Objectif :

L'objectif de ce TD est de mettre en œuvre et de comparer les performances de deux méthodes de recherche approchée d'une racine d'une équation algébrique :

- *méthode de dichotomie,*
- *méthode de Newton (avec variante selon la connaissance ou non de la dérivée de la fonction).*

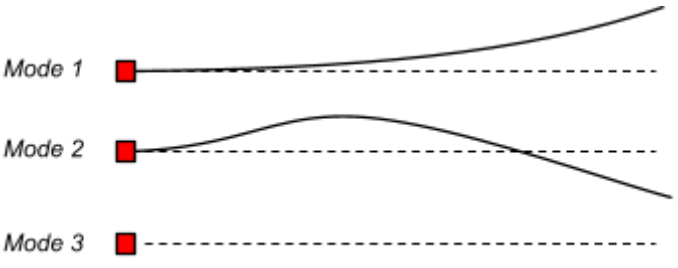
Problème proposé : recherche des fréquences propres des modes de vibrations d'une pale d'hélicoptère.
(*d'après une idée de Philippe FICHOU, Lycée Chateaubriand, RENNES*)

Les pales d'hélicoptère sont assimilables (*en première approximation*) à des poutres encastrées au rotor principal et libres à leurs extrémités. Ces pales peuvent entrer en résonance sous l'effet des actions mécaniques aérodynamiques (*excitation cyclique*), il est donc impératif d'estimer ces fréquences propres afin de vérifier qu'elles ne correspondent pas à celles de la source d'excitation.

Une modélisation du phénomène (*modélisation de type MMC [= Mécanique des Milieux Continus]*) montre que pour une première approche avec un modèle simplifié de poutre encastrée, les modes propres en flexion sont tels que les fréquences propres (f_i) sont liées aux racines (β_i) de l'équation : $\cosh(\beta_i)\cos(\beta_i)+1=0 \ \ (E_1)$,

La relation de correspondance étant : $\beta_i^4 = 4\pi^2 f_i^2 \dfrac{mL^3}{EI}$,

avec :
- m , masse de la pale,
- L , longueur de la pale,
- E , module d'élasticité du matériau,
- I , moment quadratique de la section.



La figure ci-contre illustre les allures des déformées des trois premiers modes propres (le carré symbolise l'encastrement de la pale dans le rotor).

Il n'est pas possible de proposer des solutions formelles (*analytiques*) à cette équation, on doit donc rechercher des solutions approchées par une méthode numérique. On peut alors obtenir les fréquences propres de vibrations d'une pale.

On obtient pour les trois premiers modes propres : $f_1 = 0{,}56\sqrt{\frac{EI}{mL^3}}, \quad f_2 = 3{,}51\sqrt{\quad}, \quad f_3 = 9{,}82\sqrt{\quad}$

Les valeurs exactes des fréquences correspondantes s'obtiendraient avec les caractéristiques physiques d'une pale, par exemple :

$L = 4\,\text{m}, \quad m = 84\,\text{Kg}, \quad E = 80\,\text{GPa}, \quad I = 1\,562\,500\,\text{mm}^4$

A – Prévisualisation de la courbe pour une estimation visuelle de ces racines et de leurs encadrements.

Afin de se rendre compte de la position, du nombre et des valeurs des racines de l'équation $\cosh(\beta)\cos(\beta)+1=0 \quad (E_1)$ *,*

on propose d'en obtenir un tracé sur l'intervalle $\beta \in [0,15]$ *.*

1. Obtenir la courbe représentative de l'équation (E_1) dans l'intervalle demandé en s'aidant du script Python ci-dessous.

```python
# -*- coding: utf-8 -*-

from math import *                # pour pouvoir manipuler 'pi'
import matplotlib.pyplot as plt
import numpy as np

# définition de la fonction dont on cherche les racines: f(u)
def f(u):
   return np.cosh(u) * np.cos(u) + 1

# construction d'une grille en (x) et tracé de la courbe
pas = 0.2
x   = np.arange( 0 , 16 , pas )

plt.figure(1)
plt.plot(x, f(x), '-r')    # pour un tracé de courbe continue en rouge
plt.grid(True)
plt.show()
```

On constate qu'il est difficile d'estimer le nombre (et les valeurs) des racines dans l'intervalle demandé, ce qui était prévisible puisque lorsque $\beta >> 1$ *,* $\cosh(\beta) \approx \frac{1}{2}e^{\beta}$ *, la valeur de la fonction devenant alors très grande !*

2. Proposer une autre formulation de (E_1) et en déduire une seconde fonction qui ait les mêmes racines que (E_1) tout en étant facilement visualisable. Obtenez son tracé.

Quelle est la fonction simple dont se rapproche la courbe après quelques oscillations ?

Obtenir sur un même tracé les deux fonctions ainsi superposées.

