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.saieed@gmail.com

Mise à jour la plus récente :15 mars 2017

http://cahier-de-prepa.fr/info-ipein

Table des matières

Introduction

De nombreux problèmes en chimie, mathématiques, physique et sciences de l'ingénieur nécessitent la résolution d'équations de type $h(x) = g(x)$ où h et g sont des fonctions et où x est l'inconnue. Lrosque la résolution formelle de l'équation n'est pas simple, ou même impossible, on peut utiliser un résolution numérique.
Pour cela on se ramène à une équation avec second membre nul : $g(x) - h(x) = 0$. On cherche donc à déterminer les zéros de la fonction f telle que $f(x) = g(x) - h(x)$.

Dans ce TP, nous allons voir comment évaluer numériquement une solution d'équation de la forme $f(x) = 0$. Deux méthodes vont être présentées, mais il en existe d'autres :
- La méthode de dichotomie
- La méthode de Newton

1 Recherche d'un zéro de fonction par dichotomie

1.1 Principe

Le cours de math nous apprend que si f est une fonction à valeurs réelles, continue sur un intervalle $[a,b] \subset \mathbb{R}$ telle que $f(a)$ et $f(b)$ sont de signes opposés, $f(a) \times f(b) < 0$, le théorème des valeurs intermédiaires nous assure que l'équation f admet une solution dans $[a,b]$. La démenstration de ce théorème nous donne un algorithme effectif pour déterminer des solutions approchées de telles équations.

1.2 Rappel : Théorème de la valeur intermédiaire

Soient f une fonction continue sur un intervalle I de $\mathbb{R}$, $(a,b) \in I^2$ telque $a < b$ et $f(a) \times f(b) < 0$ et ϕ l'application de $[a,b]^2$ dans lui-même définie par :

$$\phi(a,b) = \begin{cases} (a, \frac{a+b}{2}) & \text{si } f(a)f(\frac{a+b}{2}) < 0 \\ (\frac{a+b}{2}, b) & \text{sinon} \end{cases}$$

Alors,

- Les deux suites $(a_n)_{n\in\mathbf{N}}$ et $(b_n)_{n\in\mathbf{N}}$ telles que

$$\begin{cases} (a_0, b_0) & = \text{(a,b)} \\ (a_{n+1}, b_{n+1}) & = \phi(a_n, b_n) \end{cases}$$

  sont correctement définies et l'on a pour tout $n \in \mathbb{N}$, $b_n - a_n = \frac{b-a}{2^n}$,
  avec $a_0 \leq a_n \leq a_{n+1} \leq b_{n+1} \leq b_n \leq b_0$;
- il existe $c \in I$, tel que $f(c) = 0$ et, pour tout $n \in \mathbb{N}, a_n \leq c \leq b_n$.

1.3 Présentation de la méthode par un exemple : valeur approchée de $\sqrt{2}$

Déterminer une valeur approchée de $\sqrt{2}$ à une précision $\epsilon = 10^{-7}$ (la précision est un nombre réel arbitrairement petit).
Rappel : Pour calculer la valeur approchée de $\sqrt{a}$ on doit résoudre l'équation $f(x) = x^2 - a = 0$, avec a réel > 0 .

1.3.1 Méthode de travail

1. On saisit l'expression de la fonction $f(x) = x^2 - a$ ainsi que les bornes a et b de l'intervalle d'étude et la précision p désirée.
2. On calcule la valeur $m = \frac{a+b}{2}$ puis on coupe l'intervalle en deux en fonction du signe du produit $f(a) \times f(b)$:
   - si $f(a) \times f(b) < 0$ alors on considère l'intervalle $[a; m]$.
   - si $f(a) \times f(b) > 0$ alors on considère l'intervalle $[m; b]$.
3. Si l'amplitude de l'intervalle est inférieure à p on arrête le processus sinon on calcule une nouvelle valeur de m.

Remarque : A la fin du processus, les bornes de l'intervalle d'amplitude inférieure à p se trouvent dans le variables a et b ;

1.3.2 Algorithme

```
#Première implémentation de la dichotomie
def dicho (f,a,b,epsilon):
        u,v = min(a,b),max(a,b)
        while (v-u) > epsilon :
                m = (u + v)/2
                if  f(u) * f(m) < 0 :
                        v = m
                else:
                        u = m
        return u,v
```

1.3.3 Mise en Oeuvre

Tracer sur le graphe, pour chaque itération, les bornes des intervalles $[a_n, b_n]$.

