

Institut Préparatoire aux Etudes d'Ingénieurs de Nabeul

Département : Mathématiques et Informatique

Informatique : TP Python

TP 3. Simulation Numérique

Méthodes de recherche des zéros d'une fonction

Exercice 1 : Comparaison de différentes méthodes de résolution

Soient $[a; b]$ un segment de \mathbb{R} et f une fonction dérivable sur $[a; b]$. On suppose que f' est continue, ne s'annule pas et que $f(a) \times f(b) < 0$. Il existe alors un unique réel $r \in]a; b[$ tel que $f(r) = 0$.

1. Donner une équation de la corde de la courbe de f entre les points d'abscisses a et b .
Calculer l'abscisse c du point d'intersection de cette corde avec l'axe des abscisses.

Corrigé :

La corde de la courbe f entre les points d'abscisses a et b est la droite passant par le point de coordonnées $(a, f(a))$, de coefficient directeur $\frac{f(b) - f(a)}{b - a}$. Elle admet donc pour équation :

$$y = f(a) + \frac{f(b) - f(a)}{b - a}(x - a)$$

$f(a)$ est différent de $f(b)$ d'après l'hypothèse $f(a)f(b) < 0$, donc cette droite n'est pas parallèle à l'axe des abscisses. Notons c l'abscisse de son intersection avec l'axe des abscisses.

$$0 = f(a) + \frac{f(b) - f(a)}{b - a}(c - a)$$

D'où

$$c = a - \frac{b - a}{f(b) - f(a)}f(a) = \frac{af(b) - bf(a)}{f(b) - f(a)}$$

2. Méthode de la fausse position. Le principe de cette méthode, proche de celle de dichotomie, consiste aussi à diviser à chaque étape l'intervalle de travail en deux parties. Mais au lieu de couper un segment $[u; v]$ en son milieu, on construit la corde entre les points de coordonnées $(u, f(u))$ et $(v, f(v))$ et on calcule l'abscisse c de l'intersection de cette corde avec l'axe des abscisses. Comme pour la dichotomie, on choisit ensuite de continuer dans l'intervalle $[u; c]$ ou dans l'intervalle $[c; v]$ en fonction du signe de $f(u) \times f(c)$.

- a. Illustrer la méthode et définir des relations de récurrence correspondant à ce schéma en s'inspirant du cours sur la dichotomie.

Corrigé :

schéma

On définit trois suites (a_n) , (b_n) et (c_n) de la manière suivante : $a_0 = a$, $b_0 = b$

Pour tout $n \in \mathbb{N}$,

$$c_n = \frac{a_n f(b_n) - b_n f(a_n)}{f(b_n) - f(a_n)}$$

Si $f(a_n)f(c_n) < 0$, on pose $a_{n+1} = a_n$ et $b_{n+1} = c_n$, sinon on pose $a_{n+1} = c_n$ et $b_{n+1} = b_n$

- b. Définir une fonction Python `fausse_pos` d'arguments f , a , b et e permettant de trouver une valeur approchée du zéro de f sur $[a; b]$ avec un critère d'arrêt e . Cette fonction devra également renvoyer le nombre d'itérations effectuées.

Corrigé :

```
def fausse_pos(f,a,b,e):
    """ résout f(x)=0 sur [a,b] avec un critère d'arrêt e
    par la méthode de la fausse position"""
    inf,sur = a,b
    n=0
    while sur-inf > e:
        c=(inf*f(sur)-sur*f(inf))/(f(sur)-f(inf))
        if f(inf)*f(c)<0:
            sur = c
        else:
            inf = c
        n+=1
    return c,n
```

3. Méthode de la sécante. Dans certains cas, le calcul de f' est difficile ou très long, voire impossible. Dans la méthode de la sécante, on procède comme dans la méthode de Newton, mais on approche la valeur de la dérivée en un point par la pente d'une corde, autrement dit on approche la tangente en ce point par une corde : partant de u et v , on construit la corde entre les points de coordonnées $(u, f(u))$ et $(v, f(v))$ et on calcule l'abscisse w de l'intersection de cette corde avec l'axe des abscisses, puis on recommence entre v et w et ainsi de suite.
- a. Illustrer la méthode et définir des relations de récurrence correspondant à ce schéma en s'inspirant du cours sur la méthode de Newton. On prendra $x_0 = a$ et $x_1 = b$.

Corrigé :

Schéma

On définit la suite $(x_n)_{n \in \mathbb{N}}$, par $x_0 = a$, $x_1 = b$ pour tout $n \in \mathbb{N}^*$,

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

- b. Définir une fonction Python `secante` d'arguments f , a , b et e permettant de trouver une valeur approchée du zéro de f sur $[a; b]$ avec un critère d'arrêt e . Cette fonction devra également renvoyer le nombre d'itérations effectuées.

