Simulation Numérique 3 :

Problème stationnaire à une dimension

Le but de ce chapitre est d'étudier la résolution d'un problème stationnaire à une dimension, linéaire ou non, conduisant à la résolution approchée d'une équation algébrique ou transcendante.

En d'autre terme, on veut trouver une solution à l'équation

$$f(x) = 0$$

où x est une variable réelle.

Les deux algorithmes au programme sont la méthode de dichotomie et la méthode de Newton.

I MÉTHODE DE DICHOTOMIE

1 Principe

On veut déterminer une valeur approchée d'un zéro d'une fonction f continue sur un segment $[a,b]$ ($a<b$) telle que $f(a)f(b)<0$.

Le théorème des valeurs intermédiaire nous assure l'existence d'un zéro $\alpha \in]a,b[$, que l'on supposera unique [1].

Le principe de la méthode est le suivant :

- On pose $a_0 = a$ et $b_0 = b$.
- Puis on construit par récurrence une suite $([a_n,b_n])_n$ de segments en appliquant l'algorithme suivant :
  On partage le segment $[a_n,b_n]$ en deux segments de même longueur à l'aide de $c_n = \dfrac{a_n+b_n}{2}$.
  - ⋆ Si $f(a)$ et $f(c_n)$ sont de même signe, on pose $a_{n+1} = c_n$ et $b_{n+1} = b_n$.
  - ⋆ Sinon, on pose $a_{n+1} = a_n$ et $b_{n+1} = c_n$.

2 Propriétés

On a facilement les propriétés suivantes : pour tout $n \in \mathbb{N}$,

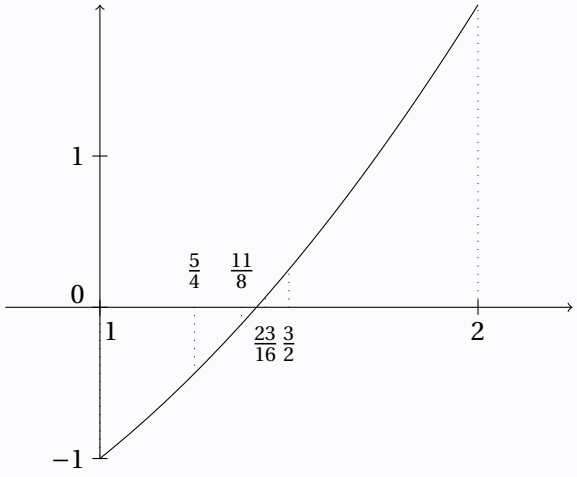
(i) $\alpha \in]a_n,b_n[$ intervalle de longueur $\dfrac{b-a}{2^n}$,

(ii) Les deux suites $(a_n)_n$ et $(b_n)_n$ sont adjacentes et ont pour limite commune α,

(iii) a_n (respectivement b_n) est une valeur approchée par défaut (respectivement par excès) de α à $\dfrac{b-a}{2^n}$ près, tandis que c_n en est une valeur approchée à $\dfrac{b-a}{2^{n+1}}$ près.

1. S'il n'est pas unique, alors l'algorithme permet d'approcher l'un des zéros de la fonction. Il suffit de se restreindre à un segment plus proche de l'un des zéros pour qu'il devienne le seul.

Exemple : Valeurs approchées de $\sqrt{2}$ par dichotomie avec $f(x) = x^2 - 2$



3 Algorithme

a Itératif

```
def dichotomie_n (f, a, b, n):
    """renvoie une valeur approchée du zéro de f entre a et b par
dichotomie au bout de n étapes."""
    A = a
    B = b
    C = (A + B) / 2
    for k in range(n):
        if f(A) * f(C) > 0:
            A = C
        else:
            B = C
        C = (A + B) / 2
    return C

def dichotomie_eps (f, a, b, epsilon):
    """renvoie une valeur approchée du zéro de f entre a et b par
dichotomie à epsilon près."""
    A = a
    B = b
    C = (A + B) / 2
    while abs(B - A) > 2 * epsilon:
        if f(A) * f(C) > 0:
            A = C
        else:
            B = C
        C = (A + B) / 2
    return C
```

