Institut **P**réparatoire aux **E**tudes d'**I**ngénieurs de **N**abeul
Informatique : TP Python

TP 3. Simulation Numérique
Résolution et intégration numérique des fonctions

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 pints 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 deroite passnt par le pint de coordonnées $(a, f(a))$, de coefficient directeur $\dfrac{f(b)-f(a)}{b-a}$. 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 $b_{n+1}=a_n$, 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 :

```python
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 sup-inf > e:
                c=(inf*f(sup)-sup*f(inf))/(f(sup)-f(inf))
                if f(inf)*f(c)<0:
                        sup = c
                else:
                        inf = c
                n+=1
        return c,n
```

3. `Méthode de la sécante`. Dans certains cas, le calcul de f' est difficil 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 :

```python
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 respeectivement 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 boprnes `a` et `b` de l'intervalle et le critère d'arret `e`.

```
def dicho(f,a,b,e):
        """ résout f(x)=0 sur [a,b] avec un critère d'arrêt e
        par dichotomie"""
        inf,sur = 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 `x0`, 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 , parton de x0"""
        xn = x0
        xnplus1= xn - f(xn)/fp(xn)
        n=1
        while abs(xnplus1 - xn) > e:
                xn, xnplus1= xnplus1, xnplus1- f(xnplus1)/fp(xnplus1)
                n+=1
        return xnplus1,n
```

   b. Se rensigner 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 `x0`.

   c. Ecrire une fonction `compare(f, fp, a, b, x0, e)`, de paramètres une fonction, sa dérivée, les bornes de l'intervalle, une point de départ `X0` et une 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`, `dicho`, `fausse_pos`, `secante` et `newton`, ainsi que les nombre d'itérations nécessaires pour ces 4 dernières méthodes. L'appliquer avec $f : x \longrightarrow x^2 - 2$ 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), dicho(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.

Exercice 2 : Calcul approché d'une intégrale

La méthode des rectangles (à gauche) approche cette intégrale (vue comme une aire) par l'aire de N rectangles de largeur $\frac{b-a}{N}$, via la formule

$$Rg_N(f) = \frac{b-a}{N}\sum_{k=0}^{N-1} f\left(a + k\frac{b-a}{N}\right)$$

La méthode des rectangles (à droite) approche cette intégrale (vue comme une aire) par l'aire de N rectangles de largeur $\frac{b-a}{N}$, via la formule

$$Rd_N(f) = \frac{b-a}{N}\sum_{k=1}^{N} f\left(a + k\frac{b-a}{N}\right) = \frac{b-a}{N}\sum_{k=0}^{N-1} f\left(a + (k+1)\frac{b-a}{N}\right)$$

La méthode des trapèzes approche cette intégrale (vue comme une aire) par l'aire de N rectangles de largeur $\frac{b-a}{N}$, via la formule

$$T_N(f) = \frac{b-a}{N}\left[\frac{f(a)+f(b)}{2} + \sum_{k=1}^{N-1} f\left(a+k\frac{b-a}{N}\right)\right] = \frac{1}{2}\left(Rg_N(f)+Rd_N(f)\right)$$

Enfin, la méthode de Simpson approche cette intégrale (vue comme une aire) par la somme des aires sous la courbe de N polynômes du second degré, via la formule

$$S_N(f) = \frac{b-a}{6N}\left[f(a) + 2\sum_{k=1}^{N-1} f\left(a+k\frac{b-a}{N}\right) + 4\sum_{k=1}^{N-1} f\left(a+(k+\frac{1}{2})\frac{b-a}{N}\right) + f(b)\right]$$

On se réfèrera au cours de mathématiques pour plus de détails. On considère aussi la fonction suivante :

$$f:\left\{\begin{array}{lcl}[0,1] & \longrightarrow & \mathbb{R} \\ x & \longmapsto & \begin{cases} cos(8\pi x) & si \quad x \leq 1/4; \\ 1/2 + cos(160\pi x) & si \quad 1/4 < x \leq 1/2; \\ -1/2 & si \quad 1/2 < x \leq 3/4; \\ 8(x-1)(16x-13) + cos(32\pi x) & si \quad x > 3/4 \end{cases}\end{array}\right.$$

Q1 Écrire une fonction `f(x)` renvoyant $f(x)$, pour $x \in [0,1]$.

Corrigé :

