Calcul Scientifique

Cours 8: Résolution de système d'équation et zéro de fonction

Alexis Lechervy

Sommaire

Évaluation d'un ensemble d'équation

Principe

Pour évaluer un ensemble d'équation en parallèle, on réécrit le problème comme le produit matriciel entre une matrice et un vecteur.

Évaluation rapide d'un ensemble d'équation

Problème :

$$\begin{cases} 2x + 3y + 1 = ? \\ -4x + 12z = ? \\ 23x + 4y - 42z - 4 = ? \\ 17z - 2 = ? \end{cases} \text{pour } x = -11, y = 13 \text{ et } z = 16 \tag{1}$$

Réécriture

$$A = \begin{bmatrix} 2 & 3 & 0 \\ -4 & 0 & 12 \\ 23 & 4 & 42 \\ 0 & 0 & 17 \end{bmatrix}, b = \begin{bmatrix} 1 \\ 0 \\ -4 \\ -2 \end{bmatrix}, x = \begin{bmatrix} -11 \\ 13 \\ 16 \end{bmatrix}, Ax + b = ? \tag{2}$$

Solution numpy : A.dot(x)+b

Résoudre un système d'équation linéaire avec autant d'équation que d'inconnue

Principe

Pour résoudre un système d'équation linéaire, on se ramène à un problème de la forme $Ax = y$, avec A une matrice carré dont les dimensions sont égales aux nombres d'inconnue, y un vecteur de valeurs connues et x le vecteur de valeurs recherchées.

Évaluation rapide d'un ensemble d'équation

Problème :

$$\begin{cases} 2x + 3y + 1 = 1 \\ -4x + 12z = 2 \\ 23x + 4y - 42z - 4 = 3 \end{cases} \tag{3}$$

Réécriture

$$A = \begin{bmatrix} 2 & 3 & 0 \\ -4 & 0 & 12 \\ 23 & 4 & 42 \end{bmatrix}, y = \begin{bmatrix} -1+1 \\ 0+2 \\ 4+3 \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \\ 7 \end{bmatrix}, Ax = y, x = ? \tag{4}$$

Solution numpy : x=np.linalg.solve(A,y)

Tout les systèmes d'équations linéaires non pas une solution

Systèmes d'équations sans solution

$$\left\{ \begin{array}{l} x + y = 1 \\ x + y = 2 \end{array} \right. \tag{5}$$

Réécriture

$$A = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, y = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, Ax = y, x = ? \tag{6}$$

Solution numpy : x=np.linalg.solve(A,y)

Traceback (most recent call last) :
File "<stdin>", line 1, in <module>
File "/usr/local/lib/python3.5/dist-packages/numpy/linalg/linalg.py", line 375, in solve
r = gufunc(a, b, signature=signature, extobj=extobj)
File "/usr/local/lib/python3.5/dist-packages/numpy/linalg/linalg.py", line 90, in
_raise_linalgerror_singular
raise LinAlgError("Singular matrix")
numpy.linalg.linalg.LinAlgError : Singular matrix

Résoudre un système d'équation linéaire à un nombre d'équation quelconque

Principe

Pour résoudre un système d'équation linéaire, on se ramène à un problème de la forme $Ax = y$, avec A une matrice, y un vecteur de valeurs connues et x le vecteur de valeurs recherchées.

Évaluation rapide d'un ensemble d'équation

Problème :

$$\begin{cases} 2x + 3y + 1 = 1 \\ -4x + 12z = 2 \\ 23x + 4y - 42z - 4 = 3 \\ 17z - 2 = 4 \end{cases} \tag{7}$$

Réécriture

$$A = \begin{bmatrix} 2 & 3 & 0 \\ -4 & 0 & 12 \\ 23 & 4 & 42 \\ 0 & 0 & 17 \end{bmatrix}, y = \begin{bmatrix} -1+1 \\ 0+2 \\ 4+3 \\ 2+4 \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \\ 7 \\ 6 \end{bmatrix}, Ax = y, x = ? \tag{8}$$

Solution numpy : x,_,_,_ = np.linalg.lstsq(A,y)

Résultat sur un système n'ayant pas de solution

Principe

La fonction np.linalg.lstsq recherche la solution la plus proche possible. Elle cherchent à minimiser l'écart entre Ax et y. Plus exactement elle retourne le résultat de $\arg\min_x \|Ax - y\|$.

