

Informatique

Newton – Dichotomie

Cours

Newton - Dichotomie	3
1.I. Dichotomie.....	3
1.I.1 Contexte.....	3
1.I.2 Objectif.....	3
1.I.3 Principe	4
1.I.4 Exemple.....	4
1.I.5 Remarque.....	5
1.I.6 Analyse de l'algorithme.....	5
1.I.6.a Correction.....	5
1.I.6.b Terminaison et complexité.....	5
1.I.7 Exemple de programmation.....	6
1.II. Newton.....	7
1.II.1 Contexte.....	7
1.II.2 Objectif.....	7
1.II.3 Principe	8
1.II.4 Exemple.....	8
1.II.5 Remarques	9
1.II.6 Analyse de l'algorithme.....	9
1.II.7 Limites.....	10
1.II.7.a Division par zéro	10
1.II.7.b Non convergence.....	10
1.II.7.c Convergence vers une mauvaise solution	11
1.II.8 Exemple de programmation.....	11
1.III. Comparaison des méthodes	12
1.IV. Remarque.....	13
1.V. Fonction natives Python.....	13
1.V.1 Dichotomie – Fonction « bisect »	13
1.V.2 Newton – Fonction « newton »	14

Newton - Dichotomie

1.I. Dichotomie

Le principe de la dichotomie est de diviser le domaine de recherche par 2 à chaque itération.

1.I.1 Contexte

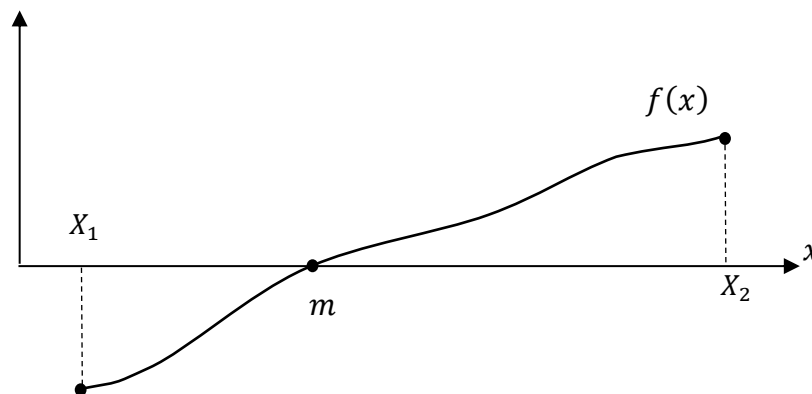
Soit une fonction $f(x)$ définie et continue sur un intervalle de départ $[X_1, X_2]$ telle qu'il existe m tel que :

$$\forall X_1 < m, \forall X_2 > m, f(X_1)f(X_2) \leq 0$$

On se placera ici dans le cas où la fonction est monotone sur l'intervalle d'étude $[X_1, X_2]$.

Autrement dit, la fonction possède un signe sur $[X_1, m]$ et le signe opposé sur $[m, X_2]$.

Exemple :



1.I.2 Objectif

On souhaite déterminer une solution approchée de l'équation :

$$f(x) = 0$$

Appelons x_{sol} la solution exacte de cette équation et x_{dicho} la solution approchée obtenue par dichotomie. Nous allons par cette méthode déterminer un intervalle $[x_1, x_2]$ tel que :

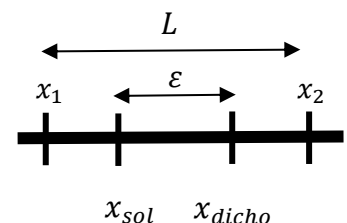
$$\begin{cases} L = |x_2 - x_1| \leq Critere \\ (x_{sol}, x_{dicho}) \in [x_1, x_2]^2 \end{cases}$$

On saura alors que l'écart entre la solution exacte et la solution approchée est inférieur à *Critere* (et même à $x_2 - x_1$) :

$$\varepsilon = |x_{dicho} - x_{sol}| \leq Critere$$

Où *Critere* est un réel définissant la précision de la solution obtenue.

