L1 - Calcul Scientifique

Youssef Chahir
youssef.chahir@unicaen.fr

Université de Caen Normandie

Introduction

- Résolution d'équations non linéaires

  Recherche des solutions de l'équation non linéaire $f(x)=0$ où f est une fonction donnée

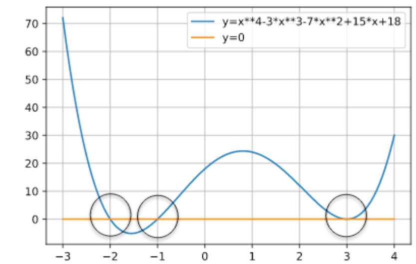
  – De nombreux problèmes peuvent se résoudre en procédant à la recherche des racines de fonctions non linéaires

L'idée que nous allons exploiter dans ce chapitre pour résoudre des équations est le fait que :
« La résolution d'une équation peut se ramener à rechercher les zéros d'une fonction. »

- Etude de plusieurs méthodes numériques permettant par approximations successives de déterminer les zéros avec la *précision demandée*. Ces méthodes nécessitent de connaître pour chaque zéro un intervalle [a;b] qui ne contienne pas d'autre zéro que celui recherché.
  1. Méthode de la bisection
  2. Méthode de Newton
  3. Méthode de la sécante
  4. Méthode du point fixe

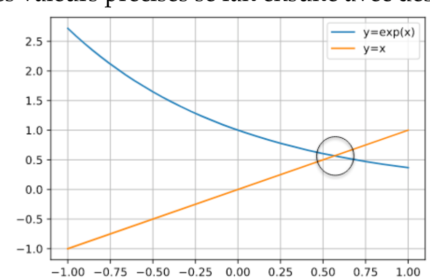
Recherche de racines d'équations non linéaires

- Étant donné une fonction f(x), le problème et de trouver la (les) valeur(s) de x telle(s) que f(x)=0
- Plusieurs racines possibles
- Par exemple, l'équation $x^4-3x^3-7x^2+15x=-18$
  peut s'écrire : $(x+2)(x-3)^2(x+1)$
  et a donc 4 racines
- D'une manière générale, ne peut être résolu par des méthodes directes (analytiques) -> méthodes numériques
- Interprétation graphique :



Différentes approches de résolution

- Méthodes analytiques : ne sont possibles que pour une toute petite classe de problèmes
  – Exemple : résoudre $ax^2+bx+c=0$
  est possible en utilisant : $roots=\frac{-b\pm\sqrt{b^2-4ac}}{2a}$
  – L'équation : $x-e^{-x}=0$ ne peut être résolue de manière analytique
- Les méthodes graphiques sont un premier pas pour imaginer où se trouvent les racines
- La recherche des valeurs précises se fait ensuite avec des méthodes numériques



Différentes méthodes numériques

- Un très grand nombre de méthodes, parmi lesquelles :
  - Méthode par Bisection
  - Méthode Newton
  - Méthode de la sécante
  - Point fixe
  - Etc.
- Question de la convergence
  - une méthode est dite convergente si la séquence des estimations d'une racine $x_1, x_2, \ldots, x_n$ est telle que pour toute valeur de $\varepsilon > 0$ il existe un N tel :

$$|x - x_n| < \epsilon, \;\; n > N$$

  que x est la racine
- Toutes les méthodes ne convergent pas à la même vitesse, certaines ne convergent pas dans certains cas

Vitesses de convergence

- Supposons que $x_1, x_2, \ldots, x_n$ converge vers x
  - La convergence est linéaire quand : $\dfrac{|x_{n+1} - x|}{|x_n - x|} \leq C$
  - La convergence est quadratique quand : $\dfrac{|x_{n+1} - x|}{|x_n - x|^2} \leq C$
  - La convergence est d'ordre P quand : $\dfrac{|x_{n+1} - x|}{|x_n - x|^p} \leq C$