```
def f(x):
        from math import cos, pi
        if x<= 1/4:
                result = cos(8*pi*x)
        elif x<=1/2:
                result = 1/2+cos(160*pi*x)
        elif x<=3/4:
                result = -1/2
        else:
                result = 8*(x-1)*(16*x-13)+cos(32*pi*x)
        return result
```

Q2 Écrire une fonction `plot_f(nom_de_fichier)` ne renvoyant rien et enregistrant dans `nom_de_fichier` le graphe de la fonction f.

Corrigé :

```
def plot_f(nom_de_fichier):
        import os
        import matplotlib.pyplot as plt
        import numpy as np
        fv = np.vectorize(f)
        tx=np.linspace(0,1,1000)
        ty=fv(tx)
        plt.plot(tx,ty)
        plt.grid()
```

```
        dossier = os.getcwd()
        fichier = dossier +"/"+nom_de_fichier+".png"
        print(fichier)
        plt.savefig(fichier)
#Test
plot_f("ma_fonction")
```

Q3 Calculer à la main $\int_0^1 f(t)dt$.

Corrigé :

$$\int_0^1 f(x)dx = \int_0^{1/4} f(x)dx + \int_{1/4}^{1/2} f(x)dx + \int_{1/2}^{3/4} f(x)dx + \int_{3/4}^{1} f(x)dx$$

$$\int_0^{1/4} cos(8\pi x)dx = \left[\frac{sin(8\pi x)}{(8\pi)}\right]_0^{1/4} = \frac{sin(2\pi)}{8\pi} = 0$$

$$\int_{1/4}^{1/2} \frac{1}{2} + cos(160\pi x)dx = \left[\frac{x}{2} + \frac{sin(160\pi x)}{(160\pi)}\right]_{1/4}^{1/2} = 0.125$$

$$\int_{1/2}^{3/4} -\frac{1}{2}dx = \left[-\frac{x}{2}\right]_{1/2}^{3/4} = -0.125$$

$$\int_{3/4}^{1} 8(x-1)(16x-13) + cos(32\pi x)dx = -0.0833333333333357$$

$$\int_0^1 f(x)dx = -0.0833333333333357$$

Q4 Écrire une fonction $R_g(g, a, b, N)$ renvoyant $Rg_N(g)$ pour une fonction g définie sur $[a, b]$.

Corrigé :

```
def Rg (f, a, b, n):
    int_rect = 0
    h = (b-a)/n
    t = a
    for k in range(n):
        int_rect += h*f(t)
        t += h
    return int_rect

#test
I = Rg(f,0,1,1000)
#I = -0.08232800000000053
```

Q5 Écrire une fonction $R_d(g, a, b, N)$ renvoyant $Rd_N(g)$ pour une fonction g définie sur $[a, b]$.

Corrigé :

```
def Rd (f, a, b, n):
        assert n>0
        int_rect = 0
        h = (b-a)/n
```

```
        t = a+h
        for k in range(1,n+1):
                int_rect += h*f(t)
                t += h
        return int_rect

#test
I = Rd(f,0,1,1000)
#I = -0.08232800000000051
```

Q6 Écrire une fonction $T(g, a, b, N)$ renvoyant $T_N(g)$ pour une fonction g définie sur $[a, b]$.

Corrigé :

```
def T(f,a,b,n):
    h = (b-a)/n
    int_trapeze = 0.
    t = a
    for k in range(n):
        int_trapeze += h * (f(t) + f(t+h)) / 2
        t += h
    return int_trapeze

#test
I = T(f,0,1,1000)
#I = -0.08232800000000073

#Autrement
def Tgd(f,a,b,n):
        return  (1/2)*(Rg(f,a,b,n)+Rd(f,a,b,n))

#test
I = Tgd(f,0,1,1000)
#I = -0.08232800000000051
```

Q7 Écrire une fonction $S(g, a, b, N)$ renvoyant $S_N(g)$ pour une fonction g définie sur $[a, b]$.

Corrigé :

```
def S(f,a,b,n):
        h=(b-a)/n
        somme1 = sum([f(a+k*h) for k in range(1,n)])
        somme2 = sum([f(a+(k+(1/2))*h) for k in range(1,n)])
        return ((b-a)/(6*n))*(f(a)+2*somme1 + 4 * somme2 + f(b))

#test
I = S(f,0,1,1000)
#I = -0.08433328069613605
```