Remarque : écrire $\leq Critere$ ou $< Critere$ n'a que peu d'importance.



1.I.3 Principe

Le principe de cette recherche par dichotomie est le suivant :

- Vérifier éventuellement l'existence de la solution sur $[X_1, X_2]$. Attention, la solution $f(x) = 0$ existe sur l'intervalle $[X_1, X_2]$ si et seulement si $f(X_1)f(X_2) \leq 0$ – Le « ou égale » est important
- Initialiser $[x_1, x_2] = [X_1, X_2]$
- Déterminer l'abscisse au centre de l'intervalle $x_m = \frac{x_1+x_2}{2}$ et calculer l'image $f_m = f(x_m)$
- Identifier dans quel intervalle $[x_1, x_m]$ ou $[x_m, x_2]$ se trouve la solution de $f(x) = 0$ par étude du signe de $f(x_1)f(m)$ ou de $f(m)f(x_2)$
- Définir le nouvel intervalle de recherche $[x_1, x_2]$ comme celui dans lequel la solution existe
- Continuer tant que l'intervalle de recherche est de largeur supérieure à un ε dépendant du critère imposé

A la fin, il faut renvoyer une valeur de x dans l'intervalle obtenu lorsque le critère est vérifié.

- Renvoyer x_1 ou x_2 permet d'obtenir une précision inférieure à ε
- Renvoyer la valeur au centre x_m permet d'obtenir une précision inférieure à $\frac{\varepsilon}{2}$

Il faut veiller à mettre le bon test $<, \leq, >$ ou \geq face au bon sous intervalle atteint :

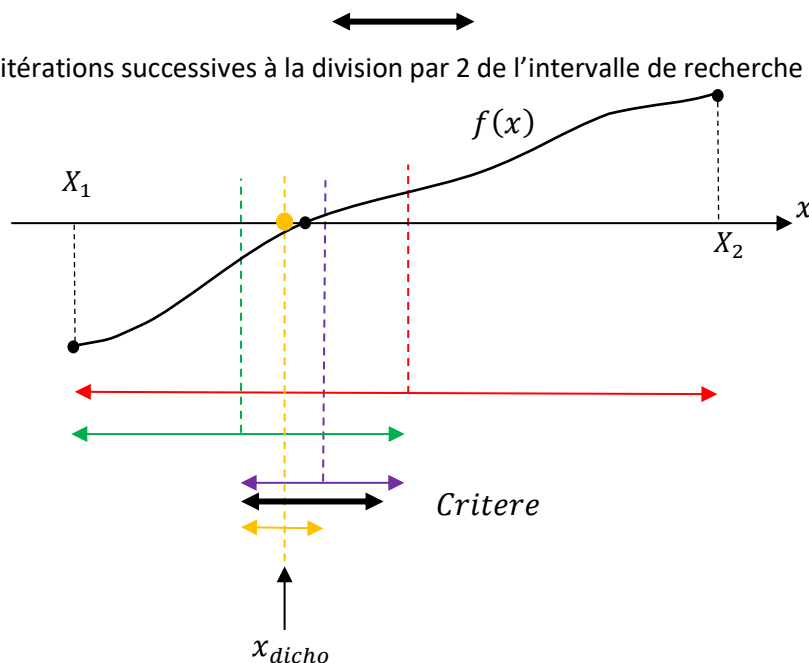
$f(x_1) f(m) \leq 0$	$f(x_1) f(m) > 0$	$f(m) f(x_2) \leq 0$	$f(m) f(x_2) > 0$
$[x_1, m]$	$[m, x_2]$	$[m, x_2]$	$[x_1, m]$

On remarque qu'il faut que le sous intervalle atteint contienne les abscisses du test nul.

