

Cours Info - 13

Résolution numérique d'équations algébriques et transcendantes

D.Malka

MPSI 2015-2016



Sommaire

- 1 Equations sur les réels
- 2 Méthode dichotomique
- 3 Méthode de Newton
- 4 Méthode de Newton vs méthode dichotomique
- 5 Les fonctions prédéfinies des bibliothèques scipy et numpy

Sommaire

- 1 Equations sur les réels
- 2 Méthode dichotomique
- 3 Méthode de Newton
- 4 Méthode de Newton vs méthode dichotomique
- 5 Les fonctions prédéfinies des bibliothèques scipy et numpy

Equation sur les réels

Types d'équation que l'on peut-être amené à résoudre :

Equations algébriques

$$x^2 - 3x^3 + 1 = 2x$$

$$x^2 + y^2 - 6xy + 3x = 0$$

Equations transcendantes

$$\cos(x) = x$$

$$e^x(x + e^x) = 3x^2 + 2$$

Résolution numérique

Qu'est-ce qui justifie une résolution numérique donc approchée ?

- ▶ **Les solutions existent mais ne peuvent être écrites analytiquement.**
- ▶ **Il existe des solutions analytiques mais le calcul est fastidieux.**
- ▶ **On souhaite tester facilement l'influence d'un paramètre sur la solution d'une équation.**

Se ramener à la recherche du zéro d'une fonction

Ecriture générale d'une équation

On peut toujours écrire une équation sous la forme :

$$f(x) = 0$$

Exemple

En vue de sa résolution, l'équation :

$$e^x = x + 5$$

sera écrite :

$$e^x - x - 5 = 0$$

soit encore :

$$f(x) = 0 \quad \text{avec} \quad f(x) = e^x - x - 5$$

Sommaire

- 1 Equations sur les réels
- 2 Méthode dichotomique**
- 3 Méthode de Newton
- 4 Méthode de Newton vs méthode dichotomique
- 5 Les fonctions prédéfinies des bibliothèques scipy et numpy

Principe de la recherche dichotomique du zéro d'une fonction

On cherche une racine x de l'équation $f(x) = 0$.

Conditions d'application

Soit $[a, b]$ l'intervalle de recherche.

- ▶ f continue sur $[a, b]$,
- ▶ f change de signe sur $[a, b]$ ($f(a)f(b) < 0$),
- ▶ théorème des valeurs intermédiaires \Rightarrow existence d'au moins un zéro de f dans $[a, b]$.

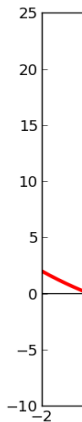
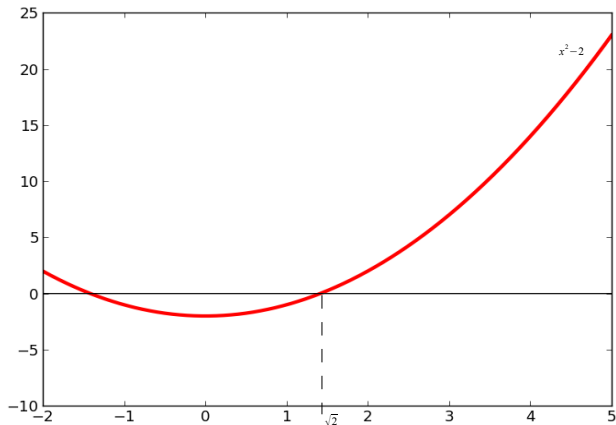
Principe de la recherche dichotomique du zéro d'une fonction

Algorithme

- ▶ On coupe en deux récursivement l'intervalle de recherche courant $[a_k, b_k]$ en maintenant l'invariant $f(a_k).f(b_k) < 0$,
- ▶ A chaque itération $x \in [a_k, b_k]$,
- ▶ **Condition d'arrêt** : soit ε l'erreur absolue maximale tolérée. On arrête l'algorithme lorsque $b_k - a_k < \varepsilon$.

