Chapitre $\underline{1100}_2$
Résolution numérique d'équations

2 janvier 2018

Cadre : soit $f : I \to \mathbb{R}$, avec I un intervalle de $\mathbb{R}$.
On cherche une approximation d'un $x \in I$ tel que $f(x) = 0$, *i.e.* d'une solution de l'équation $f(x) = 0$ (on parle aussi de *zéro* de la fonction f).

1 Méthode dichotomique

1.1 Principe

On suppose que f est continue.

— On part de a et b ($a < b$) tels que $f(a)$ et $f(b)$ soient de signes contraires. La TVI assure alors qu'il y a un point d'annulation de f entre a et b.

— On calcule $m = \frac{1}{2}(a + b)$.

— On itère en repartant de a et m ou de m et b en fonction du signe de $f(m)$.

Pour savoir si on repart de a et m ou de m et b :

— On regarde le signe de $f(a)f(m)$.

— Si $f(a)f(m) > 0$: $f(a)$ et $f(m)$ sont de même signe, donc $f(m)$ et $f(b)$ sont de signes opposés.

— Si $f(a)f(m) < 0$: $f(a)$ et $f(m)$ sont de signes opposés.

— Si $f(a)f(m) = 0$: $f(a)$ ou $f(m)$ est nul, on repart de a et m (on pourrait renvoyer directement a ou m, mais, d'une part, dans la pratique, on ne tombe jamais exactement sur un point d'annulation et, d'autre part, à cause des erreurs de manipulations des flottants, un flottant n'est jamais vraiment égal à 0).

Invariant de l'algorithme

Invariant de l'algorithme : à chaque itération, $f(a)$ et $f(b)$ sont de signes opposés (donc d'après le TVI il y a un point d'annulation de f entre a et b). C'est équivalent à : $f(a)f(b) \leq 0$.

Terminaison

On se donne un critère d'arrêt de boucle : on va demander à obtenir une valeur approchée du résultat à une précision $\varepsilon > 0$ près.

Pour un tel résultat à une précision $\varepsilon > 0$, on s'arrête quand $b - a \leq 2\varepsilon$ et on renvoie $\frac{a+b}{2}$. On obtient bien une valeur approchée à $\varepsilon > 0$ près.

Comme l'écart $b - a$ suit une progression géométrique de raison $\frac{1}{2}$, donc tend vers 0, l'algorithme s'arrête (après un nombre fini d'itérations).

1.2 Implantation

```
def dichotomie(f, a, b, epsilon):
    """Zéro de f sur [a,b] à epsilon près, par dichotomie
       Préconditions : f(a) * f(b) <= 0
                       epsilon > 0"""
    c, d = a, b
    fc, fd = f(c), f(d)
    while d - c > 2 * epsilon:
        m = (c + d) / 2.
        fm = f(m)
        if fc * fm <= 0:
            d, fd = m, fm
        else:
            c, fc = m, fm
    return (c + d) / 2.
```

1.3 Démonstration

Terminaison : le variant est $\left\lceil \log_2\left(\frac{d-c}{\varepsilon}\right)\right\rceil - 1$, qui diminue de un à chaque tour de boucle. Comme c'est un entier, il devient nul ou négatif en un nombre fini de tours de boucle. Or il est négatif ou nul si et seulement si $\log_2\left(\frac{d-c}{\varepsilon}\right) \leq 2$, si et seulement si $d - c \leq 2\varepsilon$, ce qui est la condition d'arrêt de la boucle `while`.

Correction : l'invariant est : « $f(c).f(d) \leq 0$, donc f possède un point d'annulation entre c et d ». Il se démontre aisément par récurrence. Ainsi, après la boucle `while`, f possède un point d'annulation entre c et d, et la distance entre c et d est inférieure à 2ε. L'algorithme renvoie $(c + d)/2$, qui est nécessairement à une distance d'un point d'annulation inférieure à ε.

1.4 Complexité

Le nombre de tours de boucle est

$$\left\lceil \log_2 \left(\frac{b-a}{\varepsilon} \right) \right\rceil - 1.$$

Ainsi, pour avoir p bits significatifs, $\Theta(p)$ tours de boucle sont effectués.