Chapitre IV : Recherche des racines de fonctions non linéaires

Méthodes par encadrement :
Méthode de la bisection / dichotomie

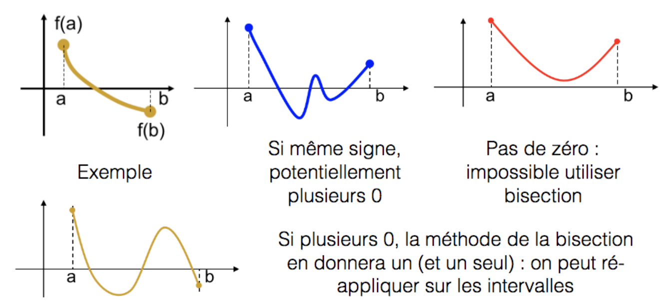
Théorème des valeurs intermédiaires

- Soit f(x) une fonction continue définie sur l'intervalle [a,b].
- Théorème de Bolzano / des valeurs intermédiaires :

Théorème 1.
Soit $f : [a,b] \to \mathbb{R}$ une fonction continue sur un segment.

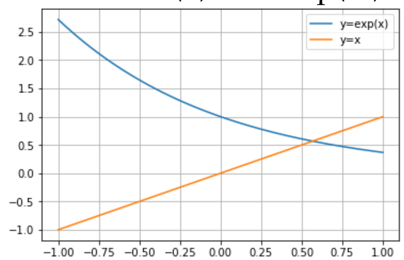
$$\boxed{\text{Si } f(a)\cdot f(b) \leqslant 0, \text{ alors il existe } \ell \in [a,b] \text{ tel que } f(\ell) = 0.}$$

  - Si f(a) et f(b) ont des signes différents, alors la fonction possède au moins 1 zéro sur [a,b]



Méthodes d'encadrement

- La méthode d'encadrement la plus simple consiste à diviser l'intervalle d'étude (obtenu par une méthode graphique) et à regarder à quel moment la fonction change de signe.
- Revenons à la fonction f(x) = x - exp(-x)



- La racine se trouve dans l'intervalle [0.5, 0.75]
- Nous pouvons le subdiviser en 100 valeurs et repérer le changement de signe

Méthode de la bisection

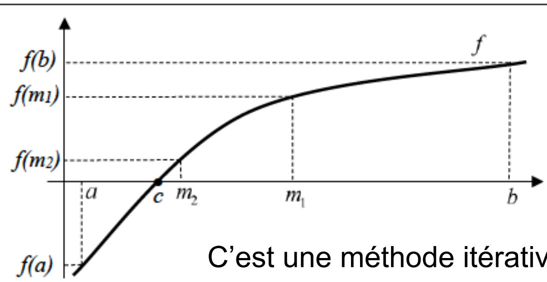
1) On divise l'intervalle $[a;b]$ en deux parties égales et on note son milieu par $m_1 = \dfrac{a+b}{2}$.

2) Si $f(a)\cdot f(m_1) \le 0$ **alors** le zéro se trouve dans cet intervalle et on continue la méthode sur l'intervalle $[a\,;m_1]$ en prenant $m_2 = \dfrac{a+m_1}{2}$ le milieu de $[a;m_1]$.

3) Sinon, on a nécessairement que $f(m_1)\cdot f(b) \le 0$ et on poursuit avec l'intervalle $[m_1;b]$.

4) On poursuit les calculs tant que la longueur de l'intervalle est supérieure à une *tolérance positive donnée (précision)*.

