La simulation numérique
Méthodes de recherche des zéros d'une fonction

Anis SAIED
Institut Préparatoire aux Etudes d'Ingénieurs de Nabeul (IPEIN),
Université de Carthage,
Tunisie
anis_saied@hotmail.com

Mise à jour la plus récente :15 août 2016

Table des matières

Introduction

Dans ce TP, nous allons voir comment évaluer numériquement une solution d'équation de la forme $f(x) = 0$

1 Recherche d'un zéro de fonction par dichotomie

1.1 Principe

Si $f(a)$ et $f(b)$ sont de signe contraire, f étant continue, un zéro de f se trouve entre a et b par le théorème des valeurs intermédiaires. On peut le localiser dans l'une des moitiés du segment $[a,b]$ en étudiant le signe de f au milieu de $[a,b]$.

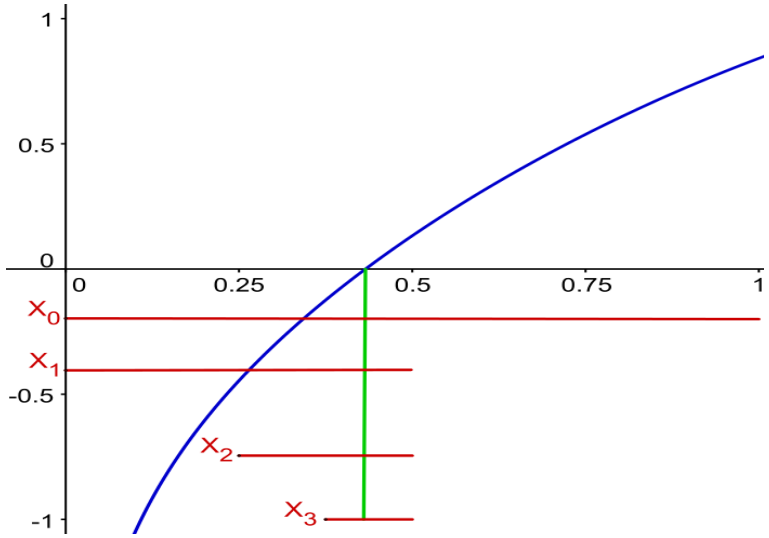

FIGURE 1 – Recherche du zéro d'une fonction par dichotomie

1.2 Programme python

```
def dicho1(f, a, b, epsilon = 1e-6):
    """ donne une valeur approchée d'un zéro de f à epsilon près par défaut
        On suppose a < b et f(a)*f(b)<= 0 """
    assert f(a) * f(b) <=0, "Il n'y a peut-être pas de zéro entre {} et {}".format(a,b)
    g = a
    d = b
    while d-g > epsilon:
        # Invariant : un zéro est entre g et d, et d-g > epsilon
        m = (g + d)/2
        if f(g)*f(m) <= 0: # attention au cas f(g) == 0
        d = m # zéro entre g et m
    else:
        g = m # zéro entre m et d
    return g
```

1.3 Terminaison de l'algorithme

On veut ici justifier que l'algorithme se terminera toujours. Cela revient à justifier que la condition `d - g > epsilon` du `while` ne restera pas toujours vraie (ce qui mènerait à une boucle infinie).

- La longueur $d-g$ est initialisée à $b-a$.
- A chaque tour de boucle la longueur $d-g>0$ est divisée par 2. Au bout de n tours de boucles on a donc $0 \leq d-g = \frac{b-a}{2^n}$

- On en déduit qu'il existera un entier n tel que $d-g \leq epsilon$ et donc que la boucle while se terminera.

1.4 Correction de l'algorithme : utilisation d'un invariant de boucle

On veut ici justifier que l'algorithme fournit bien le résultat que l'on souhaite.

- On a l'invariant de boucle *«f (g) et f (d) sont de signes contraires »*
  En effet cet invariant est supposé vrai à l'initialisation et est conservé à chaque tour de boucle.
- On en déduit ainsi mathématiquement (à l'aide du théorème des valeurs intermédiaires) l'autre invariant de boucle *« un zéro de f est compris entre g et d»*
- Par suite, en sortie de boucle, la condition $d-g \leq epsilon$, garantit que g et d encadrent un zéro de f à epsilon près ce qui est ce qu'on voulait.