1.I.4 Exemple

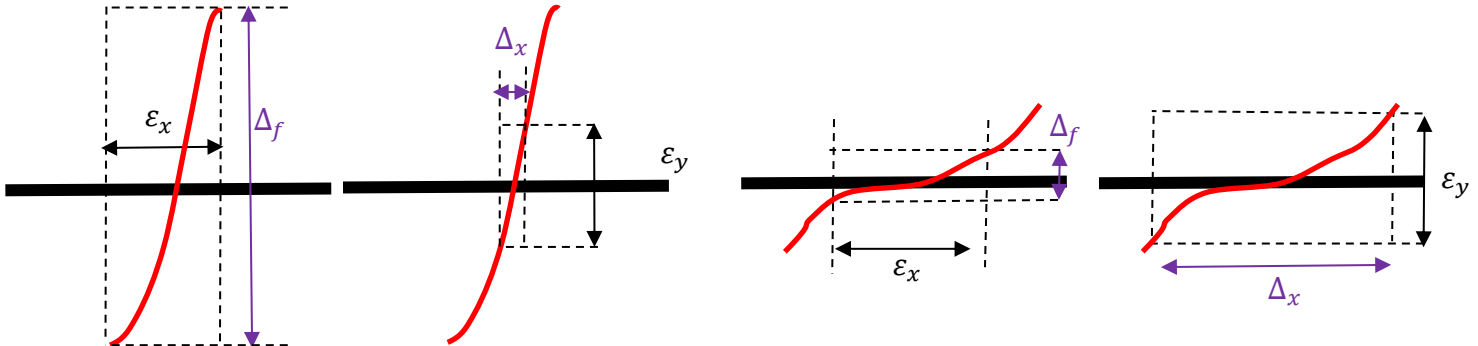
Soit un critère tel que la longueur de l'intervalle final soit inférieure à la longueur de la flèche ci-dessous :

On procède par itérations successives à la division par 2 de l'intervalle de recherche :



1.I.5 Remarque

Il est envisageable de proposer un critère d'arrêt sur les ordonnées $|f(x)|$ plutôt que sur les abscisses, voire les deux en même temps. Cela dépend de la forme de la fonction et de la précision attendue, sur x ou sur $f(x)$. Voici l'illustration pour deux fonctions en prenant les mêmes critères :



1.I.6 Analyse de l'algorithme

1.I.6.a Correction

Vous montrerez dans le cours de mathématiques que :

Si $f(a)f(b) \leq 0$, l'algorithme permet bien d'obtenir une valeur approchée de la solution de l'équation $f(x) = 0$ sur l'intervalle $[a, b]$

Ceci prouvera la correction de l'algorithme.

1.I.6.b Terminaison et complexité

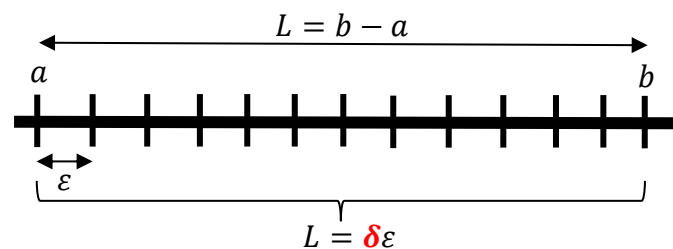
La longueur *delta* de l'intervalle étant divisée par 2 à chaque itération de la boucle while, on peut remarquer aisément que, au bout de α itérations de la boucle : $delta = \frac{b-a}{2^\alpha}$

Pour démontrer la terminaison et déterminer la complexité de l'algorithme on résout donc l'inéquation d'inconnue α suivante où ε est le critère d'arrêt :

$$\frac{b-a}{2^\alpha} < \varepsilon \Leftrightarrow 2^\alpha = e^{\alpha \ln(2)} > \frac{b-a}{\varepsilon} \Leftrightarrow \alpha \ln(2) > \ln\left(\frac{b-a}{\varepsilon}\right) \Leftrightarrow \alpha > \frac{\ln\left(\frac{b-a}{\varepsilon}\right)}{\ln(2)} = \log_2\left(\frac{b-a}{\varepsilon}\right)$$

On a trouvé un variant qui est le numéro de l'itération i , entier qui croît strictement dans l'intervalle $\left[0, \left\lceil \log_2\left(\frac{b-a}{\varepsilon}\right) \right\rceil + 1\right]$. Ainsi, l'algorithme se termine toujours et la complexité est en $O\left(\log_2\left(\frac{b-a}{\varepsilon}\right)\right)$ soit une complexité logarithmique.