Illustration



C'est une méthode itérative

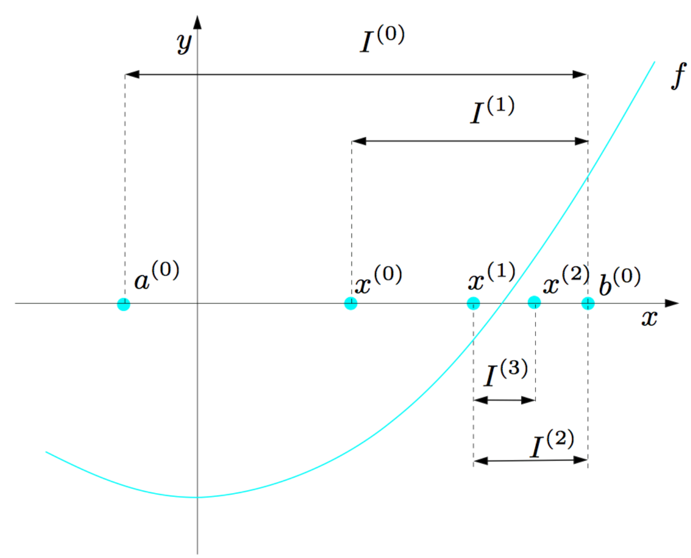
Méthode de bisection

1. on pose $x^{(0)} = \frac{a+b}{2}$,
2. si $f(x^{(0)}) = 0$ alors $x^{(0)}$ est le zéro cherché.
3. si $f(x^{(0)}) \neq 0$:
   (a) soit $f(x^{(0)})f(a) > 0$ et alors le zéro $\alpha \in (x^{(0)}, b)$ et on définit $a = x^{(0)}$ et $x^{(1)} = (a+b)/2$ pour ce nouveau a
   (b) soit $f(x^{(0)})f(a) < 0$ et alors $\alpha \in (a, x^{(0)})$ et on pose $b = x^{(0)}$ et $x^{(1)} = (a+b)/2$ pour ce nouveau b

Par des divisions de ce type, on construit la suite $x^{(0)}, x^{(1)}, \ldots, x^{(k)}$ qui vérifie pour tout k,

$$\boxed{|x^{(k)} - \alpha| \le \frac{b-a}{2^{k+1}},}$$

Méthode de bisection (suite)



Implementation (bis)

```
def f(x):
    return x**3-3*x**2+1

# Méthode de la bissection

def BISSECTION(y,a,b,Tol):
    while b-a>Tol:
        m=(a+b)/2
        if y(a)*y(m)<=0:
            b=m
        else:
            a=m
    return a,b

# Programme principal

a=eval(input("Entrez la borne inférieure de l'intervalle : a="))
b=eval(input("Entrez la borne supérieure de l'intervalle : b="))
Tol=eval(input("Entrez la précision : Tol="))

while f(a)*f(b)>0 or Tol<0:
    a=eval(input("Entrez la borne inférieure de l'intervalle : a="))
    b=eval(input("Entrez la borne supérieure de l'intervalle : b="))
    Tol=eval(input("Entrez la précision : Tol="))

a,b = BISSECTION(f,a,b,Tol)

print("Un zéro de f se trouve dans l'intervalle [",a,";",b,"]")
```


Méthode naïve (avec boucle) / Méthode 'Numpy'> (sans boucle)

```
def f(x):
    return x-np.exp(-x)

x = np.linspace(0.5,0.75,100)

for i in range(99):
    if f(x[i])*f(x[i+1])<0:
        print('[',x[i],x[i+1],']')
        print('[',f(x[i]),f(x[i+1]),']')

[ 0.565656565657 0.568181818182 ]
[ -0.00233053782516 0.00162721609235 ]
```

```
i = np.argwhere(f(x[1:])*f(x[:-1])<0)[0]
print('[',x[i],x[i+1],']')
print('[',f(x[i]),f(x[i+1]),']')
```

Possibilité de recommencer sur le nouvel intervalle (plus petit)

scipy.optimize.bisect

scipy.optimize.bisect

scipy.optimize.bisect(*f, a, b, args=(), xtol=1e-12, rtol=4.4408920985006262e-16, maxiter=100, full_output=False, disp=True*) [source]

Find root of a function within an interval.

Basic bisection routine to find a zero of the function *f* between the arguments *a* and *b*. *f(a)* and *f(b)* can not have the same signs. Slow but sure.

Parameters:

f : *function*
Python function returning a number. *f* must be continuous, and f(a) and f(b) must have opposite signs.

a : *number*
One end of the bracketing interval [a,b].

b : *number*
The other end of the bracketing interval [a,b].

xtol : *number, optional*
The routine converges when a root is known to lie within *xtol* of the value return. Should be >= 0. The routine modifies this to take into account the relative precision of doubles.

rtol : *number, optional*
The routine converges when a root is known to lie within *rtol* times the value returned of the value returned. Should be >= 0. Defaults to `np.finfo(float).eps * 2`.

maxiter : *number, optional*
if convergence is not achieved in *maxiter* iterations, and error is raised. Must be >= 0.

args : *tuple, optional*
containing extra arguments for the function *f*. *f* is called by `apply(f, (x)+args)`.

full_output : *bool, optional*
If *full_output* is False, the root is returned. If *full_output* is True, the return value is `(x, r)`, where x is the root, and r is a *RootResults* object.

disp : *bool, optional*
If True, raise RuntimeError if the algorithm didn't converge.

