

Approximations de réels

Alexandre Benoit

BCPST

- Étant donné une fonction f , on cherche une valeur x pour laquelle

$$f(x) = 0$$

- Une solution x est une **racine** de l'équation ou un **zéro** de la fonction f .

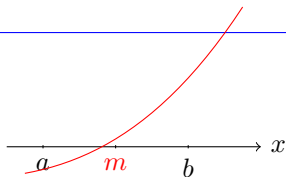
Pour simplifier le propos, dans la suite on suppose que $f(x) = 0$ admet une unique solution dans l'intervalle où l'on travaille. On peut démontrer l'unicité d'une solution à l'aide par exemple du théorème de la bijection.

I Dichotomie

Dichotomie

Le principe de la méthode par dichotomie est de partir d'un intervalle qui contient une solution et de diviser ensuite par deux sa longueur jusqu'à ce qu'on ait isolé la solution avec une précision suffisante

```
while  $(b - a) > e$  do  
   $m = a + (b - a) / 2$   
  if  $\text{signe}(f(a)) = \text{signe}(f(m))$  then  
     $a = m$   
  else  
     $b = m$   
  end  
end
```



- L'idée est que $f'(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$.
Donc quand h est proche de 0, on a :

$$f(x+h) \approx f(x) + hf'(x)$$

Il s'agit d'une fonction linéaire en h approximant f au voisinage de x

- On remplace la fonction non linéaire f par cette fonction linéaire dont la racine est $h = -f(x)/f'(x)$
- Les racines de la fonction f et de l'approximation linéaire ne sont en général pas identique donc on itère le processus

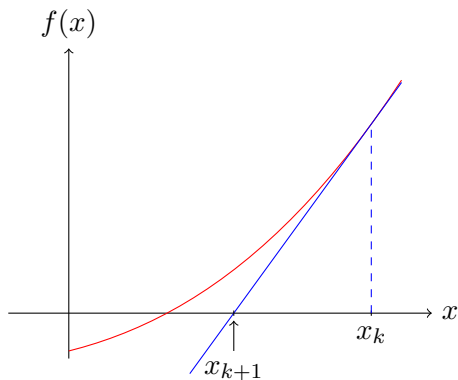
$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

- On s'arrête lorsque $x_{k+1} - x_k$ sont « assez proche ». C'est-à-dire plus petit qu'une certaine valeur e .

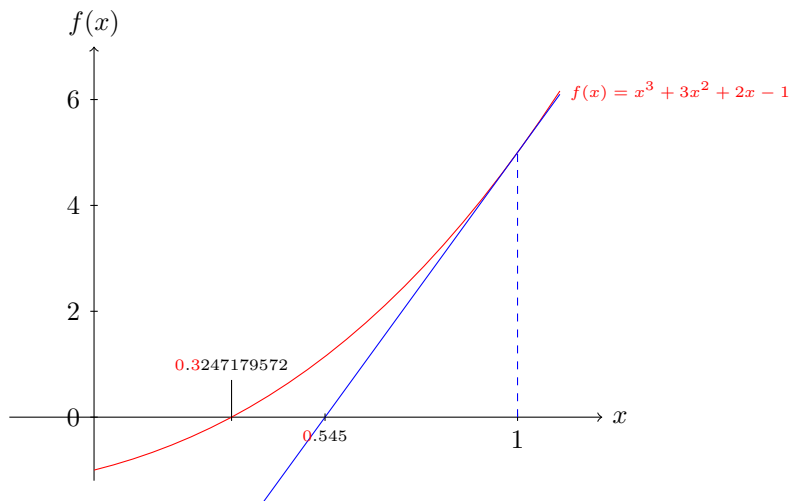
II Méthode de Newton : Graphiquement

Méthode de Newton (suite)