Soit $\delta = \frac{b-a}{\varepsilon}$ le nombre d'intervalles de longueur ε entre a et b . Alors la complexité de l'algorithme de dichotomie continue est en $O(\log_2(\delta)) = O(\ln \delta)$.



Plus on recherche la solution sur un intervalle grand, et plus le critère est petit, plus le temps d'exécution sera long.

1.I.7 Exemple de programmation

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

```
def dichot(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 dichot(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 = dichot(f,0,1,0.00001)
```

```
def dichot_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 dichot_rec(f,a,b,eps)
```

```
Sol = dichot_rec(f,0,1,0.00001)
```

```
print(Sol)
```

Sans assertion, une solution est renvoyée même si elle n'existe pas !

1.II. Newton

La méthode de Newton est une seconde méthode de résolution d'équation de la forme $f(x) = 0$ qui converge plus vite que la méthode de dichotomie vers la solution recherchée.

1.II.1 Contexte

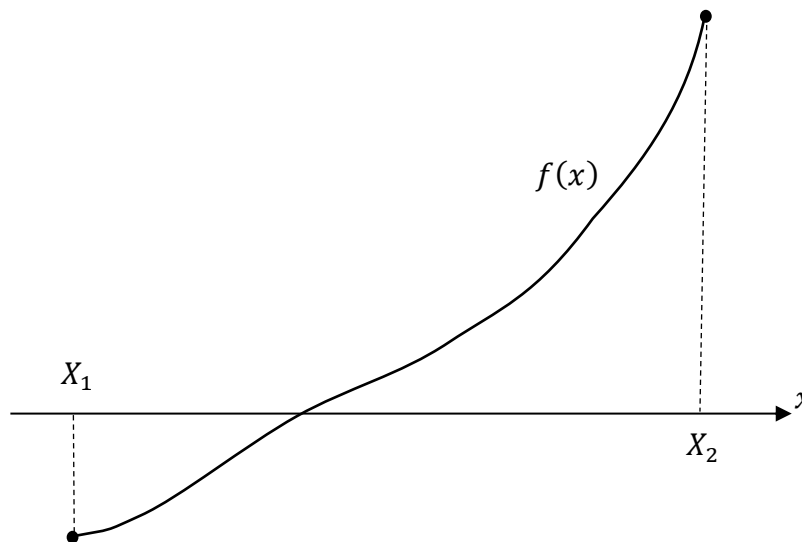
Soit une fonction $f(x)$ définie, continue et dérivable sur un intervalle $[X_1, X_2]$ telle qu'il existe m tel que :

$$\forall a < m, \forall b > m, f(a)f(b) < 0$$

On se placera ici dans le cas où la fonction est monotone sur l'intervalle d'étude $[X_1, X_2]$.

Autrement dit, la fonction possède un signe sur $[X_1, m]$ et le signe opposé sur $[m, X_2]$.

Exemple :



1.II.2 Objectif

On souhaite déterminer une solution approchée de l'équation :

$$f(x) = 0$$

Contrairement à la résolution par Dichotomie vue au paragraphe précédent, nous ne sommes pas en mesure d'être sûrs que la solution approchée par la méthode de Newton est à une distance précise de la solution réelle.

Nous allons suivre l'écart entre deux solutions successives et proposer un critère d'arrêt lorsque deux solutions successives sont « assez proches ».

1.II.3 Principe

Le principe de la méthode de Newton consiste à approcher la courbe de $f(x)$ avec sa tangente $T_0(x) = f(x_0) + f'(x_0)(x - x_0)$ en un point initial x_0 choisi arbitrairement. On détermine alors l'abscisse du point d'intersection $(x_1, 0)$ de cette tangente avec l'axe des abscisses, soit $T_0(x_1) = 0$.