Corrigé :

```
def secante(f,a,b,e):
    """ résout f(x)=0 sur [a,b] avec un critère d'arrêt e
    par la méthode de la sécante"""
    u,v = a,b
    n=0
    while abs(v-u)> e:
        u,v=v,(u*f(v)-v*f(u))/(f(v)-f(u))
        n+=1
    return v,n
```

4. On veut maintenant comparer entre elles les différentes méthodes.
- a. Ecrire des fonctions decho et Newton, d'arguments à préciser, permettant la résolution approchée sur $[a;b]$ de l'équation $f(x) = 0$, utilisant respectivement les méthodes de dichotomie et de Newton. Comme les précédentes, ces fonctions devront également le nombre d'itérations effectuées.

Corrigé :

C'est une question de cours. Pour la dichotomie, la fonction prend en arguments la fonction f , les bornes a et b de l'intervalle et le critère d'arrêt e .

```
def dichotomie(f, a, b, e):
    """ résout f(x)=0 sur [a,b] avec un critère d'arrêt e
    par dichotomie """
    inf, sup = a, b
    n = 0
    while sup - inf > e:
        m = (sup + inf) / 2
        if f(inf) * f(m) < 0:
            sup = m
        else:
            inf = m
        n += 1
    return m, n
```

Pour la méthode de Newton s'ajoute aux arguments précédents la fonction dérivée notée fp et le point de départ x_0 , mais a et b ne servent plus.

```
def newton(f, fp, x0, e):
    """ résout f(x)=0 sur [a,b] avec un critère d'arrêt e
    par la méthode de Newton, partant de x0 """
    xn = x0
    xn_plus1 = xn - f(xn) / fp(xn)
    n = 1
    while abs(xn_plus1 - xn) > e:
        xn, xn_plus1 = xn_plus1, xn_plus1 - f(xn_plus1) / fp(xn_plus1)
        n += 1
    return xn_plus1, n
```

- b. Se renseigner sur la fonction `fsolve` du sous-module `optimize` de `scipy`.

Corrigé :

On importe la fonction `fsolve` du sous-module `optimize` de `scipy` par la commande `from scipy.optimize import fsolve`

L'aide nous apprend que cette fonction permet des résolutions approchées d'équations : `fsolve(f, x0)` renvoie une valeur approchée de la solution de l'équation $f(x) = 0$, par une méthode itérative partant de x_0 .

- c. Ecrire une fonction `compare(f, fp, a, b, x0, e)`, de paramètres une fonction, sa dérivée, les bornes de l'intervalle, un point de départ X_0 et un critère d'arrêt e , permettant de renvoyer les valeurs approchées de la solution de l'équation $f(x) = 0$ sur $[a;b]$ obtenues par `fsolve`, `dichotomie`, `fausse_pos`, `secante` et `newton`, ainsi que le nombre d'itérations nécessaires pour ces 4 dernières méthodes. L'appliquer avec $f : x \rightarrow x^2 - 2$ sur $[1;2]$ pour différentes valeurs de e .

Corrigé :

```
def compare(f,fp,a, b, x0, e):
    """ Comparaison des méthodes de résolution approchée
    de f(x)=0 """
    print(fsolve(f,x0), dichotomie(f,a,b,e), fausse_pos(f,a,b,e), \
    secante(f,a,b,e), newton(f,fp,e,x0))

#définir la fonction et sa dérivée
def g(x):
    return x**2
def gp(x):
    return 2*x

compare(g,gp,1,2,1,0.1)
compare(g,gp,1,2,1,0.01)
compare(g,gp,1,2,1,0.001)
compare(g,gp,1,2,1,10**(-15))
```

Les résultats obtenus sont présentés dans le tableau suivant :

Méthode	fsolve	dicho	fausse_pos
e=0.1	1.41421356	(1.4375, 4)	(1.4142135623730951, 21)
e=0.01	1.41421356	(1.4140625, 7)	(1.4142135623730951, 21)
e=0.001	1.41421356	(1.4150390625, 10)	(1.4142135623730951, 21)
e=10 ⁻¹⁵	1.41421356	(1.414213562373095, 50)	(1.4142135623730951, 21)

Méthode	sécante	newton
e=0.1	(1.4, 2)	(1.4166666666666667, 2)
e=0.01	(1.41421143847487, 4)	(1.4142156862745099, 3)
e=0.001	(1.41421143847487, 4)	(1.4142135623746899, 4)
e=10 ⁻¹⁵	(1.414213562373095, 7)	(1.414213562373095, 6)

On observe qu'en nombre d'itérations, sur cet exemple, la méthode de newton est la plus rapide, suivie par la méthode de la sécante. La méthode de la fausse_pos apparaît ici plus lente que celle de la dichotomie pour les valeurs de e les plus grandes, mais plus rapide ensuite.