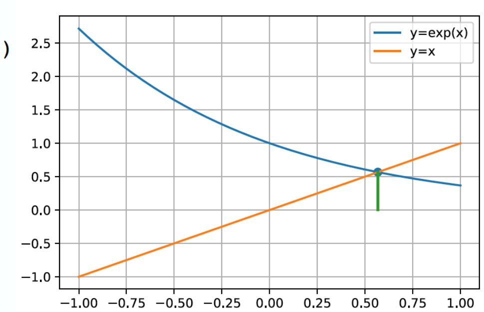
Returns:

x0 : *float*
Zero of *f* between *a* and *b*.

r : *RootResults (present if `full_output = True`)*
Object containing information about the convergence. In particular, `r.converged` is True if the routine converged.

Exemple

```
from scipy import optimize

def f(x):
    return x-np.exp(-x)

x0 = optimize.bisect(f,0.5,0.75)

x = np.linspace(-1,1,500)
y = x-f(x)

plt.plot(x,y,label='y=exp(x)')
plt.plot(x,x,label='y=x')
plt.plot([x0, x0],[0,x0-f(x0)])
plt.scatter([x0],[x0-f(x0)])

plt.grid()
plt.legend()
plt.savefig("visu.pdf")
```



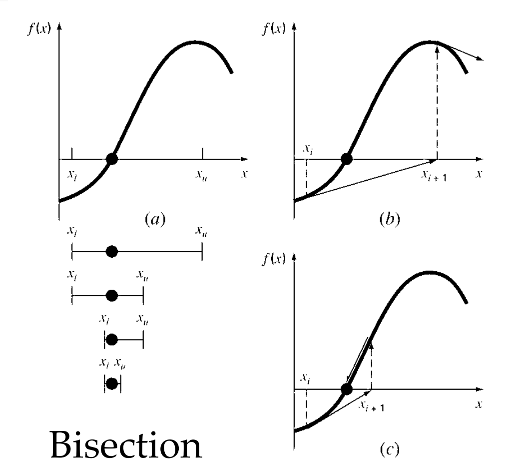
Méthode de la bisection : Avantages/Inconvénients

- Advantages
  - Simple et facile à implanter
  - Une appel à la fonction par itération
  - Réduction de l'intervalle de 50% à chaque itération
  - Possibilité de déterminer le nombre d'itérations nécessaires a priori
  - Pas besoin de connaître la dérivée de la fonction
  - La fonction ne doit pas être nécessairement différentiable
- Inconvénients :
  - Lente
  - Il est possible que certains points proches de la solutions ne soient pas conservés au fil des itérations

Chapitre IV : Recherche des racines de fonctions non linéaires

Méthode de Newton

Méthodes "ouvertes"



Ne converge pas

Converge

Pour trouver la racine, on construit une fonction $x_{i+1}= g(x_i)$ pour prédire itérativement la valeur suivante de x, jusqu'à convergence

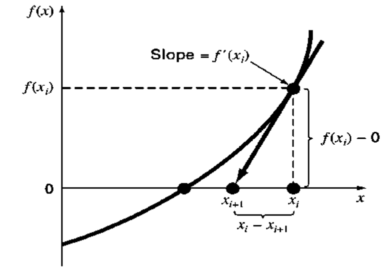
Méthodes "ouvertes"

- Comment construire la fonction g(x) ?
- Comment assurer la convergence ?
- Qu'est ce qui fait qu'une méthode converge rapidement ?
- Quelle est la vitesse de convergence ?

