Informatique en CPGE (2018-2019)
Résolution d'une équation : méthodes de dichotomie et de Newton

1 Recherche dichotomique

L'algorithme de recherche dichotomique ("bisection search" en anglais) consiste à partir de deux valeurs a et b encadrant une solution unique d'une équation $f(x) = 0$, à tester si la solution est plus grande ou plus petite que $m = (a + b)/2$. Suivant le résultat, on restreint la recherche à l'intervalle $[a; m]$ ou à l'intervalle $[m; b]$. On reproduit ce schéma tant que l'amplitude de l'intervalle (qui est divisée par deux à chaque étape) est supérieure à une précision epsilon donnée.

Algorithme : les variables sont a et b, les bornes de l'intervalle, f la fonction (qui change de signe entre a et b), epsilon la précision, m la valeur courante du milieu.

```
Tant que b - a > epsilon
    m prend la valeur (a+b)/2
    Si f(m) et f(a) sont de même signe alors
        a prend la valeur m
    sinon
        b prend la valeur m
```

Un programme en Python :

```
def zero_dic(f,a,b,eps):
    while b-a>eps:
        m=(a+b)/2
        if f(a)*f(m)>0:
            a=m
        else:
            b=m
    return (a+b)/2
```

L'amplitude de l'intervalle étant divisée par deux à chaque étape, on gagne un bit de précision à chaque passage dans la boucle while. L'intérêt de cette méthode est que les conditions sur la fonction f ne sont pas trop exigeantes : être continue et changer de signe.

Analyse de l'algorithme :
Il est nécessaire de démontrer la validité de cet algorithme puis d'étudier sa complexité.

Terminaison : il suffit de remarquer qu'après k étapes, $b - a$ a été divisé par 2^k et comme $\dfrac{b-a}{2^k}$ a pour limite 0 quand k tend vers l'infini, pour tout $\epsilon > 0$, il existe une valeur de k à partir de laquelle toutes les amplitudes des intervalles seront inférieures à ϵ.

Correction : on utilise l'invariant $f(a)f(b) \leq 0$. Cet invariant est bien vérifié avant l'entrée dans la boucle par hypothèse. Ensuite on suppose que cet invariant est vérifié avant un passage dans la boucle : si $f(a)$ et $f(m)$ sont de même signe, alors a prend la valeur de m et donc garde un signe contraire à celui de

b ; si $f(a)$ et $f(m)$ sont de signe contraire, alors b prend la valeur de m et donc garde un signe contraire à celui de a.

Ainsi les valeurs de a et b en sortie sont les bornes d'un intervalle d'amplitude maximale ϵ telles que $f(a)f(b) \leq 0$. D'après le théorème des valeurs intermédiaires, la solution appartient à cet intervalle.

Complexité : si on ne tient pas compte de la complexité des calculs de $f(m)$ lors des appels à la fonction, on remarque que la boucle est exécutée k fois si et seulement si $\dfrac{b-a}{2^k} \leq \epsilon < \dfrac{b-a}{2^{k-1}}$, soit $\dfrac{b-a}{\epsilon} \leq 2^k < 2\dfrac{b-a}{\epsilon}$. On obtient alors $\ln\left(\dfrac{b-a}{\epsilon}\right) \leq k\ln 2 < \ln 2 + \ln\left(\dfrac{b-a}{\epsilon}\right)$ ce qui nous donne $\log_2\left(\dfrac{b-a}{\epsilon}\right) \leq k < 1 + \log_2\left(\dfrac{b-a}{\epsilon}\right)$. Par exemple si $b-a=1$ et $\epsilon = 2^{-p}$ alors $k=p$; ce qui justifie qu'on gagne un bit de précision à chaque étape.

2 Méthode de Newton

2.1 Principe

On cherche la solution de l'équation $f(x) = 0$, c'est-à-dire l'abscisse du point d'intersection de la courbe $\mathcal{C}$ représentant f avec l'axe des abscisses. Sous certaines conditions sur f, on part d'une valeur x_0 et on détermine l'abscisse x_1 du point d'intersection de la tangente T_1 à la courbe $\mathcal{C}$ au point d'abscisse x_0 avec l'axe des abscisses ; x_1 est solution de l'équation : $f'(x_0)(x-x_0)+f(x_0)=0$. Donc $x_1 = x_0 - \dfrac{f(x_0)}{f'(x_0)}$ et x_1 est une valeur approchée de x. On recommence un certain nombre de fois avec x_n et la tangente T_n au point d'abscisse x_{n-1}. Soit $x_n = x_{n-1} - \dfrac{f(x_{n-1})}{f'(x_{n-1})}$; la suite (x_n) converge vers la solution x.

2.2 Exemples

2.2.1 Calcul de l'inverse

On détermine la solution $x = \dfrac{1}{a}$ de l'équation $\dfrac{1}{x} - a = 0$. On a $f'(x) = -\dfrac{1}{x^2}$ et :

$$x_n = x_{n-1} - \frac{f(x_{n-1})}{f'(x_{n-1})} \iff x_n = x_{n-1} - \frac{\dfrac{1}{x_{n-1}} - a}{\dfrac{-1}{x_{n-1}^2}}$$

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

$$\iff x_n = x_{n-1} + x_{n-1} - a x_{n-1}^2$$

$$\iff x_n = x_{n-1}(2 - a x_{n-1})$$

2.2.2 Calcul de la racine carrée

On détermine la solution de l'équation $x^2 - a = 0$. On a $f(x) = x^2 - a$ et $f'(x) = 2x$.