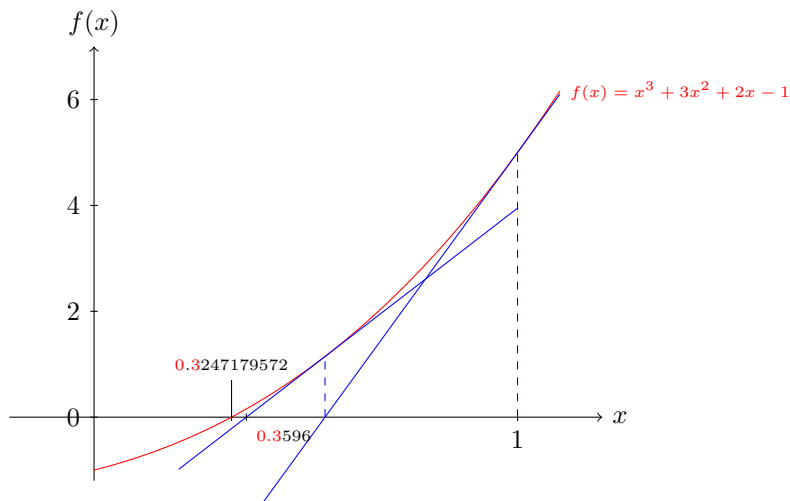
La méthode de Newton approxime une fonction non linéaire f près de x_k par sa tangente en $f(x_k)$



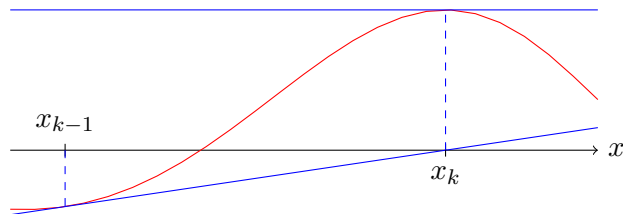
Exemple avec le polynôme $x^3 + 3x^2 + 2x - 1$



Exemple avec le polynôme $x^3 + 3x^2 + 2x - 1$

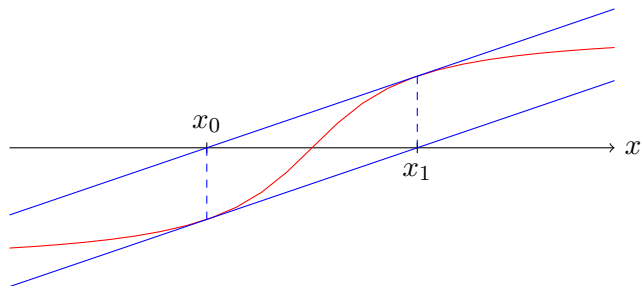


Exemples où la Méthode de Newton ne fonctionne pas (1)



La méthode ne fonctionne pas si il existe x_k tel que $f'(x_k) = 0$.

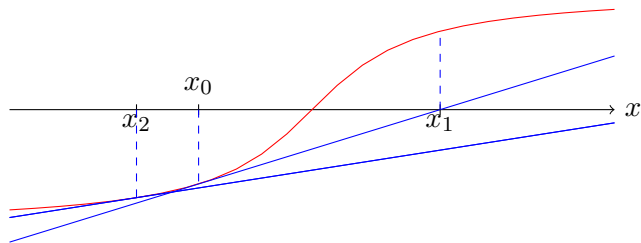
Exemples où la Méthode de Newton ne fonctionne pas (2)



La méthode peut provoquer des cycles

ici on a $x^* \neq x_0 = x_2 = x_4 = \dots$ et $x^* \neq x_1 = x_3 = x_5 = \dots$

Exemples où la Méthode de Newton ne fonctionne pas (3)



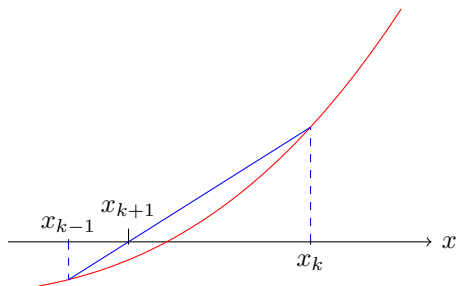
La méthode peut diverger complètement.

