Module: Calcul Scientifique **Classes :** $3^{\text{ème}}$ année **AU: 2023 / 2024**

Correction TP 4: Résolution numérique de l'équation $f(x) = 0$

Objectif de ce TP :

Le but de ce TP est d'implémenter deux méthodes pour la résolution numérique de l'équation $f(x) = 0$ et de comparer ces deux méthodes.

1 Méthode itérative 1: méthode de Dichotomie

On considère une fonction f continue et strictement monotone sur $[a, b]$.

1. Écrire une fonction iterativemethod1(f,a,b,epsilon,Nmax) qui renvoie la valeur approchée du zéro x^* d'une fonction f continue, strictement monotone sur $[a, b]$, selon la méthode de dichotomie , et renvoie aussi le nombre d'itérations.
   • Les arguments de la fonction dichotomie devront être: la fonction f, des réels a et b, avec $a < b$, un réel $\varepsilon > 0$ et $Nmax$ le nombre maximal d'itérations.
   • Le résultat renvoyé doit être composé d'une valeur approchée de x^* à ε près et le nombre d'itérations.
   • On testera au préalable si $f(a)f(b) > 0$, et dans ce cas, on renverra $f(a)$ et $f(b)$ ne sont pas de même signe.
   • On utilise le test d'arrêt $|b_n - a_n| \leq \varepsilon$. (voir le cours de l'analyse numérique pour avoir plus de détails).

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
```

[ ]:
```
def iterativemethod1(f,a,b,epsilon,Nmax):
    k=0
    if f(a)*f(b)>0:
        print("f(a) et f(b) sont de même signe")
    else:
        c=(a+b)/2
        while ((abs(a-b)>epsilon )&(k<Nmax)):
            if f(c)==0:
```

```
            return c,k
        elif f(a)*f(c)<0:
            b=c
            c=(a+b)/2
        else:
            a=c
            c=(a+b)/2
        k=k+1
    return c,k
```

2. On considère la fonction f continue sur $[1,2]$

$$f(x) = \ln(1+x^2) - \sin(x).$$

NB: La fonction logarithme s'écrit sous python comme suit : np.log() ou sp.log().

   (a) Justifier graphiquement que $f(x) = 0$ admet une unique solution dans $[1,2]$.

[ ]:
```
import matplotlib.pyplot as plt
f=lambda x: np.log(1+x**2)-np.sin(x)
a=1
b=2
x=np.linspace(a,b,100)
y=f(x)
plt.plot(x,y)
plt.plot(x,0*x)
plt.title("f(x)")
plt.grid(True)
```

   (b) Tester la fonction iterativemethod1(f,a,b,epsilon,Nmax) pour : $\varepsilon = 10^{-5}$ et $Nmax = 20$.

[ ]:
```
a=1
b=2
epsilon=10**(-5)
f=lambda x: np.log(1+x**2)-np.sin(x)
Nmax=20
[x,k]=iterativemethod1(f,a,b,epsilon,Nmax)
print([x,k])
```

2 Méthode itérative 2

On considère une fonction f continue sur $[a,b]$ telle que $f(a) \neq f(b)$.
On souhaite approcher numériquement la racine d'une équation (E) : $f(x) = 0$ en utilisant le schéma itératif suivant:

$$(MI)_n \; : \begin{cases} x_{n+1} = x_n - \dfrac{b-a}{f(b)-f(a)} f(x_n), \\ x_0 \in \; [a,b], \end{cases}$$

avec (x_n) est une suite convergente vers la solution exacte x^* de (E) ($\lim_{n\to+\infty} x_n = x^*$).

1. (a) Écrire une fonction `iterativemethod2(a,b,x0,f,epsilon)` qui prend les paramètres suivants: a et b les deux extrémités de $[a,b]$, la fonction `f`, la valeur initiale `x0` et la tolérance `epsilon`. Cette fonction doit retourner **deux listes** dont la première contient tous les itérés $x = [x_0, x_1, x_2, \cdots]$ et la deuxième contient $y = [f(x_0), f(x_1), f(x_2), \cdots]$. Dans cette fonction, on utilise le test d'arrêt $\left|\texttt{f}(x_n)\right| \leq$ `epsilon,;`$\forall\, n \geq 0$.

[ ]:
```
def iterativemethod2(a,b,x0,f,epsilon):
    x=[x0]
    y=[f(x0)]
    n=0
    while np.abs(y[n])>epsilon:
        x.append(x[n]-((b-a)/(f(b)-f(a)))*f(x[n]))
        n+=1
        y.append(f(x[n]))
    return x,y
# ou bien
def iterativemethod2(a,b,x0,f,epsilon):
    x=[x0]
    y=[f(x0)]
    c=((b-a)/(f(b)-f(a)))
    while (abs(y[-1])>epsilon):
        x0=x0-c*f(x0)
        x.append(x0)
        y.append(f(x0))
    return x,y
```

(b) Donner l'instruction nécessaire pour déterminer l'expression de la dérivée de f, notée df.

[ ]:
```
import sympy as sp
x=sp.symbols('x')
f=sp.Lambda(x,sp.log(x**2+1)-sp.sin(x))
df=sp.Lambda(x,sp.diff(f(x),x))
df
```

L'expression de la dérivée de f est

$$df(x) = \frac{2x}{1+x^2} - cos(x)$$

(c) Représenter graphiquement df, la fonction dérivée de f sur [1,2].