(*la seconde fonction sera tracée en pointillés d'une autre couleur, par exemple avec la commande ci-dessous)*

```python
plt.plot(x, g(x), '-- b') # pour un tracé de courbe en traits interrompus bleu
```

3. Quelles valeurs approchées de (β_i) peut-on proposer lorsque $i \geq 3$?
Quelle est l'erreur ainsi commise !? Justifier à l'aide dune petite figure…
Faire les applications numériques avec … Python !

B – Recherche des racines par dichotomie.

La recherche des premières racines n'est pas possible que ce soit de manière formelle ou approchée simplement, on propose donc de mettre en œuvre une méthode numérique, la première méthode est celle de la dichotomie.

On note :
- $f(x)$ *la fonction dont un cherche une racine,*
- (g,d) *l'intervalle de départ dans lequel on cherche une racine,*
- $\varepsilon(=\text{eps})$ *la précision souhaitée (critère d'arrêt).*

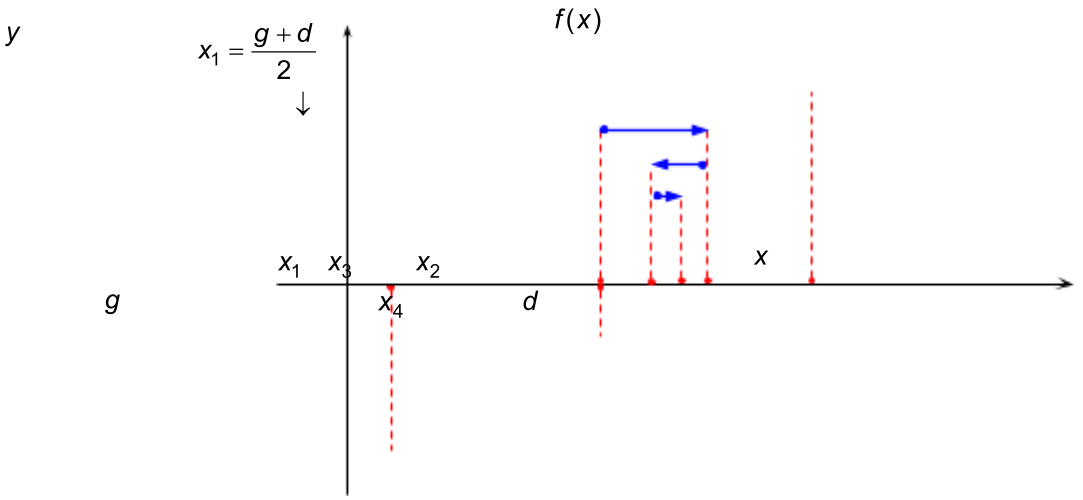


Illustration graphique du principe de la méthode de dichotomie.

4. Compléter votre script sous Python avec la définition d'une méthode de dichotomie qui utilisera les arguments :

```
def recherche_dichotomie (f , g , d , eps):
```

et qui renvoie une valeur approchée de $f(x)=0$ lorsque l'intervalle de recherche devient plus petit que $\varepsilon(=\text{eps})$.

indication: *il est tout à fait possible de reprendre une fonction que vous auriez déjà codée sous Python (!).*

5. Compléter votre script sous python de manière à ce que l'exécution permette d'obtenir :

1) le tracé de la fonction dont on cherche une racine dans un intervalle $[a,b]$ donné au clavier par l'utilisateur, *(cf; indication ci-dessous)*

2) la recherche d'une racine dans un intervalle restreint $[g,d]$ donné au clavier par l'utilisateur *(compte-tenu de ce qu'il aura estimé à partir du tracé obtenu)*

Mettre en œuvre votre démarche et rechercher ainsi les valeurs approchées à $10^{-6}(=1\text{e-}6)$ des quatre premières valeurs de (β_i), vérifier ainsi que les valeurs approchées proposées à la question 3 sont pertinentes(!).

indication : on donne ci-dessous la syntaxe pour permettre l'affectation d'une valeur à partir d'une saisie au clavier lors de l'exécution du programme.

avertissement : pour reprendre l'exécution après l'affichage de la figure(1), il faut "tuer" la fenêtre de la figure(1) !

```
a = float(input("Entrer la borne inférieure de l'intervalle (a) : "))
b = float(input("Entrer la borne supérieure de l'intervalle (b) : "))
```

C – Recherche des racines par méthode de Newton (ou Newton-Raphson).

Isaac Newton – 1643 / 1727

La seconde méthode est celle de Newton dite "méthode de la tangente".

