

Informatique

Newton – Dichotomie

Résumé

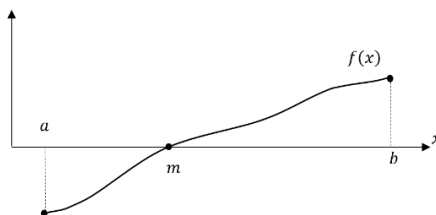
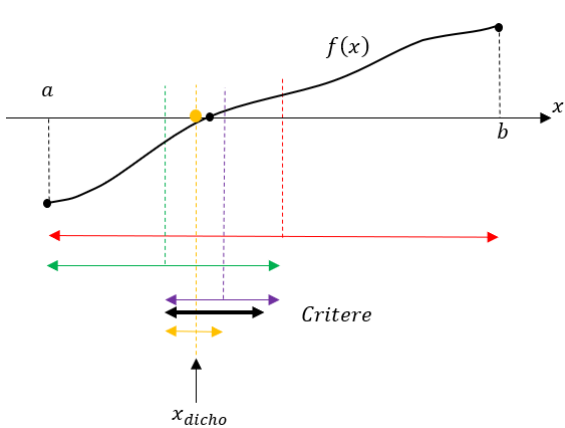
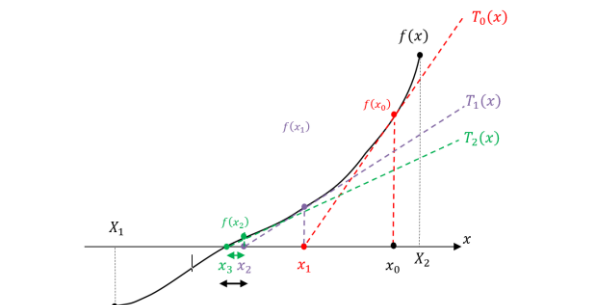
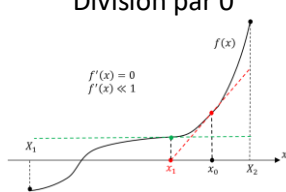
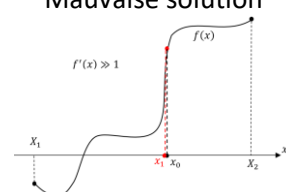
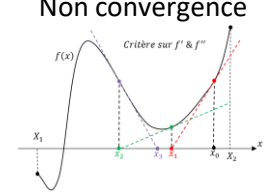
Recherche par dichotomie dans un tableau trié

Le principe de la recherche par dichotomie dans une liste triée est le suivant :

- Vérifier que L est non vide, sinon retourner False
- Définir la plage d'indices de recherche notée $[I_g, I_d]$ (pour Indice gauche, Indice droite) qui au départ vaut $[0, N - 1]$
- Déterminer l'indice milieu I_m de la plage de recherche : $I_m = \text{int}\left(\frac{I_g + I_d}{2}\right)$
- Déterminer la valeur T_m de l'élément de la liste L à l'indice I_m
- Tant que $I_g \neq I_d$ et $E \neq T_m$, répéter la procédure suivante :
 - $I_m = \text{int}\left(\frac{I_g + I_d}{2}\right)$ et $T_m = L[I_m]$
 - Si $E < T_m$, définir la nouvelle plage de recherche à $[I_g, I_m - 1]$
 - Si $E > T_m$, définir la nouvelle plage de recherche à $[I_m + 1, I_d]$
- A la sortie de la boucle deux possibilités :
 - $E = T_m$, renvoyer True
 - $I_g = I_d$, renvoyer True si $L[I_g] = E$, False sinon

```
def recherche_dicho(L,E):
    if len(L)==0:
        return False
    ig = 0
    id = len(L) - 1
    im = (ig + id) // 2
    Tm = L[im]
    while ig != id and Tm != E:
        im = (ig + id) // 2
        Tm = L[im]
        if E >= Tm: # Droite
            ig = im + 1 # im exclus
        else: # Gauche
            id = im - 1 # im exclus
    if Tm==E:
        return True
    else:
        Tm = L[ig]
        return Tm == E
```