On procède alors ainsi par itérations :

$$x_{i+1}/T_i(x_{i+1}) = f(x_i) + f'(x_i)(x_{i+1} - x_i) = 0$$

Soit :

$$f(x_i) + f'(x_i)x_{i+1} - x_i f'(x_i) = 0$$
$$x_{i+1} = \frac{x_i f'(x_i) - f(x_i)}{f'(x_i)} = x_i - \frac{f(x_i)}{f'(x_i)}$$

On procède alors ainsi jusqu'à ce que :

$$|x_{i+1} - x_i| \leq \text{ou} < \text{Critere}$$

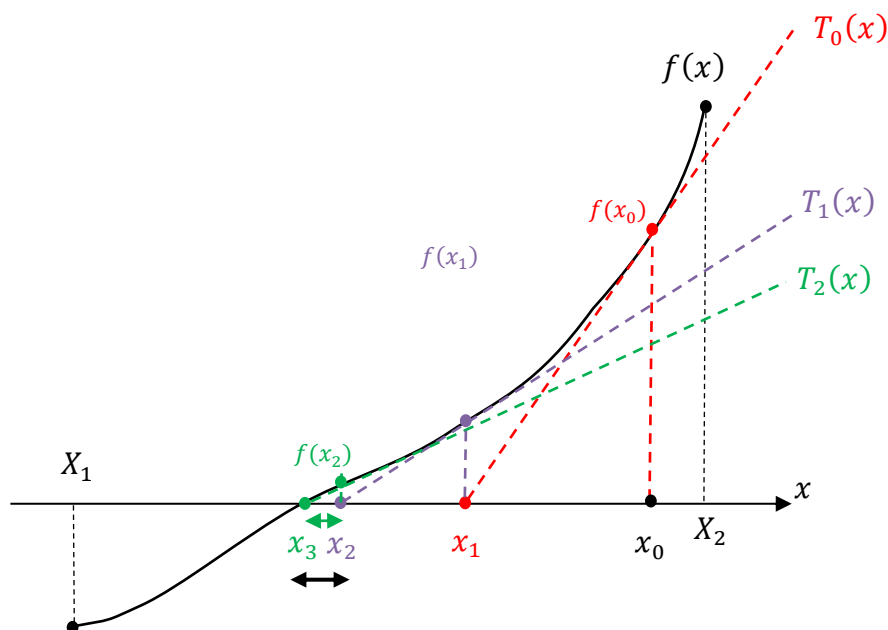
On renvoie alors x_{i+1} avec une précision $< \varepsilon$

1.II.4 Exemple

Soit un critère tel que la longueur de l'intervalle final soit inférieure à la longueur de la flèche ci-dessous :

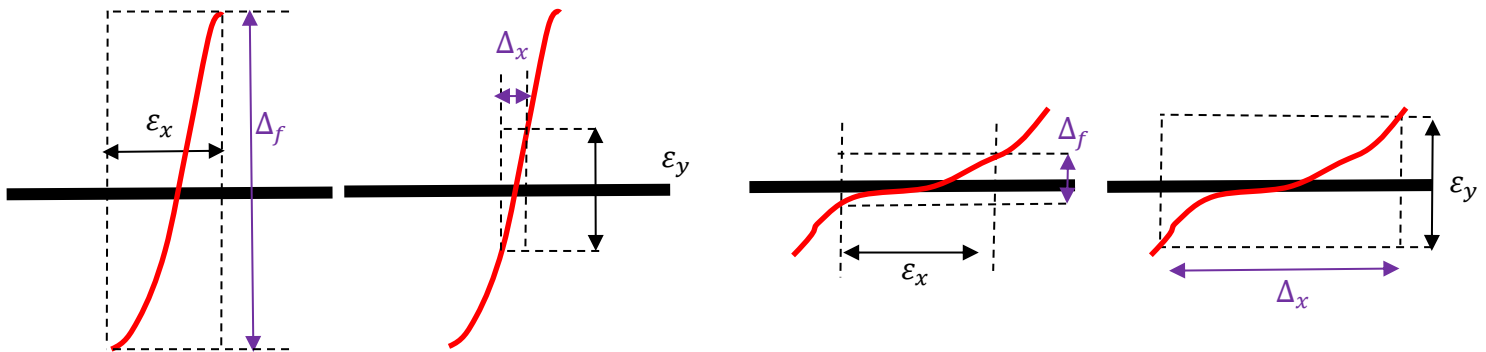


On procède par itérations successives :