Remarque

⚠ On n'aurait pas pu écrire

```
if f(A) * f(C) < 0:
    B = C
else:
    A = C
```

Que se passe-t-il alors si `f(A)` vient à s'annuler au cours de l'exécution ?
On aurait aussi pu tester si `f(C) == 0` et renvoyer `C` dans ce cas, avec toute la prudence que peut nécessiter un test d'égalité entre des flottants.

Invariant : à la fin de l'étape k, A, B, C contiennent respectivement a_k, b_k, c_k, $f(a)$. On a toujours $f(A)f(B) \leqslant 0$.

Terminaison : pour le second (boucle while), l'étude mathématique donne $b_k - a_k \to 0$. Mais attention à ne pas dépasser la précision machine ! ($\approx a \times 10^{-16}$ en double précision)

Correction : l'étude mathématique et l'invariant permettent de justifier que le résultat renvoyé par l'algorithme est celui attendu : une approximation de α à ε près. Sous réserves évidemment que f vérifie les bonnes hypothèses de départ !

Complexité (appel de f) : $T(n) \sim n$, $T(\varepsilon) = \mathrm{O}\left(\log \varepsilon\right)$. Complexité spatiale constante.

b Récursif

```
def dicho_rec(f, a, b, n):
    """renvoie une valeur approchée du zéro de f entre a et b par
dichotomie au bout de n étapes."""
    c = (a + b) / 2
    if n == 0:
        return c
    elif f(a) * f(c) > 0:
        return dicho_rec(f, c, b, n - 1)
    else:
        return dicho_rec(f, a, c, n - 1)

def dicho_rec_eps(f, a, b, epsilon):
    """renvoie une valeur approchée du zéro de f entre a et b par
dichotomie à epsilon près."""
    c = (a + b) / 2
    if abs(b - a) <= 2 * epsilon :
        return c
    elif f(a) * f(c) > 0:
        return dicho_rec(f, c, b, epsilon)
    else:
        return dicho_rec(f, a, c, epsilon)
```

4 Vitesse de convergence

L'étude mathématique donne $\boxed{|c_n - \alpha| \leqslant \frac{b-a}{2^{n+1}}.}$

La convergence est donc exponentielle. On ajoute un chiffre significatif (en base 2) à chaque étape.

II MÉTHODE ITÉRATIVES

1 Principe

L'idée est de partir d'une valeur approchée grossière de la solution et d'en améliorer la précision par une application itérée d'un algorithme bien choisi.

L'idée est de transformer notre zéro en point fixe d'une fonction g et de créer une suite récurrence $x_{n+1} = g(x_n)$ qui, sous certaines conditions, convergera vers le point fixe, c'est-à-dire notre zéro de f.

Les mathématiques nous disent qu'une bonne condition pour cela est que la fonction g soit **contractante** [1], c'est-à-dire k-lipschitzienne [2] avec $k \in [0,1[$, c'est-à-dire telle que

$$\forall\, x, x' \in [a,b],\quad \left|g(x) - g(x')\right| \leqslant k\left|x - x'\right|.$$

On obtient en plus dans ce cas une estimation de la vitesse de convergence :

$$|u_n - \alpha| \leqslant k^n\,|u_0 - \alpha|$$

Pour se ramener à un problème de point fixe, il suffit de poser :