Exemple

$$\begin{cases} x + y = 1 \\ x + y = 2 \end{cases} \tag{9}$$

Réécriture

$$A = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, y = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, Ax = y, x = ? \tag{10}$$

Solution numpy :

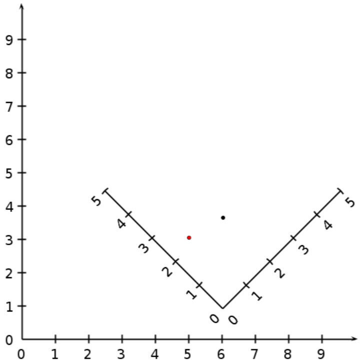
```
x,_,_,_ = np.linalg.lstsq(A,y)
print(x)
-> array([ 0.75, 0.75])
```

Exemple sur un changement de repère

Problème à résoudre

On dispose des coordonnées d'un point dans un repère et on les veut dans un autre repère.

Exemple



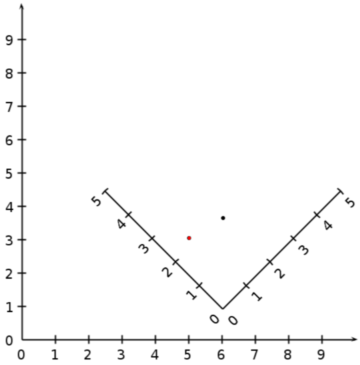
- Le deuxième repère est définis dans le premier repère par les vecteurs $[0.71; 0.71]$ et $[-0.71; 0.71]$ suivi d'une translation de $[6; 1]$.
- Le point noir a pour coordonnée dans le deuxième repère $[2; 2]$.
- Le point noir a donc pour coordonnée dans le premier repère $\begin{bmatrix} 0.71 & -0.71 \\ 0.71 & 0.71 \end{bmatrix} \begin{bmatrix} 2 \\ 2 \end{bmatrix} + \begin{bmatrix} 6 \\ 1 \end{bmatrix} = \begin{bmatrix} 6 \\ 3.84 \end{bmatrix}$

Exemple sur un changement de repère

Problème à résoudre

On dispose des coordonnées d'un point dans un repère et on les veux dans un autre repère.

Exemple



- Le point rouge a pour coordonnée dans le premier repère $[5; 3]$.
- Le point rouge a donc pour coordonnée dans le premier repère
$$\begin{bmatrix} 0.71 & -0.71 \\ 0.71 & 0.71 \end{bmatrix} \begin{bmatrix} ? \\ ? \end{bmatrix} + \begin{bmatrix} 6 \\ 1 \end{bmatrix} = \begin{bmatrix} 5 \\ 3 \end{bmatrix}$$
- Solution numpy :
np.linalg.solve(np.array([[0.71,-0.71],[0.71,0.71]]), np.array([5,3])-np.array([6,1]))
–> array([0.70422535, 2.11267606])

Sommaire

Zéro d'une fonction

Définition

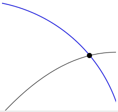
Soit f une fonction quelconque continue (par exemple une fonction python prenant un réel et retournant un réel), on définit les zéros de la fonction comme étant les valeurs de x tel que $f(x) = 0$.

Cas générale

Soit f une fonction continue, trouvez les valeurs de x tel que l'on est $f(x) = a$ (a étant connu) revient à trouver les zéros de la fonction $f(x) - a$. On peut donc toujours se ramener à une recherche des zéros d'une fonction.

Autres exemples d'utilisation : recherche des intersections de deux fonctions

Soit f et g deux fonctions continues. Trouver les x tel que $f(x) = g(x)$ revient à trouver les zéros de la fonction $f(x) - g(x) = 0$.



Comment trouver les zéros d'une fonction

De nombreuses solutions

- La littérature mathématique propose de nombreuse solution pour résoudre ce problème.
- Nous allons étudier trois solutions :
  1. L'approche par dichotomie,
  2. La méthode des sécantes,
  3. La méthode de Newton.
- Il faut garder à l'esprit qu'il existe de nombreuses autres solutions qui peuvent être meilleurs en fonction des problèmes.
- Fonction de résolution dans scipy : voir le paragraphe Root finding de `https://docs.scipy.org/doc/scipy/reference/optimize.html`.

L'approche par dichotomie

But

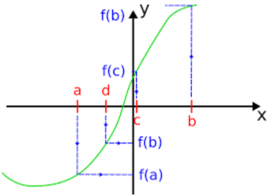
La méthode par dichotomie a pour objectif de trouver un zéro dans un intervalle de départ pour une fonction f **continue**. Une des extrémité de l'intervalle doit donner une valeur de la fonction supérieur à 0 et l'autre inférieur à 0.
On va supposer que l'intervalle est $[a, b]$ et que $f(a) < 0$ et $f(b) > 0$. Si ce n'est pas le cas, vous pouvez vous ramener à ce cas en prenant la fonction $-f(x)$ qui a les même zéros que $f(x)$.