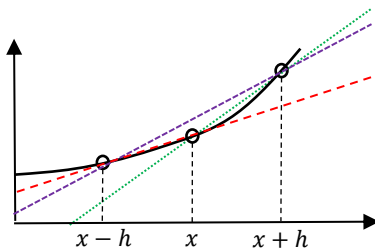


1.II.5 Remarques

Comme pour la dichotomie, il est envisageable de proposer un critère d'arrêt sur les ordonnées $|f(x)|$ plutôt que sur les abscisses, voire les deux en même temps. Cela dépend de la forme de la fonction et de la précision attendue, sur x ou sur $f(x)$. Voici l'illustration pour deux fonctions en prenant les mêmes critères :



- Pour pouvoir appliquer la méthode de Newton, il faut a priori connaître la fonction f et sa dérivée f' . Dans le cas où on ne connaît que la fonction f et que l'on ne peut pas déterminer sa dérivée, il est possible d'obtenir une approximation de f' à partir des valeurs de f en utilisant l'une des formules suivantes :



$$\begin{cases} f'(x) \approx \frac{f(x) - f(x-h)}{h} \\ f'(x) \approx \frac{f(x+h) - f(x)}{h} \\ f'(x) \approx \frac{f(x+h) - f(x-h)}{2h} \end{cases} \text{ avec } h \ll 1$$

- En choisissant d'approximer $f'(x_i)$ par $\frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}$ dans la formule de calcul de x_{i+1} de la méthode de Newton, on obtient une méthode dérivée de la méthode de Newton appelée **méthode de la sécante** (utile quand on ne connaît pas la dérivée de f):

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} = x_i - \frac{f(x_i)}{\frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}} = x_i - \frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})} f(x_i)$$

1.II.6 Analyse de l'algorithme

On peut montrer mathématiquement qu'une condition suffisante pour assurer la convergence de la méthode de Newton est que f soit de classe C^2 sur $[a, b]$ avec $f' > 0$, $f'' > 0$ sur $[a, b]$ et $f(a)f(b) < 0$. **Ceci sera démontré en maths** lorsque vous aurez les outils nécessaires.

La complexité sera démontrée en maths. Quand la méthode converge, la convergence est quadratique. **La précision double à chaque itération.** Autrement dit, $|x_n - x_{sol}|$ est dominé par une suite k^{2^n} , $k \in]0, 1[$.

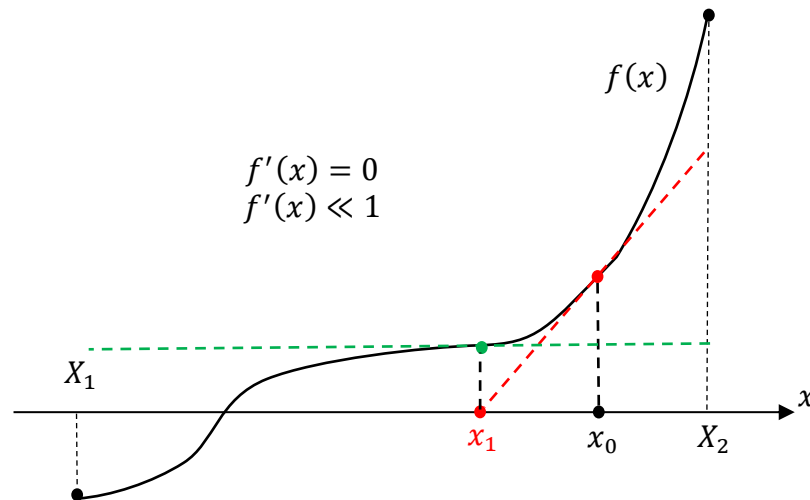
1.II.7 Limites

1.II.7.a Division par zéro

À tout moment, on calcule :