NB : en écrivant cet algorithme, une attention a été portée aux calculs des valeurs de f : on a évité de calculer plusieurs fois la même valeur. Cela ne change rien à la complexité asymptotique, mais peut avoir une importance en pratique.

1.5 Avantages et inconvénients

Avantages : la convergence théorique est assurée et nous avons une bonne idée de l'erreur (division par deux à chaque étape). De plus, les hypothèses sur f sont faibles.

Inconvénients :

— la convergence n'est pas si rapide, comparée à d'autres méthodes (un bit significatif de plus à chaque étape).

— si l'une des deux bornes de l'intervalle de départ est très proche d'un point d'annulation, la convergence ne sera pas plus rapide pour autant.

— elle ne permet pas de trouver d'approximation pour un point d'annulation où le graphe de f est tangente à l'axe des abscisses, car f n'y change pas de signe.

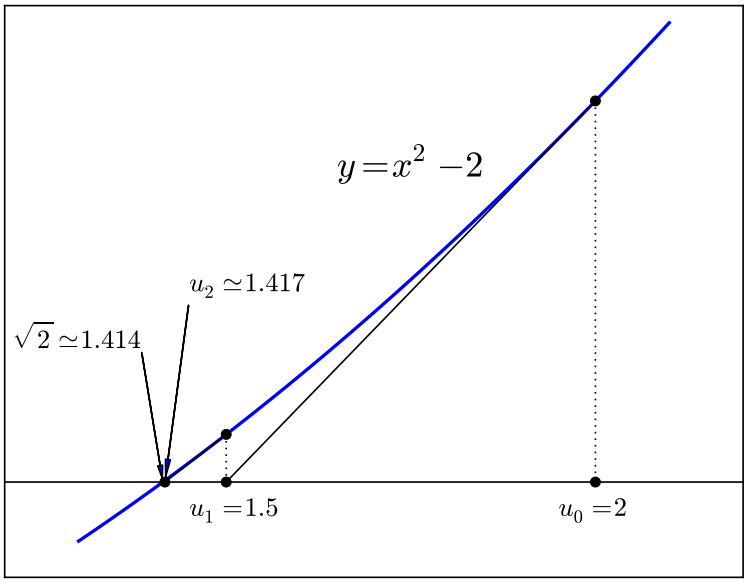
— Problèmes numériques : si les valeurs manipulées sont trop petites, on peut avoir $f(a) \times f(b)$ nul avec $f(a) \neq 0$ et $f(b) \neq 0$. Le logiciel peut aussi donner $f(b) < 0$ alors qu'en réalité $f(b) > 0$. Dans ce cas la mauvaise moitié est choisie et on peut aboutir à une approximation très éloignée de la réalité.

Cette méthode est intéressante pour avoir rapidement une première approximation, et se trouver en position d'utiliser ensuite une méthode plus performante.

2 Méthode de Newton (ou de Newton-Raphson)

2.1 Un exemple

Voyons-en graphiquement le principe sur l'exemple de $f : x \mapsto x^2 - 2$.



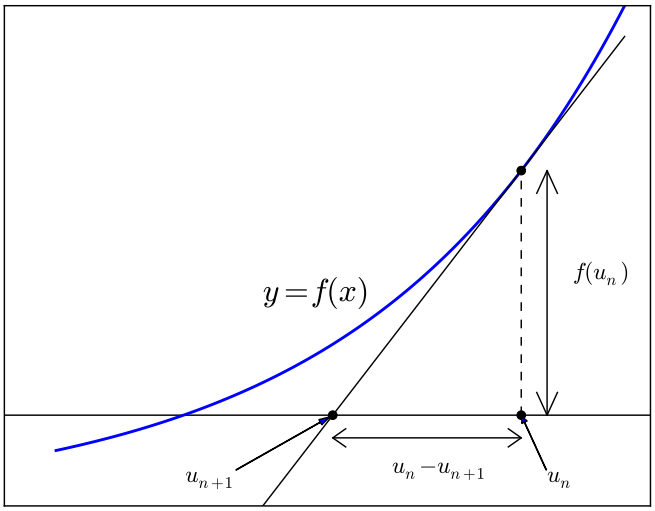
2.2 Définition

Cette méthode est *itérative* : on construit une suite $(u_n)_{n\in\mathbb{N}}$ telle que :

— u_0 est une approximation d'un zéro de f, souvent grossière.