Meilleur des cas	$O(1)$
Pire des cas	$O(\ln n)$

Résolution de l'équation $f(x) = 0$		
<p>Soit une fonction $f(x)$ définie et continue sur un intervalle $[X_1, X_2]$ telle qu'il existe m tel que :</p> $\forall X_1 < m, \forall X_2 > m, f(X_1)f(X_2) \leq 0$		
Dichotomie	Newton	
<p>Déterminer l'abscisse au centre de l'intervalle $x_m = \frac{x_1 + x_2}{2}$ et calculer son image par $f_m = f(x_m)$</p> <p>Identifier dans quel intervalle $[x_1, x_m]$ ou $[x_m, x_2]$ se trouve la solution de $f(x) = 0$ à l'aide du produit $f(x_1)f(m)$ ou $f(m)f(x_2)$</p> <p>Définir le nouvel intervalle de recherche $[x_1, x_2]$ comme celui dans lequel la solution existe</p> <p>Continuer jusqu'à obtention du critère d'arrêt</p> $ x_2 - x_1 < \varepsilon. \text{ Renvoyer } \begin{cases} x_1 \text{ ou } x_2 : < \varepsilon \\ \frac{x_1 + x_2}{2} : < \frac{\varepsilon}{2} \end{cases}$ <p>ATTENTION : test $<$ ou \leq, si $f(x_1)f(m) = 0$, x_1 doit être dans le nouvel intervalle</p>		
<p>$O(\ln \delta)$ avec $\delta = \frac{b-a}{\varepsilon}$</p> <p>Condition de convergence : $f(a)f(b) \leq 0$</p> <p>Variante : itération i strictement croissant dans l'intervalle $\left[0, \left\lceil \log_2 \left(\frac{b-a}{\varepsilon} \right) \right\rceil + 1 \right]$</p>	<p>En une abscisse x_i, déterminer l'équation de la tangente à la fonction :</p> $T_i(x) = f(x_i) + f'(x_i)(x - x_i)$ <p>Déterminer l'abscisse x_{i+1} où cette tangente croise l'axe des abscisses :</p> $x_{i+1}/T_i(x_{i+1}) = f(x_i) + f'(x_i)(x_{i+1} - x_i) = 0$ <p>Soit :</p> $x_{i+1} = \frac{x_i f'(x_i) - f(x_i)}{f'(x_i)} = x_i - \frac{f(x_i)}{f'(x_i)}$ <p>Continuer jusqu'à obtention du critère d'arrêt</p> $ x_{i+1} - x_i < \varepsilon. \text{ Renvoyer } x_{i+1}, \text{ précision } < \varepsilon$	
		
<p>Remarque pour les deux méthodes</p> <p>Ne pas être trop gourmand avec ε en gardant en tête la limite de la représentation des nombres en virgule flottante</p> <p>Il arrive que l'on souhaite un critère d'arrêt sur f, soit $f(m) < \varepsilon$</p> <p>Dans ce cas, renvoyer le m associé pour obtenir la précision sur $f < \varepsilon$</p>	Division par 0	Dérivée approchée
		$\begin{cases} f'(x) \approx \frac{f(x) - f(x-h)}{h} (*) \\ f'(x) \approx \frac{f(x+h) - f(x)}{h} \\ f'(x) \approx \frac{f(x+h) - f(x-h)}{2h} \end{cases} \text{ avec } h \ll 1$ <p>(*) Méthode de la sécante</p> $x_{i+1} = x_i - \frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})} f(x_i)$
	Mauvaise solution	Non convergence
		

Exemples de programmations	
Dichotomie	Newton
<pre>def f(x): return x**2-1</pre>	
<pre>def dicho(f,a,b,eps): assert f(a)*f(b) <= 0, "Pas de solution dans [a,b]" while abs(b-a) > 2*eps: m = (a+b)/2 if f(a)*f(m) <= 0: b = m else: a = m return (a+b)/2 def dicho(f,a,b,eps): assert f(a)*f(b) <= 0, "Pas de solution dans [a,b]" while abs(b-a) > eps: m = (a+b)/2 if f(a)*f(m) <= 0: b = m else: a = m return m sol = dicho(f,0,1,0.00001)</pre>	<pre>def newton(f,fp,x0,eps): xi = x0 xip1 = x0 + 2*eps while abs(xip1-xi) >= eps: xi = xip1 xip1 = xi - f(xi)/fp(f,xi) return xip1 def fp1(f,x): # arg f nécessaire pour newton Val = 2 * x return Val def fp2(f,x): # Approximation dx = 1e-5 Val = (f(x+dx)-f(x))/dx return Val sol = newton(f,fp1,0,0.00001)</pre>
<pre>def dicho_rec(f,a,b,eps): assert f(a)*f(b) <= 0, "Pas de solution dans [a,b]" m = (a+b)/2 if abs(b-a) < 2*eps: return m else: if f(a)*f(m) <= 0: b = m else: a = m return dicho_rec(f,a,b,eps) Sol = dicho_rec(f,0,1,0.00001)</pre>	
<pre>print(Sol)</pre>	
Sans assertion, une solution est renvoyée même si elle n'existe pas !	Si la solution n'existe pas, le code risque de ne jamais s'arrêter

Concernant la dichotomie, voici les tests possibles et l'intervalle associé :

$f(x_1)f(m) \leq 0$	$f(x_1)f(m) > 0$	$f(m)f(x_2) \leq 0$	$f(m)f(x_2) > 0$
$[x_1, m]$	$[m, x_2]$	$[m, x_2]$	$[x_1, m]$

Fonction native python pour la dichotomie : bisect de scipy.optimize

```
from scipy.optimize import bisect

def f(x):
    return x**2-1

a = 0
b = 1
Res = bisect(f,a,b)
print(Res)
```

Ne fonctionne que si $f(a)f(b) < 0$

Fonction native python pour Newton : newton de scipy.optimize

```
from scipy.optimize import newton

def f(x):
    return x**2-1

def fp(x):
    return 2*x

x0 = 2
Res = newton(f,x0,fp)
print(Res)
```

On peut préciser la fonction dérivée fp si connue

```
from scipy.optimize import newton

def f(x):
    return x**2-1

x0 = 2
Res = newton(f,x0)
print(Res)
```

Sans fprime en 3° argument ou avec fprime=None: Méthode de la sécante (dérivée inutile).

```
from scipy.optimize import newton

def f(x):
    return x**2-1

def fp(x):
    return 2*x

x0 = 2
Res = newton(f,x0,tol=0.1)
print(Res)
```

On peut préciser l'écart entre deux valeurs successives arrêtant les iterations en précisant `tol=0.1`. Lorsque rien n'est précisé, la fonction adapter automatiquement cet écart intelligemment à chaque itération