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

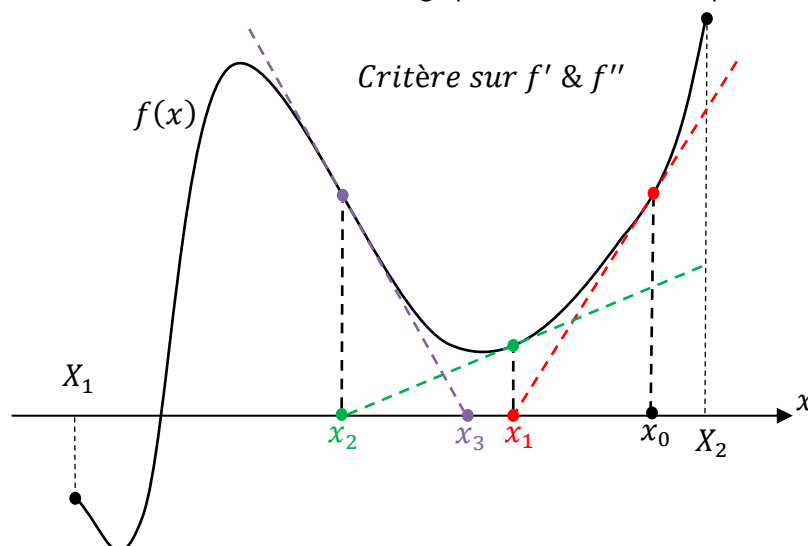
Il peut donc y avoir un problème si $\exists x \in [X_1, X_2] / f'(x) = 0$. En effet, la tangente étant horizontale, il n'est plus possible de trouver une nouvelle valeur de x ...



D'une manière générale, si $f'(x_i) \ll 1$, x_{i+1} risque de devenir très grand. On pourra donc ajouter un test du type : si $f'(x_i) < 10^{-6}$ alors on stoppe le programme.

1.II.7.b Non convergence

Il peut arriver que la méthode de Newton ne converge pas. En voici un exemple :

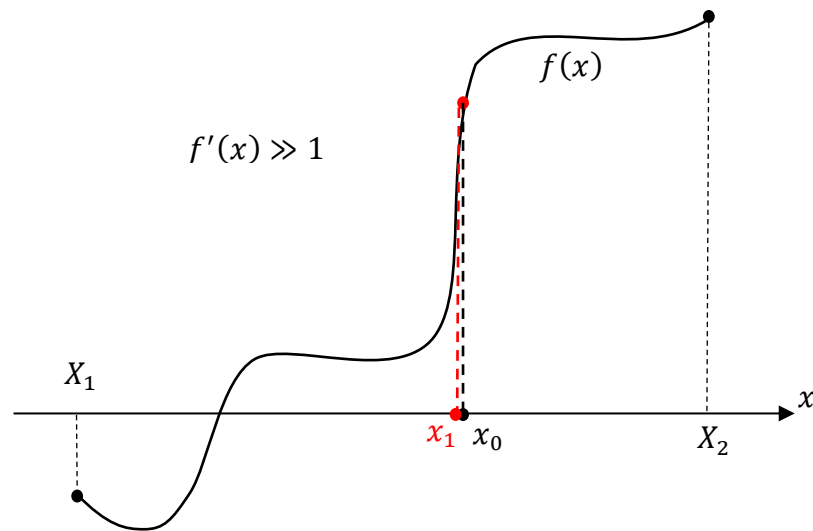


En effet, la fonction tracée ne respecte pas le critère : $f' > 0, f'' > 0$ sur $[X_1, X_2]$.

Remarque : on pourra compter les itérations réalisées et stopper le programme après N itérations...

1.II.7.c Convergence vers une mauvaise solution

Imaginons une fonction dont la pente est très verticale sur un intervalle (fonction presque non dérivable...) :



Du fait de la présence d'un critère d'arrêt sur deux valeurs successives de la solution, on voit clairement que selon la pente de la fonction et la largeur du critère, il y aura arrêt sur une solution éloignée de la vraie solution.