— On pose, pour $n \in \mathbb{N}$, $u_{n+1} = F(u_n)$, où $F : x \mapsto x - \frac{f(x)}{f'(x)}$.

Ainsi, le point $(u_{n+1}, 0)$ est le point d'intersection de l'axe des abscisses et de la tangente à la courbe de f en u_n.



2.3 Implantation

```
def newton(f, fp, x0, epsilon):
    """Zéro de f par la méthode de Newton
       départ : x0, f' = fp, critère d'arrêt epsilon"""
    u = x0
    v = u - f(u)/fp(u)
    while abs(v-u) > epsilon:
        u, v = v, v - f(v)/fp(v)
    return v
```

2.4 Correction, terminaison, complexité

Se posent notamment les problèmes suivants.

1. La suite est-elle bien définie ? Plus précisément, évite-t-on les divisions par zéro ? La dérivée f' s'annule-t-elle ?
2. L'algorithme se termine-t-il ? La suite converge-t-elle ? Cela n'est pas assuré (ex. : point singulier au point d'annulation ($x \mapsto \sqrt{|x|}$), point initial éloigné du zéro, ...).
3. Le résultat est-il proche d'un zéro de f ?

2.4.1 Mathématiquement

Soit x_0 tel que $f(x_0) = 0$.
Supposons que

— f est de classe $\mathcal{C}^2$

— et $f'(x_0) \neq 0$.

Alors, pour u_0 suffisamment proche de x_0, il existe $C > 0$ tel que

$$\forall n \in \mathbb{N} \quad |u_n - x_0| \leq C 2^{-2^n}.$$

Le nombre de chiffres corrects *double* à chaque itération !

De plus, l'impact des erreurs de calcul est généralement faible (c'est souvent l'intérêt d'une méthode itérative).

2.4.2 Démonstration

Pour h au voisinage de 0 :

$$\begin{aligned} f(x_0 + h) &= f(x_0) + hf'(x_0) + f''(x_0)h^2 + o(h^2) \\ &= hf'(x_0) + O(h^2) \end{aligned}$$

et

$$\begin{aligned} f'(x_0 + h) &= f'(x_0) + hf''(x_0) + o(h) \\ &= f'(x_0) + O(h). \end{aligned}$$

Ainsi,

$$\begin{aligned} F(x_0 + h) &= x_0 + h - \frac{hf'(x_0) + O(h^2)}{f'(x_0) + O(h)} \\ &= x_0 + h - \frac{hf'(x_0)}{f'(x_0)} \times \frac{1 + O(h)}{1 + O(h)} \\ &= x_0 + O(h^2). \end{aligned}$$

Autrement dit, au voisinage de x_0,

$$F(x) = x_0 + O\left((x - x_0)^2\right).$$

Il existe donc des constantes $\alpha > 0$ et $K > 0$ vérifiant

$$\forall x \in [x_0 - \alpha, x_0 + \alpha] \quad |F(x) - x_0| \leq K |x - x_0|^2.$$

On peut supposer $K\alpha < \frac{1}{2}$ (quitte à diminuer α).

Alors :

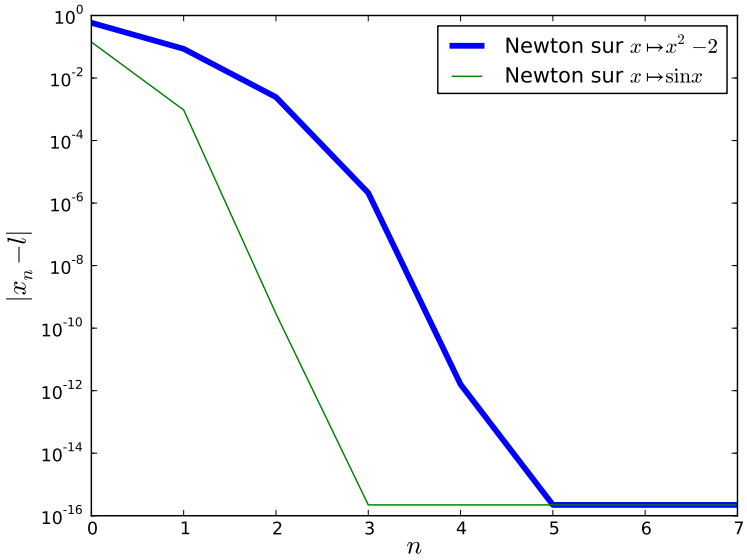
— $[x_0 - \alpha, x_0 + \alpha]$ est stable par F ;

— $\forall n \in \mathbb{N} \quad K |u_{n+1} - x_0| \leq (K |u_n - x_0|)^2$.

