les conditions suivantes:

- B = True ⇔ (e ∈ s)
- (e ∈ s et s[k] = e) ou (e ∉ s et en insérant e en position k de s l'ordre de s serait conservé)

Remarque: En python, un couple (un cas particulier de n-uplet) est de type tuple. Ses éléments sont indexés comme ceux d'une liste. Par exemple, pour le couple C = (True, 8), on aura C[0] = True et C[1] = 8.

Tester vos fonctions avec s = [2, 3, 5, 7, 7, 9, 11, 13, 13, 13, 17, 19, 23, 29] et e ∈ {1, 5, 13, 20, 30}, comparer avec 3).

5) Une application

On pose `s = [floor(10**8 *cos(i)) for i in range(10**6)]` et
`a = [floor(10**8 *sin(i)) for i in range(10**5)]`. (N'oubliez pas `from math import *`).

Q8) Ecrire une fonction final() calculant le nombre N d'éléments de la liste a présents dans la liste s à l'aide de la fonction recherche_dichotomique.

6) Pour aller plus loin: terminaison et complexité

Dans chaque itération de la boucle while, il y a $O(1)$ opérations élémentaires. La complexité de l'algoritnme est donc $O(N)$ où N est le nombre d'éxécution de la boucle while. On note $n = \text{len(s)}$.

On note a_k et b_k les valeurs de a et b après la $k^{\text{ième}}$ itération et $\Delta_k = b_k - a_k$. On note $a_0 = 0$, $b_0 = n - 1$ et $\Delta_0 = n - 1$ (avant la première itération)

Q9) Démontrer que pour $k \geqslant 0$, $\Delta_{k+1} \leqslant \frac{\Delta_k - 1}{2}$. (On pourra distinguer suivant la parité de a_k et de b_k)

Q10) En déduire que l'algorithme de recherche dichotomique se termine. On note $k_{\max}$ la valeur finale de k à la sortie de la boucle "tant que".

Q11) Prouver que pour $k \in \mathbb{N}$, $\Delta_k + 1 \leqslant \frac{1}{2^k}(\Delta_0 + 1)$. En déduire que $k_{\max} \leqslant \left\lfloor \frac{\ln(n)}{\ln(2)} \right\rfloor + 1$, puis que l'algorithme est de complexité $O(\ln(n))$. C'est un excellent algorithme.