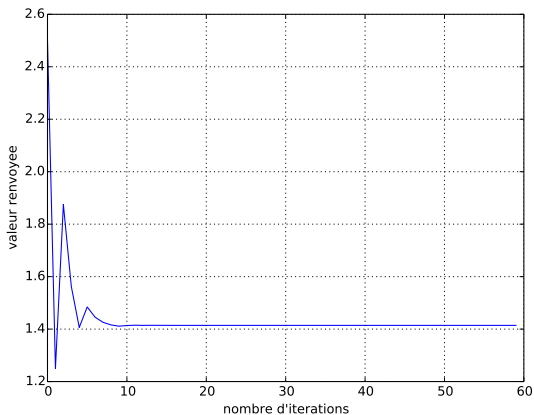
Principe de la recherche dichotomique du zéro d'une fonction

Représentation graphique



Principe de la recherche dichotomique du zéro d'une fonction

Graphiquement :



Algorithme (en Python)

Implémentation en Python :

```
1 def zero_dichotomique(f, a, b, epsilon):
2     g, d = a, b
3     while (d - g) > epsilon:
4         m = (g + d) / 2
5         if f(g) * f(m) < 0: # m sup a la racine
6             d = m
7         elif f(g) * f(m) > 0: # m inf a la racine
8             g = m
9         else: # coup de chance f(m) = 0
10            return m
11    return (g + d) / 2
```

Terminaison

Terminaison

Montrons que l'algorithme termine toujours.

- ▶ Soit $u_k = (b_k - a_k)$
- ▶ Par récurrence : $u_k = \frac{(b-a)}{2^k}$
- ▶ Donc $u_k \searrow$ et $\lim_{k \rightarrow +\infty} u_k = 0$
- ▶ Donc $\forall \varepsilon > 0, \exists n_0 \in \mathbb{N}$ tel que $u_{n_0} < \varepsilon \dots$
- ▶ ... i.e l'algorithme termine.

Correction

Preuve de la méthode dichotomique

- ▶ $f(a_k)f(b_k) < 0$ est un invariant de boucle (démonstration par récurrence)
- ▶ $\forall k, \exists$ un zéro de f dans $[a_k, b_k]$
- ▶ En particulier, en sortie de boucle $x_0 \in [a_n, b_n]$ avec $b_n - a_n < \varepsilon$ soit $x_0 - \varepsilon < x < x_0 + \varepsilon$

Complexité

La complexité de la méthode dichotomique est $O\left(\log_2\left(\frac{b-a}{\varepsilon}\right)\right)$.

Démonstration

- ▶ n : nombre d'itérations effectuées à la sortie de la boucle.
- ▶ or $(b_n - a_n) \leq \varepsilon$
- ▶ avec $(b_n - a_n) = \frac{b_0 - a_0}{2^n}$
- ▶ soit $\frac{b_0 - a_0}{2^n} \leq \varepsilon$
- ▶ Donc au plus : $n = \log_2\left(\frac{b-a}{\varepsilon}\right)$

Convergence

Convergence globale

La convergence globale de la méthode dichotomique est assurée par le théorème des valeurs intermédiaires.

Ordre de convergence

Soit x_n la suite des valeurs approchées du zéro \bar{x} de f .

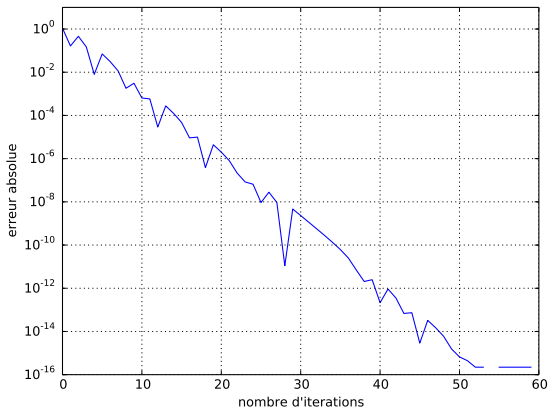
A partir d'un certain rang n_0 ,

$$(x_{n+1} - \bar{x}) < \frac{1}{2}(x_n - \bar{x})$$

La convergence de la méthode dichotomique est linéaire.

Convergence

Graphiquement, sur l'exemple $f(x) = 0$ avec $f : x \rightarrow x^2 - 2$.



Sommaire

1 Equations sur les réels

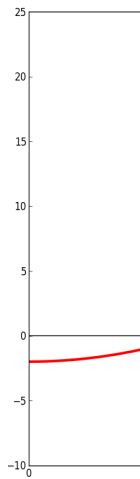
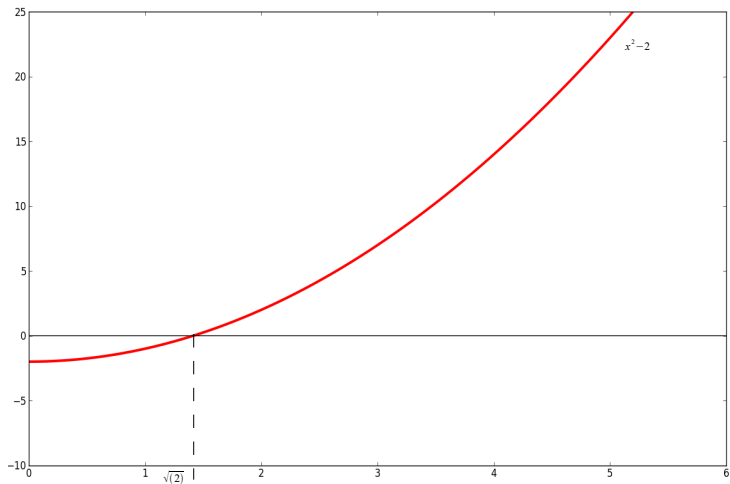
2 Méthode dichotomique