$$g(x) = x - \lambda(x) f(x)$$

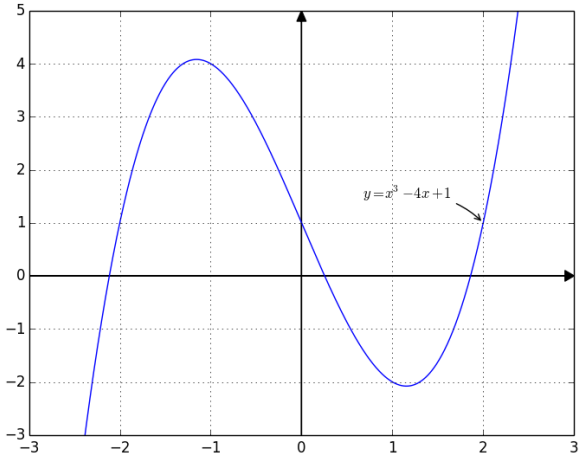
avec $\lambda(x)$ bien choisi. S'il ne s'annule pas, on vérifie sans mal que

$$f(x) = 0 \Longleftrightarrow g(x) = x.$$

2 Un exemple

On veut résoudre numériquement l'équation $f(x) = x^3 - 4x + 1 = 0$ sur $\mathbb{R}$. On étude mathématique rapide donne la courbe :

1. et on démontre que dans ce cas, le point fixe est unique.
2. On utilise en général pour les fonctions suffisamment régulières, l'inégalité des accroissement finis ($\left|f'\right| \leqslant k$...)



Une rapide étude donne trois zéros :

$$-2,5 < \alpha_1 < -2, \qquad 0 < \alpha_2 < 0,5, \quad , \quad 1,5 < \alpha_3 < 2$$

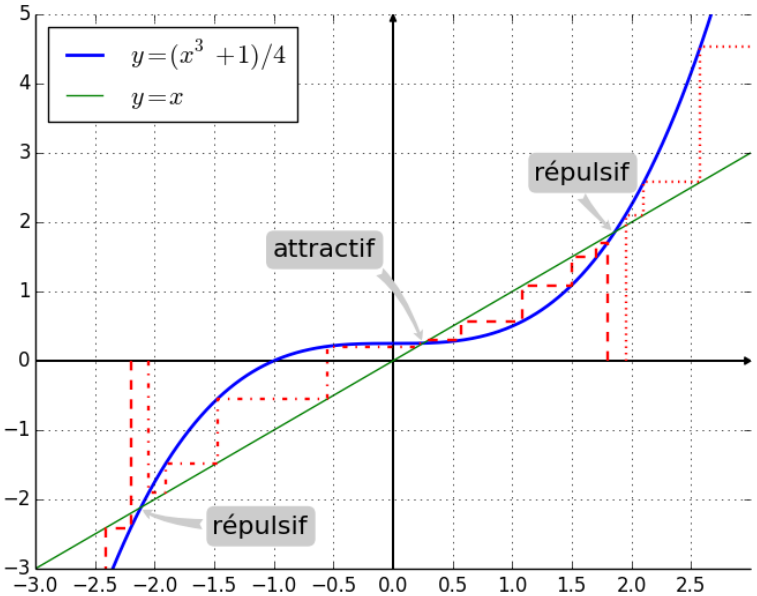
Or $f(x) = 0 \iff g(x) = x$ avec $g(x) = \frac{1}{4}(x^3+1)$. Comme $g'(x) = \frac{3}{4}x^2$, on obtient :

- Sur $[-2,5;-2]$, $g' \geqslant 3$: α_1 point fixe répulsif.
- Sur $[1,5;2]$, $g' \geqslant 1,6875$: α_3 point fixe répulsif.
- Sur $[0;0,5]$, $0 \leqslant g' \leqslant 0,1875$: point fixe attractif. On peut utiliser une suite récurrente $u_{n+1} = g(u_n)$ sur l'intervalle $[0;0,5]$ stable par g pour approcher numériquement α_2. La convergence est plus rapide qu'avec la dichotomie ($0,1875 < \frac{1}{2}$).