Par une récurrence simple, on obtient directement que

$$\forall n \in \mathbb{N} \quad K |u_n - x_0| \leq (K |u_0 - x_0|)^{2^n} \leq (K\alpha)^{2^n} \leq 2^{-2^n}.$$

2.4.3 En pratique



2.5 Applications

2.5.1 Au calcul de $1/a$ ($a > 0$)

Les calculs du rapport de deux flottants ou de l'inverse d'un flottant sont parmi les opérations les plus difficiles à calculer sur un processeur. Une méthode fréquente consiste à implanter un micro-logiciel de calcul dans le processeur.

On calcule une première approximation u_0 de l'inverse de a, puis on applique la méthode de Newton à la fonction $f : x \mapsto \frac{1}{x} - a$. On pose donc :

$$u_{n+1} = u_n - \frac{f(u_n)}{f'(u_n)} = u_n + u_n - a u_n^2 = u_n(2 - a u_n).$$

2.5.2 Au calcul de la racine carrée (version matheuse)

On applique la méthode de Newton à la fonction $f : x \mapsto x^2 - a$, où $a > 0$ est le nombre dont on veut la racine carrée :

$$u_{n+1} = u_n - \frac{u_n^2 - a}{2u_n} = \frac{1}{2}\left(u_n + \frac{a}{u_n}\right)$$

Cette méthode fonctionne correctement sur le papier mais nécessite un calcul d'inverse à chaque étape (plus deux multiplications, une addition et une division par deux).

2.5.3 Au calcul de la racine carrée (version informaticienne)

On applique la méthode de Newton à la fonction $f : x \mapsto \frac{1}{x^2} - a$, où $a > 0$ est le nombre dont on veut la racine carrée :

$$u_{n+1} = u_n - \frac{\frac{1}{u_n^2} - a}{-\frac{2}{u_n^3}} = \frac{1}{2} u_n \left(3 - a u_n^2\right).$$

On multiplie l'approximation de $\frac{1}{\sqrt{a}}$ obtenue par a pour obtenir une approximation de $\sqrt{a}$.

Dans les jeux vidéos les calculs d'inverses et de racines carrées sont très utilisés, en particulier pour les calculs d'angles d'incidence et de réflexion (calcul des lumières et des ombres en imagerie numérique). Pour cela on a besoin de normaliser des vecteurs (*i.e.* calculer $\vec{u}/\|\vec{u}\|$), on a donc besoin de calculer l'inverse d'une racine carrée. Cela a ammené la création de la méthode *Fast inverse square root* : développée vers 1990 chez Silicon Graphics (probablement), elle a par exemple été utilisée dans Quake III Arena (1999), et elle utilise la méthode de Newton.

2.6 Calcul approché de la dérivée

Remarque : pour la méthode de Newton, on a besoin de connaître f'.

Un moyen de la calculer numériquement est d'utiliser l'approximation suivante :

$$\frac{f(x_0 + h) - f(x_0)}{h} \approx f'(x_0) \quad \text{(pour } h \text{ petit)}.$$