Principe

On va itérativement couper l'intervalle en deux et on continuera avec l'intervalle contenant la solution.

Algorithme en pseudo code

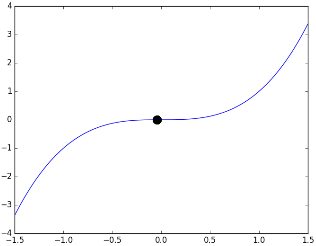
Tant que $(b - a) > \epsilon$
  Calculez $x = \frac{a+b}{2}$
  Calculez f(x)
  Si f(x) > 0 alors
    $b \longleftarrow x$
  sinon
    $a \longleftarrow x$



L'approche par dichotomie sous scipy

Code scipy

```
import scipy as sc
import scipy.optimize
def f(x):
    return x**3
print(sc.optimize.bisect(f,-1,1))
        -> 0.0
```



Code scipy en utilisant des lambda fonction python

```
import scipy as sc
import scipy.optimize
print(sc.optimize.bisect(lambda x:x**3,-1,1))
        -> 0.0
```

La méthode de la sécante

But

La méthode de la sécante à pour objectif de trouver un zéro d'une fonction f **continue**. Elle s'appuie sur deux points initiales a et b (proches si possible de la solution recherchée) mais n'encadrant pas nécessairement la solution.

L'idée

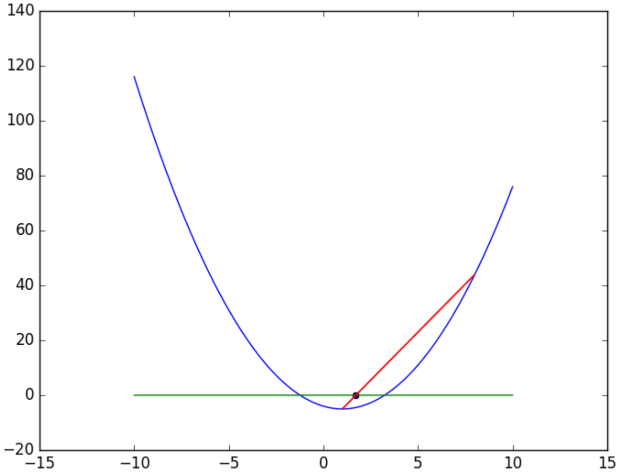
On va approximer la courbe par la droite passant par les points $(a,f(a))$ et $(b,f(b))$. En calculant l'intersection de cette droite avec l'axe des abscisses, on va mettre à jours l'un des points et recommencer l'approche.

L'algorithme

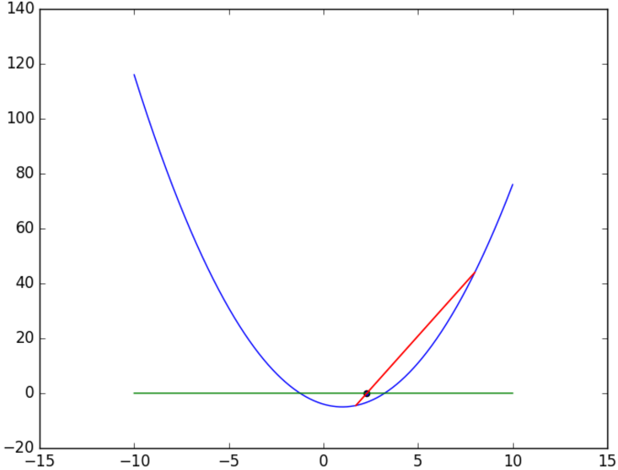
On calcul la suite :

$$\left\{\begin{array}{ccc} x_0 & = & a \\ x_1 & = & b \\ x_{n+1} & = & x_n - \dfrac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} f(x_n) \end{array}\right.$$