3 Méthode de Newton

4 Méthode de Newton vs méthode dichotomique

5 Les fonctions prédéfinies des bibliothèques scipy et numpy

Extraction de racine par la méthode de Newton



Extraction de racine par la méthode de Newton

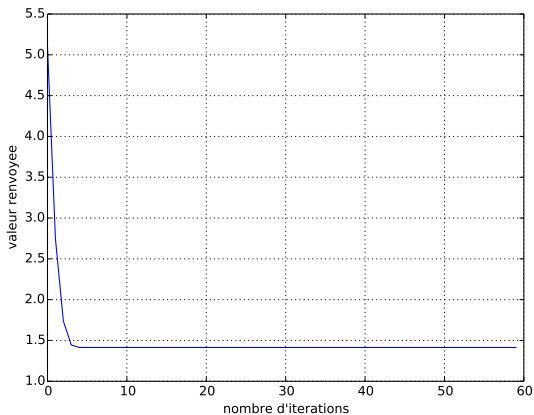
Si on note u_n les abscisses des points successifs où l'on évalue la tangente à la courbe, on a :

- ▶ $u_0 = 5$
- ▶ $u_1 = 2,7$
- ▶ $u_2 = 1,72$
- ▶ $u_3 = 1,441395349$
- ▶ $u_4 = 1,4144695859$

On sent que la suite u_n va converger vers $\sqrt{2} \approx 1,414213562\dots$ qui est la racine positive de $x^2 - 2$.

Extraction de racine par la méthode de Newton

Graphiquement :



Extraction de racine par la méthode de Newton

Principe

- ▶ On approche f par la tangente à sa courbe en C_f en u_0 i.e :

$$f(x) = f(u_0) + f'(u_0)(x - u_0) + O(x - u_0)$$

- ▶ On approxime le zéro \bar{x} de f par celui de sa tangente, u_1
- ▶ On réitère la méthode à partir de u_1
- ▶ On construit ainsi la suite $(u_n)_n$ des zéros des tangentes à C_f en u_n
- ▶ On espère que la suite converge vers \bar{x} !

Conditions d'application

Nécessaires mais non suffisantes !

- ▶ Connaître f ,
- ▶ Connaître f' .

Algorithme (en Python)

Implémentation en Python.

```
1 def newton(f, df, u_0, epsilon):
2     #f(u_n)=f'(u_n)*(u_n-u_{n+1})
3     u=u_0 #u contient u_n
4     v=u-f(u)/df(u) #v contient u_{n+1}
5     while fabs(u-v)>epsilon:
6         u=v
7         v=u-f(u)/df(u)
8     return v
```

Terminaison et correction

L'algorithme termine-t-il ? Si oui, est-il exact ?

La convergence globale de l'algorithme de Newton n'est pas assurée !

L'algorithme peut ne pas s'arrêter, diverger, renvoyer une valeur erronée, planter. . .

Critère de convergence approximatif

C'est la concavité/convexité locale de la courbe qui détermine la convergence de la méthode. Si f est de classe C^2 avec $f^{(2)}(\bar{x}) \neq 0$ et que le point de départ u_0 est assez proche de \bar{x} alors la concavité est constante au voisinage de \bar{x} et la méthode de Newton converge vers \bar{x} .

Ordre de convergence

Ordre de convergence

Soit u_n la suite des valeurs approchées du zéro \bar{x} de f .

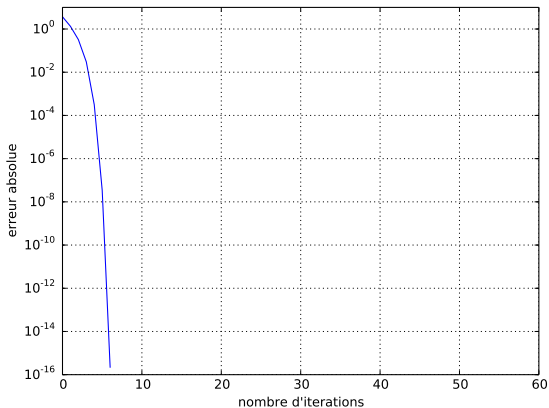
A partir d'un certain rang n_0 ,

$$(u_{n+1} - \bar{x}) < C(u_n - \bar{x})^2$$

La convergence de la méthode de Newton est quadratique.