Pour approcher α_1 et α_3, on peut par exemple itérer la fonction :

$$g^{-1}(x) = \sqrt[3]{4x-1}$$

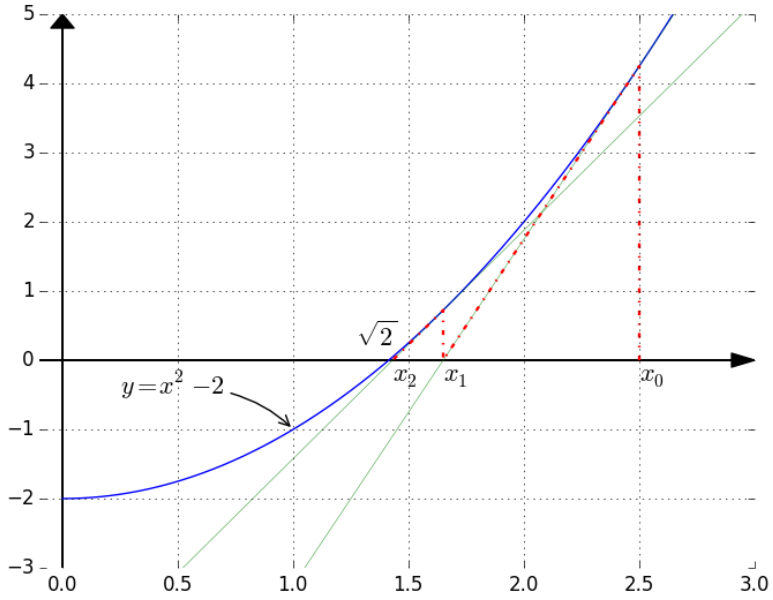
qui est bien (évidemment !) contractante au voisinage de ces points.



Il existe en fait une méthode plus générale et plus systématique.

3 Méthode de Newton

Le principe est le suivant : connaissant une valeur approchée grossière x_0 du zéro α d'une fonction f vérifiant certaines hypothèses à préciser, on va définir une suite récurrente en partant de $(x_0, f(x_0))$ et en définissant x_1 comme l'intersection de l'axe des abscisse avec la tangente [1] à f en x_0, puis en itérant le procédé.



L'équation de la tangente en x_n étant

$$y = f'(x_n)(x - x_n) + f(x_n)$$

on obtient : [2]

$$\boxed{x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}}$$

Les mathématiques nous donnent la

Propriété

Soit f est de classe $\mathscr{C}^2$ sur le segment I et f' ne s'annule pas sur I, et $M = \max_{x\in I} \left|\frac{f''(x)}{f'(x)}\right|$.
Alors f ne peut avoir qu'un seul zéro α sur I, et si $|x_0 - \alpha| < \frac{1}{M}$, pour tout n,

$$|x_n - \alpha| \leqslant \frac{1}{M}\left(M|x_0 - \alpha|\right)^{2^n} (\to 0)$$

En d'autres termes, si x_0 est suffisamment proche de α, $(x_n)_n$ converge diaboliquement vite vers α : c'est bi-exponentiel !

À chaque étape on va **doubler** le nombre de chiffres significatifs !

On parle de **convergence quadratique**.

1. qui est la droite qui approche le mieux $\mathscr{C}_f$ au voisinage de x_0.
2. Expression de la forme $x_{n+1} = g(x_n)$ où $g(x) = x - \lambda(x)f(x)$ avec $\lambda(x) = 1/f'(x)$. Comme $g'(\alpha) = 0$, le point fixe α est dit **superattractif**.

Exemple

L'algorithme de Babylone permet de calculer la valeur approchée d'une racine carrée par la méthode de Newton. On cherche à calculer numériquement $\sqrt{a}$ où $a > 1$. Pour cela, on introduit la fonction $f : x \mapsto x^2 - a$. Avec x_0 assez proche de $\sqrt{a}$, on obtient :

$$x_{n+1} = x_n - \frac{x_n^2 - a}{2x_n} = \frac{x_n^2 + a}{2x_n} = \frac{1}{2}\left(x_n + \frac{a}{x_n}\right).$$

Pour $\sqrt{2}$ (méthode de Héron), par exemple, on peut se placer sur $[1, \frac{3}{2}]$.