Méthode de Newton

- **Idée** : Approcher le graphique de la fonction f, dont on cherche à déterminer un zéro, par une droite tangente pour laquelle il sera simple de déterminer un zéro

Illustration



- Utilisation de la dérivée de la fonction pour calculer le point suivant et s'approcher de la racine
  - Suppose que la fonction soit continue et que la dérivée soit connue
  - Il faut connaître un point de départ (dénommé xi dans par la suite)
- Extrapolation de la tangente jusqu'à la rencontre avec l'axe des x

Méthode Newton :Implementation

```
def f(x):
    return x**3-3*x**2+1

def df(x):
    return 3*x**2-6*x

# Methode de Newton

def NEWTON(y,dy,x0,Tol):
    x1=x0-y(x0)/dy(x0)
    while abs(x1-x0)>Tol:
        x0=x1
        x1=x0-y(x0)/dy(x0)
    return x1

# Programme « principal »

x0=eval(input("Entrez l'approximation initiale d'un zéro x0= "))
Tol=eval(input("Entrez la précision : Tol="))

x1 = NEWTON(f,df,x0,Tol)

print("Une approximation d'un zéro de f est ",x1)
```


Méthode de Newton

- Point de départ, le théorème de Taylor : $f(x+h) \approx f(x) + hf'(x)$
- Or on recherche h tel que : $f(x+h) = 0$
- Ce qui donne : $h \approx -\dfrac{f(x)}{f'(x)}$
- La valeur suivante est donc $x_{i+1} = x_i - \dfrac{f(x)}{f'(x)}$

Soit f une fonction dérivable et soit c un zéro réel de f.

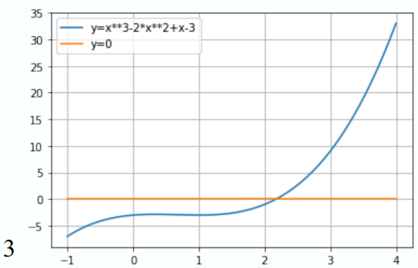
Si x_n est une approximation de c,

alors l'approximation suivante x_{n+1} est donnée par $\boxed{x_{n+1} = x_n - \dfrac{f(x_n)}{f'(x_n)}}$

à condition que $f'(x_n) \neq 0$.

Exemple

- Trouver la racine de $f(x) = x^3 - 2x^2 + x - 3$
- En partant du point $x_0 = 4$
- La dérivée est : $f'(x) = 3x^2 - 4x + 1$



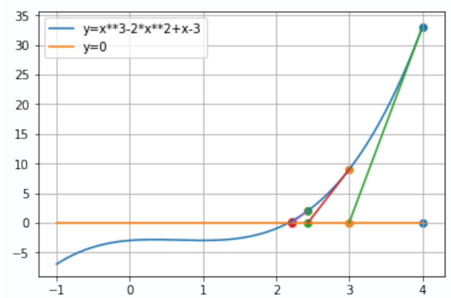
Iteration 1 : $x_1 = x_0 - \dfrac{f(x_0)}{f'(x_0)} = 4 - \dfrac{33}{33} = 3$

Iteration 2 : $x_2 = x_1 - \dfrac{f(x_1)}{f'(x_1)} = 3 - \dfrac{9}{16} = 2.4375$

Iteration 3 : $x_3 = x_2 - \dfrac{f(x_2)}{f'(x_2)} = 2.4375 - \dfrac{2.0369}{9.0742} = 2.2130$

Exemple (suite)

k (Iteration)	x_k	$f(x_k)$	$f'(x_k)$	$x_k + 1$	$\lvert x_{k+1} - x_k \rvert$
0	4	33	33	3	1
1	3	9	16	2.4375	0.5625
2	2.4375	2.0369	9.0742	2.2130	0.2245
3	2.2130	0.2564	6.8404	2.1756	0.0384
4	2.1756	0.0065	6.4969	2.1746	0.0010



Théorème de la convergence pour la méthode de Newton

Soit $f : [a;b] \to \mathbb{R}$ une fonction.