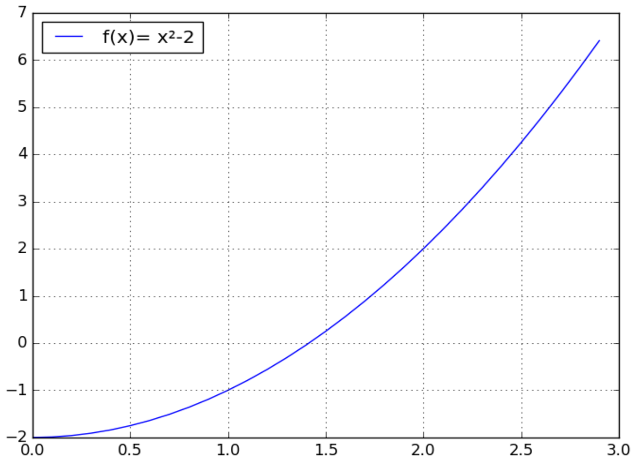

FIGURE 1 – Recherche du zéro d'une fonction $f(x) = x^2 - a = 0$ par dichotomie

Pour $[a, b] = [1, 2]$ on aura :

- $f(1) = -1 < 0 < 2 = f(2)$, donc l'équation $f(x) = 0$ possède un solution dans $[1, 2]$.
  La valeur médianne de cet intervalle est 1.5.
- $f(1) = -1 < 0 < f(1.5) = 0.25$, donc l'équation $f(x) = 0$ possède un solution dans $[1, 1.5]$.
  La valeur médianne de cet intervalle est 1.25.
- $f(1.25) = -0.4675 < 0 < f(1.5)$, donc l'équation $f(x) = 0$ possède un solution dans $[1.25, 1.5]$.
- ...
- $f(1.4140625) < 0 < f(1.41430664062) \simeq 0.000263273715973$, donc l'équation $f(x) = 0$ possède un solution dans $[1.4140625, 1.41430664062]$.

Exercice 1 :

Soit $p \in \mathbb{N}$. Peut on evaluer le nombre n d'itérations qui permettront d'obtenir $b_n - a_n \leq 10^{-p}$?

Exercice 2 :

On choisit la fonction $f : x \longmapsto x^2 + 6x + 1$.

1. Calculer ses racines formellement
2. Déterminer des valeurs approchées à 10^{-12} près de chacune de ses racines avec votre programme $dico(f, a, b, epsilon)$. Avant de lancer le programme déterminer le nombre d'itérations prévisible. Ajouter un compteur pour vérifier que tout conforme aux prévisions.

Exercice 3 :

On choisit la fonction $g : x \longmapsto 10^{-8}x^2 - \frac{4}{5}x + 10^{-8}$

a. Préciser les racines formellement et en donner ensuite des valeurs aprochées avec toute la précision possible.

b. Estimer en fonction de a et b, le nombre d'itérations provoquées par un appel `dicho (g,a , b, 10**(-12))`.

c. Lancer votre programme pour rechercher la racine comprise entre 0 et 1 avec une précision de 10^{-12} ;

d. Lancer votre programme pour rechercher la deuxième racine avec une précision de 10^{-12} ;

e. SI l'essai qui précède présente un anomalie, on ajoutera à $dico$ l'affichage de u, $f(u)$, v, $f(v)$, m et $f(m)$ à chaque itération. Relancer le programme et expliquer le problème.

On se propose de réécrire l'algorithme de dichotomie en implémentant une fonction qui calcule n, le nombre théorique ditérations (c'est-à-dire l'indice à partir duquel $b_n - a_n \leq \epsilon$) et gère une sortie de boucle si le nombre des itérations dépasse n d'une certaine valeur.

```python
import numpy as np
#pour la fonction log (népérien)
import scipy as sp #ou bien import math
def dicho (f,a,b,epsilon):
    """ donne une valeur approchée d'un zéro de f à epsilon près
    On suppose que 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)
    assert epsilon > 0
    u,v,c =min(a,b),max(a,b),0
    n=sp.floor((sp.log(v-u)-sp.log(epsilon))/sp.log(2))
    while (v-u) > epsilon and c<=n:
        c = c + 1
        m = (u + v)/2
        if  f(u) * f(m) < 0 :
            v = m
        else:
            u = m
    return u,v,c #c est le nombre d'itérations effectives.
```

Exercice 4 :

On considère l'équation (E) suivante : $x - esin(x) = 0$ Il s'agit de déterminer la solution $\alpha \in [1;10]$ avec une précision $p = 10^{-4}$.