```
t=np.linspace(1,2,100)
df=lambda t: (2*t/(t**2+1))-np.cos(t)
plt.plot(t,df(t))
plt.grid(True)
plt.xlabel('x-axis ')
plt.ylabel('y-axis')
plt.title('Curve of df')
```

(d) Déduire que (E) admet une unique solution sur $[1,2]$.

```
f'(x)>0 donc f est strictement croissante et de plus f(1)f(2)<0 alors
↪il existe une unique solution de (E) sur [1,2].
```

2. Prenant dans la suite $x_0 = 2$ et `epsilon` $= 10^{-2}$.

   (a) Donner l'instruction qui permet d'afficher la solution approchée de $x^\star$ à `epsilon` près en utilisant la fonction "iterativemethod2".

```
a=1
b=2
x0=2
epsilon=10**(-2)
f=lambda x: np.log(1+x**2)-np.sin(x)
[x,y]=iterativemethod2(a,b,x0,f,epsilon)
print(x[-1])
```

   (b) Trouver le nombre d'itération N pour approcher la solution $x^\star$ à espsilon près.

```
print(len(x)-1)
```

   (c) Remplir le tableau ci-dessous qui contient les trois premiers itérés (deux chiffres seulement après la virgule) du schéma $(MI)_n$.

X	$x_0 = 2$	$x_1 = 1.17$	$x_2 = 1.24$	$x_3 = 1.25$
Y	$f(x_0) = 0.70$	$f(x_1) = -0.05$	$f(x_2) = -0.01$	$f(x_3) = -0.00$

3 Comparaison des deux méthodes

Comparer les deux méthodes de résolution de $f(x) = 0$: "iterativemethod1" et "iterativemethod2", en terme de nombre d'itérations effectuées, pour approcher la solution x^* de la fonction $f(x) = cos(2x) - x^2$ sur l'intervalle $[0, \frac{\pi}{4}]$ pour $Nmax = 10^3$, et $\varepsilon \in \{10^{-n},\ 2 \leq n \leq 8\}$. Que peut-on conclure?

```
a=0
b=np.pi/4
Nmax =10**3
epsilon=1/10**np.arange(2,9)
```

```
f=lambda x: np.cos(2*x)-x**2
Niter1=[]
Niter2=[]
for eps in epsilon:
    [x1,n]=iterativemethod1(f,a,b,eps,Nmax)
    [x2,y]=iterativemethod2(a,b,x0,f,eps)
    Niter1.append(n)
    Niter2.append(len(x2)-1)
plt.plot(epsilon,Niter1,'r-*')
plt.plot(epsilon,Niter2,'bo-.')
plt.xscale('log')
plt.grid(True)
plt.legend(['Method1','Method2'])
```