1.II.8 Exemple de programmation

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

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)
print(sol)
```

Si la solution n'existe pas, le code risque de ne jamais s'arrêter

1.III. Comparaison des méthodes

Lorsque la méthode de Newton converge, elle converge vite comparé à la méthode de Dichotomie. Toutefois, elle est dépendante de :

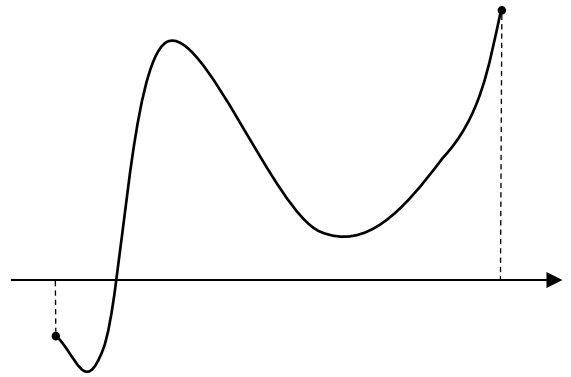
- L'allure de la fonction (monotonie en particulier)
- La dérivée (tangente horizontale et verticale problématiques)

La méthode de Newton peut effectuer de grandes variations d'une itération à l'autre et induire des temps de convergence longs.

La méthode de dichotomie, bien que plus lente, assure cependant une progression constante de l'intervalle de précision. Elle est très bien adaptée aux fonctions non monotones par exemple.

On pourra envisager des stratégies du type :

- Résolution de Newton sur 100 itérations
- Si la convergence n'est pas atteinte, résolution par Dichotomie



1.IV. Remarque

Quelle que soit la méthode, garder en tête que nous manipulons des nombres codés en virgule flottante, et ne pas chercher des critères ε trop petits.

1.V. Fonction natives Python

1.V.1 Dichotomie – Fonction « bisect »

A la condition que le produit $f(a)f(b) < 0$, il existe une fonction qui trouve le 0 d'une fonction sur un intervalle :

Code	Exécution
<pre>from scipy.optimize import bisect def f(x): return x**2-1 a = 0 b = 1 Res = bisect(f,a,b) print(Res)</pre>	<pre>>>> (executing file "<tmp 1>") -1.0</pre>

1.V.2 Newton – Fonction « newton »

A la condition que le produit $f(a)f(b) < 0$, il existe une fonction qui trouve le 0 d'une fonction sur un intervalle :

Code	Exécution
<pre> 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) </pre>	<pre> >>> (executing cell "" (line 1 of "<tmp 1>")) 1.0 </pre>
<pre> from scipy.optimize import newton def f(x): return x**2-1 x0 = 2 Res = newton(f,x0,fprime=None) print(Res) </pre>	<pre> >>> (executing file "<tmp 1>") 1.0000000000000002 </pre>
<pre> 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) </pre>	<pre> >>> (executing file "<tmp 1>") 1.0082694424882084 </pre>

On peut préciser ne pas préciser « fprime=None » en 3° argument , ce qui revient au même. Qu'on le précise ou non, on applique finalement la méthode de la sécante. Mais si on le souhaite, on peut préciser en 3° argument la fonction dérivée si elle est connue, pour plus de précision.

Lorsque l'on ne précise pas de critère dans la fonction de Newton, celle-ci estime d'elle-même évoluant à chaque itération, proche du carré de la précision à l'étape précédente. Voici l'extrait de la documentation associée :

The convergence rate of the Newton-Raphson method is quadratic, the Halley method is cubic, and the secant method is sub-quadratic. This means that if the function is well-behaved the actual error in the estimated zero after the n th iteration is approximately the square (cube for Halley) of the error after the $(n-1)$ th step. However, the stopping criterion used here is the step size and there is no guarantee that a zero has been found.

Consequently, the result should be verified. Safer algorithms are `brentq`, `brenth`, `ridder`, and `bisect`, but they all require that the root first be bracketed in an interval where the function changes sign. The `brentq` algorithm is recommended for general use in one dimensional problems when such an interval has been found.