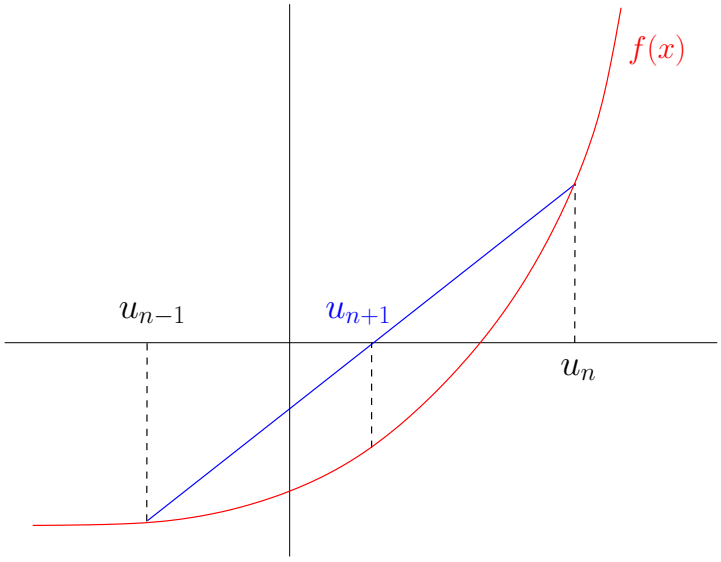
L'erreur commise est $\frac{h}{2} f''(x_0) + o(h)$ (si f est de classe $\mathcal{C}^2$).

En voici une approximation plus précise (et pas plus dure à calculer) :

$$\frac{f(x_0 + h) - f(x_0 - h)}{2h} = f'(x_0) + \frac{h^2}{6} f^{(3)}(x_0) + o(h^2) \quad \text{(si } f \text{ est de classe } \mathcal{C}^3).$$

Une autre méthode est celle de la sécante : on choisit deux approximations initiales u_0 et u_1 et on approche $f'(u_n)$ par $\frac{f(u_n) - f(u_{n-1})}{u_n - u_{n-1}}$:

$$\forall n \in \mathbb{N}^* \quad u_{n+1} = u_n - f(u_n) \frac{u_n - u_{n-1}}{f(u_n) - f(u_{n-1})}.$$



On peut montrer que si u_0 et u_1 sont suffisamment proches du zéro cherché et si f est de classe $\mathcal{C}^2$, il existe $C > 0$ tel que

$$\forall n \in \mathbb{N} \quad |u_n - x_0| \leq C 2^{-(\frac{1+\sqrt{5}}{2})^n}.$$

3 Méthodes proposées par numpy et scipy

Intérêt de l'utilisation de numpy et scipy : ne pas réinventer l'eau chaude.

3.1 Racines d'un polynôme

Pour calculer les racines de

$$P = \sum_{k=0}^{n} a_k X^k$$

on utilise

```
numpy.roots([a0, a1, ..., an])
```

Exemple :

```
>>> import numpy
>>> numpy.roots([2, 0, 1])
array([-0.+0.70710678j,  0.-0.70710678j])
```

Quelle est la méthode employée ?
La plupart des utilisateurs utilisent `numpy.roots` sans se poser de questions.
Mais il y a des cas où nous avons besoin d'avoir une idée de l'erreur :

1. On peut essayer de tester expérimentalement (mais comment être exhaustif ?) ;

2. On peut enquêter pour trouver la réponse.

Enquêtons...
La documentation (`help(numpy.roots)`) nous dit notamment :

> the algorithm relies on computing the eigenvalues of the companion matrix [1].
>
> [1] r. a. horn & c. r. johnson, *matrix analysis*. cambridge, uk : cambridge university press, 1999, pp. 146-7.

Pour continuer : consulter la bibliothèque universitaire la plus proche.

Mais par quelle méthode calcule t-on ces valeurs propres ? Le code nous renseigne :

```
a = diag(nx.ones((n-2,), p.dtype), -1)
a[0, :] = -p[1:] / p[0]
roots = eigvals(a)
```

On construit une matrice a nulle à l'exception de la sous-diagonale (où on met des 1), puis on remplace la première ligne par les coefficients $-\frac{a_1}{a_0}, \ldots, -\frac{a_n}{a_0}$ et on calcule les valeurs propres (*eigenvalues*) de la matrice obtenue.

Vous en comprendrez la raison après avoir étudié un peu plus d'algèbre linéaire (voir les *matrices compagnon*).

Allons donc voir `help(numpy.linalg.eigvals)` :

> This is a simple interface to the LAPACK routines dgeev and zgeev that sets those routines' flags to return only the eigenvalues of general real and complex arrays, respectively.

Qu'est-ce que LAPACK ? D'après Wikipédia, c'est l'acronyme de *Linear Algebra Package* :

> LAPACK (pour Linear Algebra PACKage) est une bibliothèque logicielle écrite en Fortran, dédiée comme son nom l'indique à l'algèbre linéaire numérique.