Si

1) f est deux fois continûment dérivable sur $[a;b]$ (f' est dérivable et f'' est continue)

2) $f(a)$ et $f(b)$ sont de signes opposés càd $f(a) \cdot f(b) \leq 0$

3) f' et f'' ont un signe constant sur $[a;b]$ $\left(f'(x) \neq 0 \text{ et } f''(x) \neq 0 \quad \forall x \in [a;b] \right)$

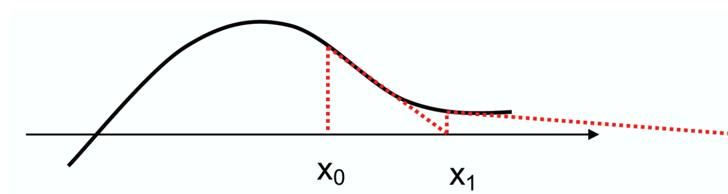
4) on choisit $x_0 \in [a;b]$ tel que $f(x_0)$ et $f''(x_0)$ sont de même signe càd $f(x_0) \cdot f''(x_0) > 0$

Alors

la suite de Newton de premier terme x_0 converge vers l'unique solution c de l'équation $f(x) = 0$ dans $[a;b]$.

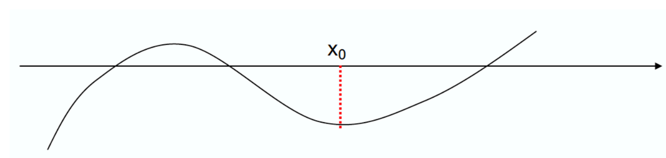
Analyse de la convergence

- Si suffisamment proche de la racine, convergence quadratique
- Cela signifie que le nombre de chiffres corrects double à chaque itération
- Limitation : il faut partir d'un point suffisamment proche de la racine, sinon la méthode diverge
- Convergence faible si la dérivée est proche de 0 sur la racine

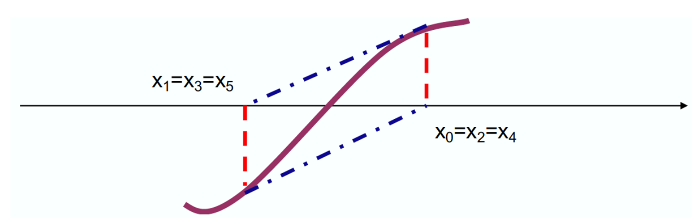


Autres difficultés

- Lorsque la dérivée est proche de 0 en x :



- Rencontre de cycles



scipy.optimize.newton

scipy.optimize.newton

scipy.optimize.newton(*func, x0, fprime=None, args=(), tol=1.48e-08, maxiter=50, fprime2=None*) [source]

Find a zero using the Newton-Raphson or secant method.

Find a zero of the function *func* given a nearby starting point *x0*. The Newton-Raphson method is used if the derivative *fprime* of *func* is provided, otherwise the secant method is used. If the second order derivate *fprime2* of *func* is provided, parabolic Halley's method is used.

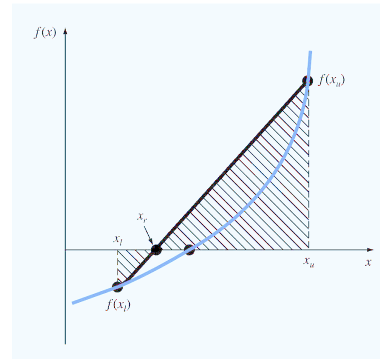
Parameters:	**func** : *function* The function whose zero is wanted. It must be a function of a single variable of the form f(x,a,b,c...), where a,b,c... are extra arguments that can be passed in the *args* parameter. **x0** : *float* An initial estimate of the zero that should be somewhere near the actual zero. **fprime** : *function, optional* The derivative of the function when available and convenient. If it is None (default), then the secant method is used. **args** : *tuple, optional* Extra arguments to be used in the function call. **tol** : *float, optional* The allowable error of the zero value. **maxiter** : *int, optional* Maximum number of iterations. **fprime2** : *function, optional* The second order derivative of the function when available and convenient. If it is None (default), then the normal Newton-Raphson or the secant method is used. If it is given, parabolic Halley's method is used.
Returns:	**zero** : *float* Estimated location where function is zero.