Ordre de convergence

Graphiquement, sur l'exemple $f(x) = 0$ avec $f : x \rightarrow x^2 - 2$.



Sommaire

- 1 Equations sur les réels
- 2 Méthode dichotomique
- 3 Méthode de Newton
- 4 Méthode de Newton vs méthode dichotomique**
- 5 Les fonctions prédéfinies des bibliothèques scipy et numpy

Newton vs dichotomie

Quelle méthode vaut-il mieux utiliser ? Cela dépend du critère prépondérant entre rapidité d'exécution et robustesse.

	Dichotomie	Newton
Robustesse	++	-
Rapidité	+	+++

Pour résumer, si la fonction est « sympathique » et qu'on a une bonne idée du zéro de la fonction on pourra appliquer la méthode de Newton en partant d'un point proche du zéro. La recherche du point proche du zéro pour la méthode de Newton peut se faire par la méthode dichotomique ! Sinon on utilise la méthode dichotomique.

Attention aussi aux erreurs numériques si on cherche une grande précision !

Sommaire

- 1 Equations sur les réels
- 2 Méthode dichotomique
- 3 Méthode de Newton
- 4 Méthode de Newton vs méthode dichotomique
- 5 Les fonctions prédéfinies des bibliothèques scipy et numpy**

Fonction prédéfinies

Python dispose de bibliothèques de fonctions scientifiques prédéfinies :

- ▶ même principe que les fonctions « maison » précédente,
- ▶ + optimisation.

On utilisera donc ces fonctions pour résoudre des problèmes.

Racines d'un polynôme

Le module `roots` de la bibliothèque `numpy` détermine les racines dans \mathbb{C} d'un polynôme.

```
1 import numpy as np
2
3 P1=[1,0,-2]#x**2-2
4 racines1=np.roots(P1)
```

OUTPUT : [1.41421356 -1.41421356]

```
1 P2=[1,1,1]#x**2+x+1
2 racines2=np.roots(P2)
```

OUTPUT : [-0.5+0.8660254j -0.5-0.8660254j]

Dichotomie

Le module `optimize` de la bibliothèque `scipy` propose la fonction `bisect` qui implémente la méthode dichotomique.

```
1 import numpy as np
2 import scipy.optimize as op
3
4 f=lambda x: np.cos(x)
5 x0=op.bisect(f,0,np.pi,xtol=0.01)
```

OUTPUT : 1.5707963268 soit $\frac{\pi}{2}$

```
1 f=lambda x: np.cos(x)
2 x0=op.bisect(f,0,9*np.pi)
```

OUTPUT : 20.35203522483 soit $\frac{13\pi}{2}$

Méthode de Newton

Le module `optimize` de la bibliothèque `scipy` propose la fonction `newton` qui implémente la méthode de Newton.

```
1 import numpy as np
2 import scipy.optimize as op
3
4 f=lambda x: np.cos(x)#f
5 df=lambda x: -np.sin(x)#f'
6
7 x0=op.newton(f,np.pi/3,df)
```

OUTPUT : 1.57079632679 soit $\frac{\pi}{2}$

```
1 x0=op.newton(f,np.pi/100,df)
```

OUTPUT : 32.9867228627 soit $\frac{21\pi}{2}$

Méthode de Newton

Sans rentrer la dérivée (méthode de la sécante !) :

```
1 | x0=op.newton(f,np.pi/3)
```

OUTPUT : 1.57079632679 soit $\frac{\pi}{2}$. Aussi précis qu'avec la dérivée.

```
1 | x0=op.newton(f,np.pi/100)
```

OUTPUT : raise RuntimeError(msg)
RuntimeError: Failed to converge after 50 iterations, value is
17603867.0337

Méthode de Newton

Le module `optimize` de la bibliothèque `scipy` propose la fonction `fsolve` qui recherche les zéros d'une fonction scalaire ou vectorielle.

```
1 import numpy as np
2 import scipy.optimize as op
3
4 f=lambda (x,y): (x+y**2,1-y+x*y)#f
5 x0=op.fsolve(f, (0,0))
```

OUTPUT : [-0.356557123 0.6823278]