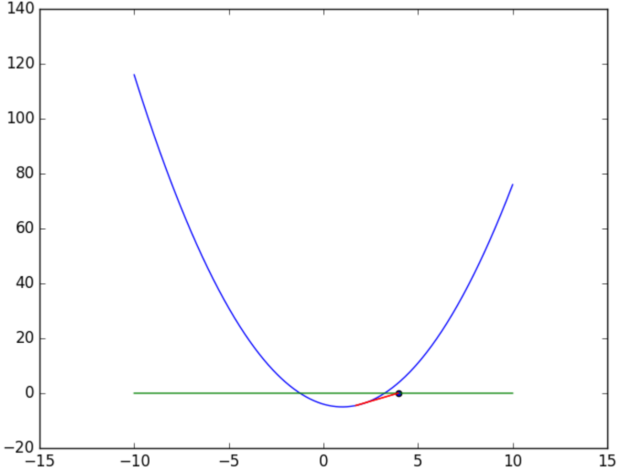
Exemple de la méthode de la sécante



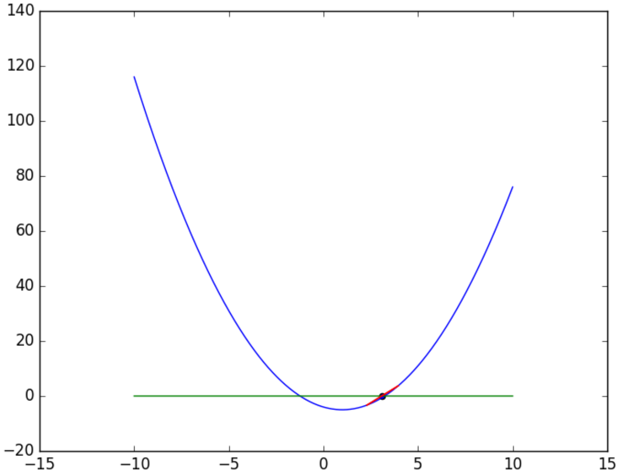
Exemple de la méthode de la sécante



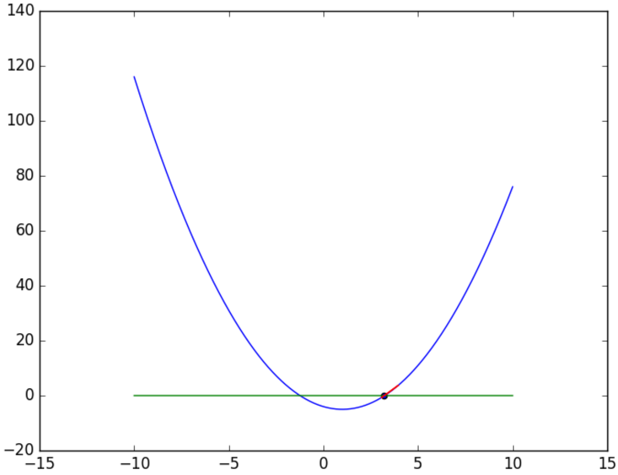
Exemple de la méthode de la sécante



Exemple de la méthode de la sécante



Exemple de la méthode de la sécante



Code scipy

Code scipy en utilisant des lambda fonction python

On utilise la fonction sc.optimize.newton avec les valeurs par défaut pour l'attribut fprime (=None). Les arguments sont la fonction à étudier et une valeur x autour du quel on cherche la solution. Scipy prendra le deuxième point très proche de x, plus exactement la valeur $(1 + 1e^{-4})x + \mathrm{sign}(x)1e^{-4}$.

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

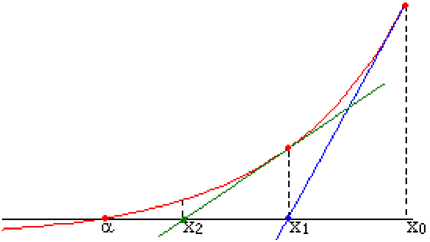
La méthode de Newton

But

La méthode de Newton à pour objectif de trouver un zéro d'une fonction f **continue dérivable**. Elle s'appuie sur un point initiale a (proche si possible de la solution recherchée).

L'idée

On va approximer la courbe par la tangente de la fonction. En calculant l'intersection de cette droite avec l'axe des abscisses, on va mettre à jours le point de référence et recommencer l'approche.



La méthode de Newton

Calcul

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

Démonstration

Équation de la tangente en $x^\star$:

$$f'(x^\star)x + f(x^\star) - f'(x^\star)x^\star$$

Intersection de la tangente avec l'axe des abscisses :

$$\begin{aligned}
f'(x^\star)x + f(x^\star) - f'(x^\star)x^\star &= 0 && (11)\\
f'(x^\star)x &= -f(x^\star) + f'(x^\star)x^\star && (12)\\
x &= \frac{-f(x^\star) + f'(x^\star)x^\star}{f'(x^\star)} && (13)\\
& && (14)
\end{aligned}$$

La méthode de Newton avec scipy

Code scipy en utilisant des lambda fonction python

On utilise la fonction sc.optimize.newton avec la fonction dérivée dans l'attribut fprime. Les arguments sont la fonction à étudier et une valeur x autour du quel on cherche la solution.

```
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
```