Comparatif : Nombre d'étapes maximum nécessaires au calcul de $\sqrt{2}$ avec la dichotomie, $k = 0,1$ pour le point fixe contractant et la méthode de Newton.

ε	0.1	0.01	0.001	10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{-8}	10^{-9}	10^{-10}
dichotomie	3	6	9	13	16	19	23	26	29	33
k=0,1	1	2	3	4	5	6	7	8	9	10
Newton théorique	2	2	3	3	5	5	5	5	5	5
Newton réel	1	1	2	2	2	3	3	3	3	3

Encore plus parlant, pour l'algorithme de Newton :

n	1	2	3	4	5	6	7	8	9
nb th. de déc. correctes	0	2	4	9	18	38	76	153	307
nb réel de décimales	2	5	11	24*	48*	97*	195*	391*	783*

* Avec la norme IEEE754, on ne peut pas dépasser 16 décimales pour les flottants. J'ai dû utiliser le module `decimal` pour m'affranchir de cette limite.

En itératif :

```
def newton(f, df, x0, n):
    """renvoie le terme d'indice n de la méthode de Newton approchant
un zéro de f de dérivée df, de premier terme x0."""
    x = x0
    for k in range(n):
        x -= f(x) / df(x)
    return x

def newton_eps(f, df, x0, epsilon):
    """renvoie le premier terme x[n] de la méthode de Newton
approchant  un zéro de f de dérivée df, de premier terme x0
tel que |x[n] - x[n - 1]| <= epsilon."""
    x = x0
    y = x - f(x) / df(x)
    while abs(y - x) > epsilon:
        x = y
        y = x - f(x) / df(x)
    return y
```

⚠ La **terminaison** de la boucle while n'est assurée que si les conditions sur f et x_0 de la propriété précédente sont vérifiées. De plus, si $f'(x)$ est trop petit, on risque la division par zéro... En récursif :

```
def newton_rec(f, df, x0, n):
    """renvoie le terme d'indice n de la méthode de Newton approchant
un zéro de f de dérivée df, de premier terme x0."""
    if n == 0:
        return x
    else:
        y = x - f(x) / df(x)
        return newton_rec(f, df, y, n - 1)

def newton_rec_eps(f, df, x0, epsilon):
    """renvoie le premier terme x[n] de la méthode de Newton
approchant  un zéro de f de dérivée df, de premier terme x0
tel que |x[n] - x[n - 1]| <= epsilon."""
    y = x - f(x) / df(x)
    if abs(y - x) <= epsilon:
        return y
    else:
        return newton_rec_eps(f, df, y, epsilon)
```

Le gros inconvénient de ce merveilleux algorithme est qu'il faut lui fournir la fonction f'.

Comme s'y prend-on quand on ne l'a pas sous la main ?

4 Méthode de la sécante (ou fausse position, HP)

Lorsque l'on est pas capable de fournir ou de calculer f', l'idée est de la remplacer par un taux d'accroissement sur un intervalle suffisamment petit.

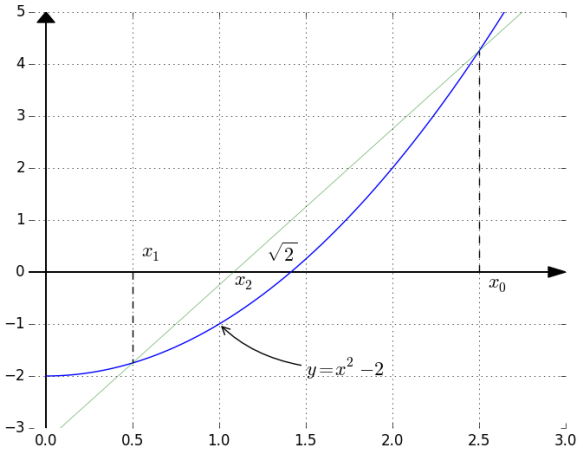
Si l'on connaît déjà deux valeurs approchés x_0 et x_1 de α, alors on effectue la même opération que pour la méthode de Newton en remplaçant la tangente par la sécante, c'est-à-dire la droite passant par $\big(x_0, f(x_0)\big)$ et $\big(x_1, f(x_1)\big)$.