$$x_n = x_{n-1} - \frac{f(x_{n-1})}{f'(x_{n-1})} \iff x_n = x_{n-1} - \frac{x_{n-1}^2 - a}{2x_{n-1}}$$

$$\iff x_n = \frac{x_{n-1}^2 + a}{2x_{n-1}}$$

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

Programme :

```
def racine(a,x,eps):
    while abs(x*x-a)>eps:
        x=0.5*(x+a/x)
    return x
```

On peut compléter le code précédent afin de compter le nombre d'itérations et comparer l'efficacité de cet algorithme avec celle de la recherche dichotomique. Avec la recherche dichotomique, pour une précision de 10^{-4}, si l'intervalle de départ a une amplitude de 1, il est nécessaire de le diviser en deux n fois avec $2^n \geq 10^4$, soit $n \geq 4\ln 10/\ln 2$ ce qui nous donne $n = 14$. Avec la méthode de Newton, trois itérations sont suffisantes. Le nombre de décimales correctes est multiplié par deux à chaque étape.

2.3 Cas général

Afin de calculer les termes de la suite (x_n) définis par $x_n = x_{n-1} - \dfrac{f(x_{n-1})}{f'(x_{n-1})}$ il est nécessaire de définir dans le programme la fonction f et la fonction f', que l'on notera df. La variable cpt est un compteur permettant d'afficher le nombre d'itérations nécessaires pour obtenir la précision souhaitée. Mais nous ne connaissons pas à l'avance le nombre d'itérations et il y a des cas où la suite diverge, donc il est important de limiter ce nombre ; c'est le rôle de l'argument N dans le programme qui suit.

```
def newton(f,x,df,eps,N=100):
    cpt=0
    while abs(f(x))>eps and cpt<=N:
        x=x-f(x)/df(x)
        cpt+=1
    return x,cpt
```

On peut améliorer ce code de plusieurs manières.

— Dans la boucle, on évalue deux fois la quantité $f(x)$. Sur de petits exemples cela n'a pas une grande importance, mais dans le cas de fonctions beaucoup plus compliquées, faire deux fois le même travail peut ne pas être négligeable. Nous pouvons donc stocker la valeur $f(x)$ dans une variable locale.

— Un problème sérieux est le risque de diviser par zéro ou par un nombre très petit qui pourrait créer une très grande valeur pour x et faire diverger la méthode. C'est pourquoi nous devons tester les valeurs de $f'(x)$ et afficher un avertissement si une valeur devient très petite.

— Il est aussi intéressant de stocker dans une liste les valeurs x et $f(x)$ obtenues à chaque itération pour les imprimer ou les utiliser dans un graphique illustrant le comportement de la méthode de Newton. Pour cela nous pouvons ajouter en argument un booléen indiquant si nous stockons ou pas ces valeurs.

Voici un code optimisé :

```
def newton(f,x,df,eps,N=100,save=False):
    valeur_f=f(x)
    cpt=0
    if save: valeurs=[(x,valeur_f)]
```

```
    while abs(valeur_f)>eps and cpt<=N:
        valeur_df=df(x)
        if abs(valeur_df)<1E-14:
            print("Attention, valeur de f' trop petite")
            break
        x=x-valeur_f/valeur_df
        cpt+=1
        valeur_f=f(x)
        if save: valeurs.append((x,valeur_f))
    if save:
        return x,cpt,valeurs
    else:
        return x,cpt
```

3 Complément

3.1 Méthode de la sécante

Le calcul de $f'(x)$ peut être compliqué et si nous devons résoudre plusieurs équations, il peut être intéressant de faire effectuer ce calcul par le programme. Pour cela nous pouvons utiliser une approximation en remplaçant $f'(x)$ par $\dfrac{f(x+h)-f(x-h)}{2h}$ avec h de l'ordre de 10^{-6} par exemple.

Cette méthode est une variante de la "méthode de la sécante".

```
def Df(f,x):
    h=1e-6
    return (f(x+h)-f(x-h))/(2*h)
```

On remplace alors df par Df dans le code de la fonction `newton`.

3.2 Optimisation avec eval et exec

Plutôt que modifier la fonction f dans le code du programme, on peut faire en sorte que le programme demande à l'utilisateur d'entrer l'expression de la fonction au clavier. On importe au préalable toutes les fonctions du module math (sin, cos, exp, ...). Puis on utilise les fonctions `eval` et `exec`.

Le code suivant doit alors se trouver au début du programme.

```
from math import *
formule=input("entrer l'expression de la fonction")

code="""
def f(x):
    return eval(formule)
"""
exec(code)
```

D'une certaine manière, l'instruction `exec(code)` remplace la partie "code= """ ..."""" par les instructions se trouvant entre les guillemets. La fonction `eval` évalue le contenu de la chaîne "formule". (Par exemple `eval('2+3')` renvoie 5).

4 Utilisation de la bibliothèque scipy

Le module `optimize` de la bibliothèque scientifique `scipy` contient les fonctions `bisect` et `newton` dans lesquelles sont programmées respectivement la méthode de dichotomie et la méthode de Newton.

Les fonctions `root` et `fsolve` permettent également de trouver les valeurs approchées des zéros d'une fonction.

```
import scipy.optimize

def f(x):
    return x**2-2

a=1
b=2

x=scipy.optimize.bisect(f,a,b)
print(x)

x=scipy.optimize.newton(f,a)
print(x)

x=scipy.optimize.fsolve(f,a)
print(x)

x=scipy.optimize.root(f,a)
print(x)
```