La méthode de la sécante

La méthode de la sécante est dérivée de celle de Newton où l'on remplace la dérivée par une sécante. Elle est utile lorsque l'on ne connaît pas la dérivée.

- On initialise l'algorithme en prenant deux points x_0 et x_1 proche de la solution recherchée
- On itère par :

$$x_{k+1} = x_k - \frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})} f(x_k)$$



Quelle méthode utilisée ?

Il n'y a pas de méthode universelle.

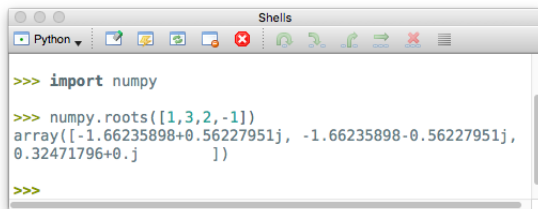
- Si on cherche simplicité et robustesse : **Dichotomie**. Elle fonctionne tout le temps elle est moins rapide mais c'est relatif pour les besoins que l'on a.
- Si on connaît une bonne approximation d'un zéro et on cherche la rapidité. On utilise une méthode du type Newton.
 - Si on connaît f' , on utilise la méthode de Newton.
 - Sinon on utilise la méthode de la sécante.
 - Si on connaît f'' , on peut utiliser la méthode de Halley qui s'inspire de Newton (à ne pas connaître).
- On peut aussi utiliser des méthodes mixtes. Location d'un zéro par dichotomie puis Newton. Si Newton ne converge pas, on peut reprendre une phase de Dichotomie.

III Python

La plupart du temps, il est déraisonnable de coder soi même une méthode de résolutions qui existe déjà et à probablement été plus testée et optimisé que votre fonction.

Recherche de zéro avec Numpy

La fonction `numpy.roots` détermine les racines d'un polynôme donné par la liste de ses coefficients.

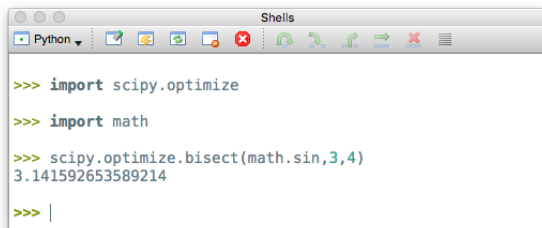
A screenshot of a Python shell window titled "Shells". The window has a menu bar with "Python" and a dropdown arrow, and a toolbar with various icons. The main area shows a Python prompt with the following code and output:

```
>>> import numpy
>>> numpy.roots([1,3,2,-1])
array([-1.66235898+0.56227951j, -1.66235898-0.56227951j,
       0.32471796+0.j          ])
>>>
```

Elle fournit les racines complexes, le nombre imaginaire i étant noté j .

Dichotomie : `scipy.optimize.bisect`

La méthode dichotomique est implémentée dans `scipy.optimize.bisect`



```
>>> import scipy.optimize
>>> import math
>>> scipy.optimize.bisect(math.sin,3,4)
3.141592653589214
>>> |
```

On obtient ici la solution de $\sin(x) = 0$ avec x compris entre 3 et 4

Newton : `scipy.optimize.newton`

La méthode de Newton est programmée dans `scipy.optimize.newton`. Il est à noter que si on ne donne pas la dérivée, c'est en fait la méthode de la sécante qui est appliquée.

```
>>> scipy.optimize.newton(math.sin,3, math.cos)
3.141592653589793
```

On peut aussi utiliser une méthode hybride (celle de Brent) qui mélange la méthode de la sécante avec la méthode de dichotomie.

```
>>> scipy.optimize.brentq(math.sin,3,4)
3.141592653589793
```