Mais quelle est la *méthode* employée ?

En utilisant un moteur de recherche («LAPACK dgeev»), on trouve un fichier `dgeev.f` (fichier FORTRAN) dont la doc évoque une décomposition QR... sans vraiment faire référence à un algorithme précis.

Conclusion : il s'agit plus d'une quête que d'une enquête.

— Les algorithmes utilisés sont souvent subtils et font appels à des mathématiques relativement évoluées/

— Il est difficile de savoir exactement ce qu'il se passe sous le capot.

— On peut quand même aller lire le code de `dgeev`.

— Heureusement, il est public...

— Wikipédia est en général un bon point d'entrée pour débuter la quête (ici *QR algorithm* sur `http://en.wikipedia.org`).

3.2 Méthode de Newton/de la sécante

Recherche d'un zéro d'une fonction :
`scipy.optimize.newton(func, x0, fprime)`
La documentation est claire :

> 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.

3.3 Méthode de Brent

`scipy.optimize.brentq(f, a, b)`.
La qualité de la documentation est remarquable.
Ça commence par dire *ce que* ça fait :

> Find a root of a function in given interval.
>
> Return float, a zero of 'f' between 'a' and 'b'. 'f' must be a continuous function, and [a,b] must be a sign changing interval.

Puis ça continue par *comment* ça le fait :

> Description : Uses the classic Brent (1973) method to find a zero of the function 'f' on the sign changing interval [a , b]. Generally considered the best of the rootfinding routines here. It is a safe version of the secant method that uses inverse quadratic extrapolation. Brent's method combines root bracketing, interval bisection, and inverse quadratic interpolation. It is sometimes known as the van Wijngaarden-Deker-Brent method. Brent (1973) claims convergence is guaranteed for functions computable within [a,b].

Puis *comment trouver plus d'information* (la référence historique et des références plus faciles d'accès) :

> [Brent1973] provides the classic description of the algorithm. Another description can be found in a recent edition of Numerical Recipes, including [PressEtal1992]. Another description is at http ://mathworld.wolfram.com/BrentsMethod.html. It should be easy to understand the algorithm just by reading our code. Our code diverges a bit from standard presentations : we choose a different formula for the extrapolation step.
>
> [Brent1973] Brent, R. P., *Algorithms for Minimization Without Derivatives*. Englewood Cliffs, NJ : Prentice-Hall, 1973. Ch. 3-4.
>
> [PressEtal1992] Press, W. H. ; Flannery, B. P. ; Teukolsky, S. A. ; and Vetterling, W. T. *Numerical Recipes in FORTRAN : The Art of Scientific Computing*, 2nd ed. Cambridge, England : Cambridge University Press, pp. 352-355, 1992. Section 9.3 : "Van Wijngaarden-Dekker-Brent Method."

4 Choisir une méthode

Le choix de la méthode à utiliser dépend de différents facteurs.

— Peut-on utiliser un programme existant ou doit-on programmer une méthode soi-même ?

— La méthode doit-elle s'appliquer à une large classe de fonctions (par exemple pas nécessairement dérivables) ou à quelques fonctions régulières et bien identifiées ?

— Dispose t-on d'une expression de la dérivée ?

— Cherche t-on une solution dans un intervalle bien précis ou n'importe quelle solution ?

— A t-on besoin d'une convergence rapide ?

Suivant les cas, on choisira la méthode :

— de dichotomie ;

— des sécantes ;

— de Brent ;

— ou de Newton.

Vous devez être capables de mettre en œuvre *vous-même* la méthode de la dichotomie ainsi que celle de Newton. Commencez par vous exercer sur les exemples vus en cours !

5 Exercices

Exercice 5.0.1. On rappelle que $\cos\left(\frac{\pi}{10}\right)$ est une des solutions de l'équation $16x^4 - 20x^2 + 5 = 0$. En déterminer une valeur approchée à 10^{-10} près.

Exercice 5.0.2. Déterminer une valeur approchée du nombre d'or à 10^{-10} près.

Exercice 5.0.3. Écrire une fonction déterminant, pour tout $p \in \mathbb{N}^*$, une valeur approchée de l'équation $1 = x + x^2 + \cdots + x^p$.