Exemple

```
import scipy as sc
import scipy.optimize
f=lambda x:  (x-1)**2-5
fp=lambda x: 2*x-2
print(sc.optimize.newton(f,5,fprime=fp))
=> 3.23606797749979
print(1+np.sqrt(5))
=> 3.23606797749979
```


Méthode de la sécante (ou Régula Falsi)

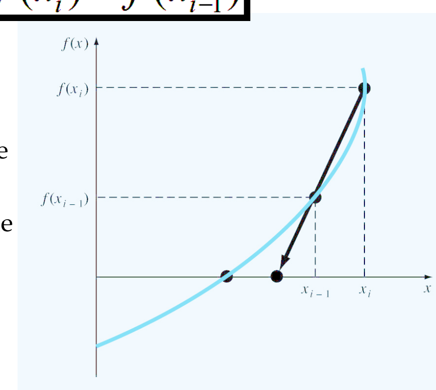
- Connue aussi sous le nom de **méthode d'interpolation linéaire**
- Contrairement à la bisection qui divise l'intervalle par 2, cette méthode considère la droite qui passe par les deux points f(a) et f(b) et calcule l'intersection avec l'axe des x.
- La pente de la droite est la pente moyenne



Méthode de la sécante

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

- Nécessite d'avoir 2 points de départ (x_0,x_1) (pas nécessairement un encadrement de la solution)
- Cependant ce n'est pas une méthode par intervalle
- Mêmes propriétés que la méthode de Newton
- Pas de test sur les signes (pas de garantie que la racine est encadrée)
- Convergence pas garantie



Méthode de la sécante

- La méthode de la sécante commence avec les points : (a, f(a)) et (b,f(b)), tels que f(a)f(b)<0.
- La ligne droite qui passe par (a, f(a)), (b, f(b)) est

$$y = f(a) + \frac{f(b) - f(a)}{b - a}(x - a)$$

- Le point suivant est celui qui coupe l'axe des x (donc y=0) :

$$x = a - \frac{b-a}{f(b) - f(a)} f(a) = \frac{af(b) - bf(a)}{f(b) - f(a)}$$

Implémentation

```python
def f(x):
    return 0.2 * (x-10)**2 + 3*x - 20
```

```python
def secante(a,b,p):
    while True:
        x = b - ( b - a ) * f(b) / ( f(b) - f(a) )
        if abs(x - b) <= 10**(-p):
            return x
        else:
            a, b = b, x
```

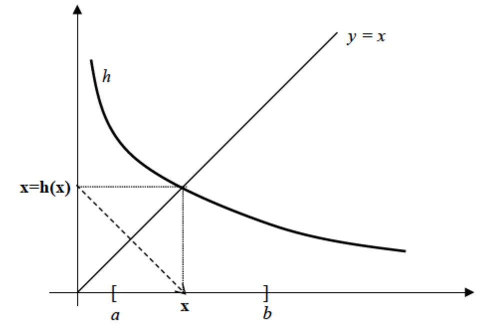
```
>>> secante(8,7,10)
5.000000000000002
```

La valeur exacte de la solution la plus proche de 7 étant 5.

Méthode des points fixes

Soit h une fonction donnée, alors toute solution de l'équation $h(x) = x$ est appelée ***un point fixe de h***. Graphiquement, cela revient à déterminer les points d'intersections entre le graphique de la fonction h et la droite identité : $y = x$. C'est aussi une valeur x tel que *la préimage* est égale à *l'image* par h.

Illustration



La résolution de l'équation $f(x) = 0$ peut se ramener à la recherche d'un point fixe pour h.

Méthode des points fixes

- Pour trouver les racines d'une fonction, on traite :
  - g(z)=z
  - Et l'on déduit f(x)
- **Exemple** : f(x)=x avec $f(x) = x^2 + 2x - 3$
- Il est possible d'écrire : $x = \frac{3 - x^2}{2}$
- S'il y a un point fixe g(x)=x pour $g(x) = \frac{3 - x^2}{2}$
- Il est possible de le trouver avec la suite :

$$x_{i+1} = \frac{3 - x_i^2}{2}$$

- Ce point vérifiera $f(x) = 0$ par construction de $g(x)$

Points fixes : Comment construire la fonction g(x) ?