Cela revient à écrire $f'(x_0) \approx \dfrac{f(x_1)-f(x_0)}{x_1-x_0}$.

D'où, en itérant :

$$x_{n+1} = x_n - \frac{f(x_n)}{\tau_n} \text{ avec } \tau_n = \frac{f(x_n)-f(x_{n-1})}{x_n-x_{n-1}}$$

Inconvénient : Lorsque x_{n-1} et x_n sont trop proches, le calcul des différences dans τ_n implique une forte perte de précision. Donc, dans la pratique, dès que $|x_n - x_{n-1}| < \sqrt{\varepsilon_0}$ où ε_0 est la précision absolue [1], on prend $\tau_n = \tau_{n-1}$.



1. précision machine × ordre de grandeur de $\alpha \approx 10^{-16} \times x_0$ en double précision.

```
def secante(f, x0, x1, epsilon):
    x = x0
    y = x1
    while abs(y - x) > epsilon:
        df = (f(x) - f(y)) / (x - y)
        x = y
        y = y - f(y) / df
    return y
```

```
def secante_bis(f, x0, x1, epsilon):
    x = x0
    y = x1
    precision = x0 ** .5 * 1e-8
    df = (f(x) - f(y) / (x - y)
    while abs(y - x) > epsilon:
        x = y
        y = y - f(y) / df
        if abs(y - x) > precision:
            df = (f(x)-f(y)) / (x-y)
    return y
```

Une étude mathématique donne que si x_0 et x_1 sont suffisamment proches de α, alors pour tout n, $|x_n - \alpha| \leqslant C \times k^{F_{n+1}}$ où $0 \leqslant k < 1$ et (F_n) est la suite de Fibonacci, qui, comme chacun sait, est équivalente à $\frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n$. Comme $\dfrac{1+\sqrt{5}}{2} \approx 1,618$, c'est tout juste un peu moins rapide que la méthode de Newton.

5 Avec plusieurs variables ?

On peut adapter la méthode de Newton pour une fonction f : $\left|\begin{array}{ccc} \mathbb{R}^m & \longrightarrow & \mathbb{R}^m \\ X = \begin{pmatrix} x_1 \\ \vdots \\ x_m \end{pmatrix} & \longmapsto & Y = f(X) = \begin{pmatrix} f_1(X) \\ \vdots \\ f_m(X) \end{pmatrix} \end{array}\right.$

L'idée est la même : à partir d'une valeur approchée grossière [1] d'une solution de $f(X) = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}$, itérer une fonction définie par

$$X_{n+1} = X_n - \left(J_f(X_n)\right)^{-1} f(X_n)$$

où $X_n = \begin{pmatrix} x_{n,1} \\ \vdots \\ x_{n,k} \end{pmatrix}$ et où J_f est la **matrice jacobienne** de f définie par

$$J_f = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_m} \end{pmatrix}.$$

On peut démontrer mathématiquement que si f est de classe $\mathscr{C}^2$, et A un point de $\mathbb{R}^m$ tel que $f(A) = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}$ et telle que la matrice $J_f(A)$ est inversible, alors A est un point fixe superattractif.

Le problème est qu'il va falloir trouver une valeur approchée grossière de A et à chaque itération calculer $J_f(X_n)$ et l'inverser ce qui n'est pas sans poser quelques problèmes numériques (voir cours sur l'algorithme du pivot de Gauss).

On peut éviter l'inversion en résolvant le système $J_f(X_n)\,(X_n - X_{n+1}) = f(X_n)$ (ce qui peut aussi poser des problèmes numériques !)

1. ma non tropo.