1.5 Complexité de l'algorithme

- L'étude de la terminaison a permis de voir, qu'au bout de n tours de boucles, on avait l'égalité

$$d-g = \frac{b-a}{2^n}$$

- Le nombre n de tours de boucles effectués avant sortie de boucle vérifie donc :

$$\frac{b-a}{2^n} \leq \epsilon < \frac{b-a}{2^{n-1}}$$

  Autour de $n-1$ on $d-g > \epsilon$

  Ce qui donne $2n-1 < \frac{b-a}{\epsilon} \leq 2^n$ puis $n-1 < \frac{ln(\frac{b-a}{\epsilon})}{ln(2)} \leq n$

  Ainsi $n = O\left(ln\left(\frac{b-a}{\epsilon}\right)\right)$
- Que l'on comptabilise les comparaisons, les appels à f ou les affectations, la complexité c vérifie

$$\boxed{c(a,b,\epsilon) = O\left(ln\left(\frac{b-a}{\epsilon}\right)\right)}$$

⊢ Remarque : Supposons donnée une fonction vérifiant les hypothèses requises sur un intervalle $[a,b]$. Pour $\epsilon = 10^{-p}$, le nombre d'opérations sera de l'ordre de

$$ln(b-a) - ln(\epsilon) = \underbrace{ln(b-a)}_{c^{te}} + p\underbrace{ln(10)}_{c^{te}}$$

Autrement dit, pour obtenir une précision de l'ordre de 10^{-p}, le nombre d'opérations sera de l'ordre de p, pour p *«grand»*.

2 Méthode de Newton

Lorsqu'on suppose que f s'annule une unique fois et que sa dérivée ne s'annule pas, on peut s'approcher de son zéro en « prenant la tangente ». La méthode de Newton aussi appelé méthode de la tangente.
Cela revient à définir la suite donnée par la relation de récurrence :

$$U_{n+1} = U_n - \frac{f(U_n)}{f'(U_n)}$$

qui converge, sous certaines hypothèses, vers le zéro de f.

2.1 Programme python

```
def newton(f, fprime, u0, eps = 1e-6):
    """ Fonction qui retourne le zéro approchée de f
    u0 valeur initiale de x0
    k le nombre d'itérations maximal"""
    u = u0
```

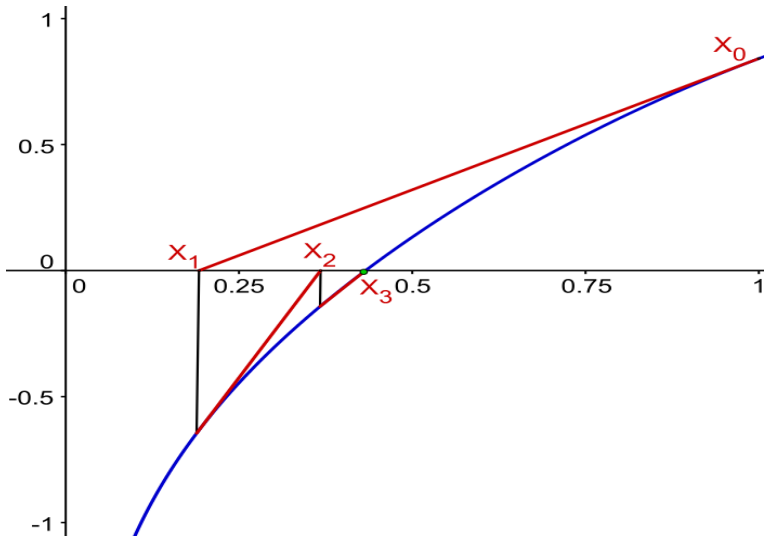

FIGURE 2 – Recherche du zéro d'une fonction par méthode de Newton

```
    v = u - f(u)/fprime(u)
    k = 0
    while abs(v - u) > eps and k < 50:
        u = v # u = u(k)
        v = v - f(v)/fprime(v) # v = u(k+1)
        k += 1
    return v
```

2.2 Exemple pratique

Calcul approché de $\sqrt{2}$. Pour obtenir une valeur approchée de 2, il suffit d'appliquer la méthode de Newton à la fonction f : $x \mapsto x^2 - 2$, en partant d'une valeur positive de x_0,par exemple $x_0 = 1$.

3 Déjà disponible en Python

Comme pour tous les algorithmes de ce chapitre, nous ne ferons que réimplémenter des fonctions déjà existantes en Python. Sans faire une description de ce qui existe déjà, je vous donnerai à chaque fois les modules Python contenant les fonctions utiles, et libre à vous d'aller en regarder les fonctionnalités précises de plus près, puisque tous les modules sont documentés en ligne. Bien sûr, cette documentation est en anglais, et pas toujours très claire, mais elle indique pour chaque fonction les arguments et options disponibles, et il faut que vous vous entrainiez à utiliser cette aide.
Concernant les méthodes de résolution approchée d'équations, tout se trouve dans le module `scipy.optimize`, qui est lui-même un sous-module du (gros) module d'analyse numérique `scipy`. Il contient entre autres les fonctions suivantes :

Fonctions utiles du module **`scipy.optimize`** :

- **`bisect(f,a,b)`** : détermine une racine de f dans $[a,b]$ en effectuant une dichotomie.
- **`newton(f,x0)`** : détermine une racine par la méthode de Newton ou approchée en partant de x_0