- Il y a une infinité de manières de construire la fonction g(x)
- Prenons l'exemple : $f(x) = x^2 + 2x - 3$ avec la recherche de f(x)=0

$$x^2 - 2x - 3 = 0 \Rightarrow x^2 = 2x + 3 \Rightarrow x = \sqrt{2x+3} \Rightarrow g(x) = \sqrt{2x+3}$$

$$x^2 - 2x - 3 = 0 \Rightarrow x(x-2) - 3 = 0 \Rightarrow x = \frac{3}{x-2} \Rightarrow g(x) = \frac{3}{x-2}$$

$$x^2 - 2x - 3 = 0 \Rightarrow 2x = x^2 - 3 \Rightarrow x = \frac{x^2-3}{2} \Rightarrow g(x) = \frac{x^2-3}{2}$$

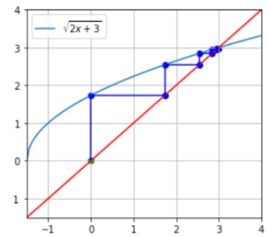
Observations

$$x_{i+1} = \sqrt{2x_i + 3} \qquad x_{i+1} = \frac{3}{x_i - 2} \qquad x_{i+1} = \frac{x_i^2 - 3}{2}$$

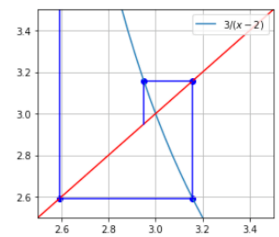
$x_{i+1} = \sqrt{2x_i + 3}$	$x_{i+1} = \frac{3}{x_i - 2}$	$x_{i+1} = \frac{x_i^2 - 3}{2}$
1. $x_0 = 4$	1. $x_0 = 4$	1. $x_0 = 4$
2. $x_1 = 3.31662$	2. $x_1 = 1.5$	2. $x_1 = 6.5$
3. $x_2 = 3.10375$	3. $x_2 = -6$	3. $x_2 = 19.625$
4. $x_3 = 3.03439$	4. $x_3 = -0.375$	4. $x_3 = 191.070$
5. $x_4 = 3.01144$	5. $x_4 = -1.263158$	
6. $x_5 = 3.00381$	6. $x_5 = -0.919355$	
	7. $x_6 = -1.02762$	
	8. $x_7 = -0.990876$	
	9. $x_8 = -1.00305$	
Converge	Diverge	Diverge

Observations

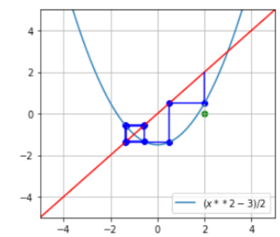
$$x_{i+1} = \sqrt{2x_i + 3} \qquad x_{i+1} = \frac{3}{x_i - 2} \qquad x_{i+1} = \frac{x_i^2 - 3}{2}$$



x0=0



x0=2.9



x0=2

La convergence dépend de la fonction et du point de départ

Implémentation

```
import math as mt

def h(x):
   return mt.log(2*x+3)

# Méthode du Point fixe

def POINTFIXE(y,x0,Tol):
   x1=y(x0)
   while abs(x1-x0)>Tol:
      x0=x1
      x1=y(x0)
   return x1

# Programme « principal »

x0=eval(input("Entrez l'approximation initiale d'un point fixe x0= "))
Tol=eval(input("Entrez la précision : Tol="))

x1 = POINTFIXE(h,x0,Tol)

print("Une approximation d'un point fixe de h est ",x1)
```

scipy.optimize.fixed_point

scipy.optimize.fixed_point(*func*, *x0*, *args=()*, *xtol=1e-08*, *maxiter=500*, *method='del2'*) [source]

Find a fixed point of the function.

Given a function of one or more variables and a starting point, find a fixed point of the function: i.e., where `func(x0) == x0`.

Parameters:

func : *function*

Function to evaluate.

x0 : *array_like*

Fixed point of function.

args : *tuple, optional*

Extra arguments to *func*.

xtol : *float, optional*

Convergence tolerance, defaults to 1e-08.

maxiter : *int, optional*

Maximum number of iterations, defaults to 500.

method : *{"del2", "iteration"}, optional*

Method of finding the fixed-point, defaults to "del2", which uses Steffensen's Method with Aitken's `Del^2` convergence acceleration [1]. The "iteration" method simply iterates the function until convergence is detected, without attempting to accelerate the convergence.