Travail démandé :

a. Ecrire le programme Python permettant de résoudre numériquement l'équation (E).
b. Saisir l'expression de la fonction $f(x) = x - esin(x)$ ainsi que les bornes a et b de l'intervalle d'étude et la précision p désirée.
c. Déterminer la solution $\alpha \in [1;10]$.

2 Méthode de Newton

On suppose, comme précédemment, que l'on dispose d'un intervalle $[a,b]$ sur lequel la fonction f est continue, strictement monotonne et vérifie $f(a)f(b) < 0$. On sait alos que f admet un unique zéro r dans $]a,b[$. Ici, on suppose également que f est dérivable et que la dérivée de f 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**.

2.1 Principe de la méthode de la tangente

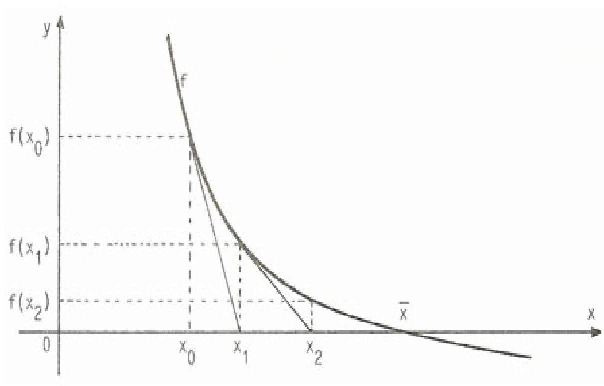

FIGURE 2 – Recherche du zéro d'une fonction par méthode de Newton

On commence par présenter le principe de la méthode sur la fonction f qui est présentée ci-dessus. Notons r le zéro de f dans l'intervalle considéré, x_0 un élément de cet intervalle.

On trace la tangente à la courbe représentative de f au point de la courbe d'abscisse x_0. Cette droite a une pente égale à $f'(x_0)$ et coupe l'axe des abscisses au point d'abscisse x_1. On remarque sur cet exemple que x_1 est plus proche que x_0 : x_1 est une meilleure approximation de r que x_0.

Exprimons x_1 en fonction de x_0, $f(x_0)$ et $f'(x_0)$:

- On sait que la tangente à la courbe de f au point d'abscisse x_0 a pour équation :

$$f(x) = f'(x_0)(x - x_0) + f(x_0)$$

- Par définition de x_1 :
$$0 = f'(x_0)(x_1 - x_0) + f(x_0)$$
$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$
- On réitère l'opération précédente à partir de x_1, on obtient une seconde tangente qui coupe l'axe des abscisses au point d'abscisse x_2, qui est encore une meilleure approximation du zéro r.
De même à la n^e itération, on obtient
$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

On continue ce procédé jusqu'à ce que la différence entre deux approximations successives, x_n et x_{n+1}, soit très petite (inférieure à une précision e, que l'on choisit au départ). Le nombre x_{n+1} est alors une valeur approchée de r.

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.

2.2 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 $x_0 = 1$.

```python
#Définition de la fonction f
def f(x):
        return x**2 - 2

#Définition de la dérivée de la fonction f
def fd(x):
        return 2*x

def newton(f, fprime, u0, eps = 1e-6):
    u = u0
    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,k #k est le nombre d'itérations effectives.

#Initialisation des variables
e = 10**(-5) #précision
x0 = 1
r,n = newton(f, fd, x0, e)
print("Au bout de ",n," itérations, la valeur approchée obtenue est ",r)
```

L'exécution du script ci-dessus donne :
`Au bout de 3 itérations, la valeur approchée obtenue est 1.4142135623746899`
Si on change $x_0 = 6$ on trouve
`Au bout de 5 itérations, la valeur approchée obtenue est 1.4142135623732204`

On remarque que plus x_0 est plus proche de la vraie valeur de la solution, plus le nombre d'itérations est petit, et donc plus le programme est rapide.

Exercice 5 :

Il s'agit de développer un programme `Python` de résolution numérique d'une équation par la méthode de Newton :
On considère l'équation (E) suivante : $x - esin(x) = 0$
Il s'agit de déterminer la solution $\alpha \in [1.5;5]$ avec une précision $p = 10^{-4}$.

Travail démandé :

a. Ecrire le programme Python permettant de résoudre numériquement l'équation (E).
b. Saisir l'expression de la fonction $f(x) = x - esin(x)$ ainsi que les bornes a et b de l'intervalle d'étude et la précision p désirée.
c. Déterminer la solution $\alpha \in [1.5;5]$.

3 Déjà disponible en Python

Comme pour tous les algorithmes de ce chapitre, nous ne ferons que réimplémenter des fonctions déjà existantes en Python. 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 en partant de x_0