On note : - $f(x)$ *la fonction dont un cherche une racine, et* $fd(x)$ *, sa fonction dérivée (à l'ordre 1),*

- x_0 *, la valeur de départ pour la recherche d'une racine,*
- $\varepsilon (= \text{eps})$ *la précision souhaitée (critère d'arrêt).*

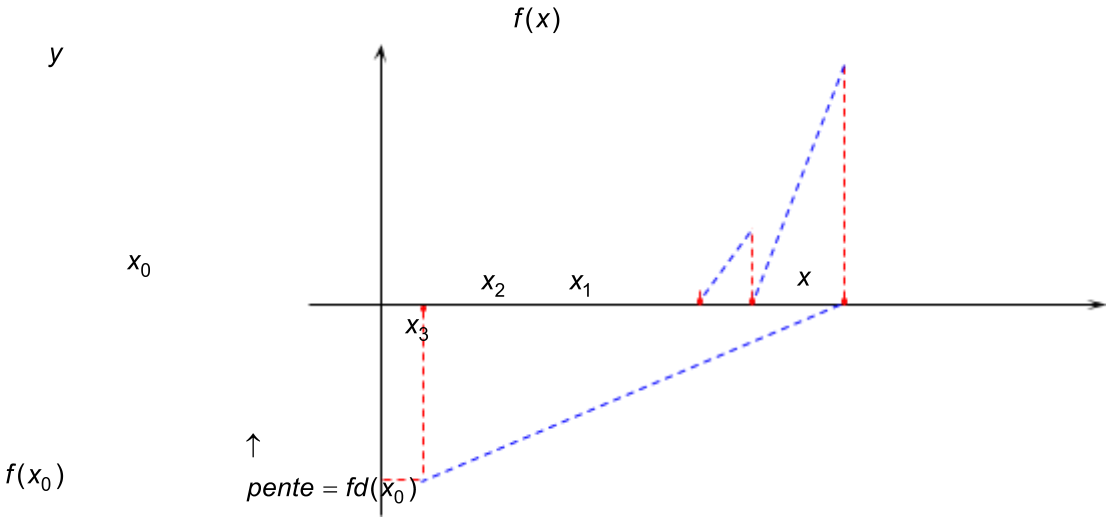


Illustration graphique du principe de la méthode de Newton.

La stratégie est de partir d'une valeur x_0 *(proche de la racine recherchée) et d'obtenir par itération successives des valeurs de plus en plus proches de la racine exacte par utilisation de la tangente à la courbe.*

Or, à partir d'une valeur x_{k-1} *, la pente de la tangente à la courbe en* x_{k-1} *permet d'estimer la valeur* x_k *, puisque graphiquement :*

$$fd(x_{k-1}) = \frac{0 - f(x_{k-1})}{x_k - x_{k-1}} \text{ , soit } x_k = x_{k-1} - \frac{f(x_{k-1})}{fd(x_{k-1})}$$

Le critère d'arrêt peut-être un test sur : - *soit la différence de position entre deux valeurs successives :* $|x_k - x_{k-1}| < \varepsilon$,

- *soit la valeur de la fonction en* x_k *:* $|f(x_k)| < \varepsilon$.

---

6. Compléter votre script sous Python avec la définition d'une méthode de Newton qui utilisera les arguments :

```
def recherche_newton (f , fd , x , eps):
```

indication : on commencera par définir la fonction dérivée $fd(x)$ *(!)*

Faire en sorte que votre script donne après exécution les valeurs approchées à $\varepsilon (= \text{eps})$ près de la racine avec les deux méthodes (*dichotomie et Newton*) et pour une valeur de départ de la méthode de Newton qui soit $x_0 = g$.

7. Afin d'évaluer les performances des deux méthodes, compléter les descriptions de "`recherche_dichotomie`" et "`recherche_newton`" de manière à ce qu'elles renvoient également le nombre d'itérations effectuées dans chaque cas pour obtenir la précision souhaitée.

Recompiler avec une recherche dans l'intervalle $[g = 1, d = 2]$ et conclure.

D – Mise en évidence d'un des inconvénients de la méthode de Newton.

8. Recompiler avec une recherche dans l'intervalle $[g = 4, d = 5]$.

9. Recompiler avec une recherche dans l'intervalle $[g = 3{,}5, d = 5]$, puis dans l'intervalle $[g = 3.25, d = 5]$.
Expliquer pourquoi les racines obtenues avec la méthode de Newton ne sont pas celles attendues (*a priori* !).

Faire un schéma qui illustre que la méthode de Newton peut ne pas converger (*contrairement à la dichotomie* !)

E – Variante de la méthode de Newton.

Une variante de la méthode de Newton est envisagée lorsque la dérivée de la fonction $f(x)$ *n'est pas connue ou difficile à estimer. Dans ce cas, on peut proposer d'utiliser une approximation "fiable" de la dérivée avec l'expression suivante :*

$$fd(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$ *lorsque :* $h << 1$ *(expression approchée de la dérivée avec un schéma dit "centré")*

On peut alors proposer d'utiliser cette approximation en lieu et place de $fd(x)$ *dans la méthode de Newton.*

Cependant, le choix de h *est capital (!), il serait incohérent d'avoir* $h > \varepsilon$ *, on prendra* $h = \varepsilon$ *pour commencer.*

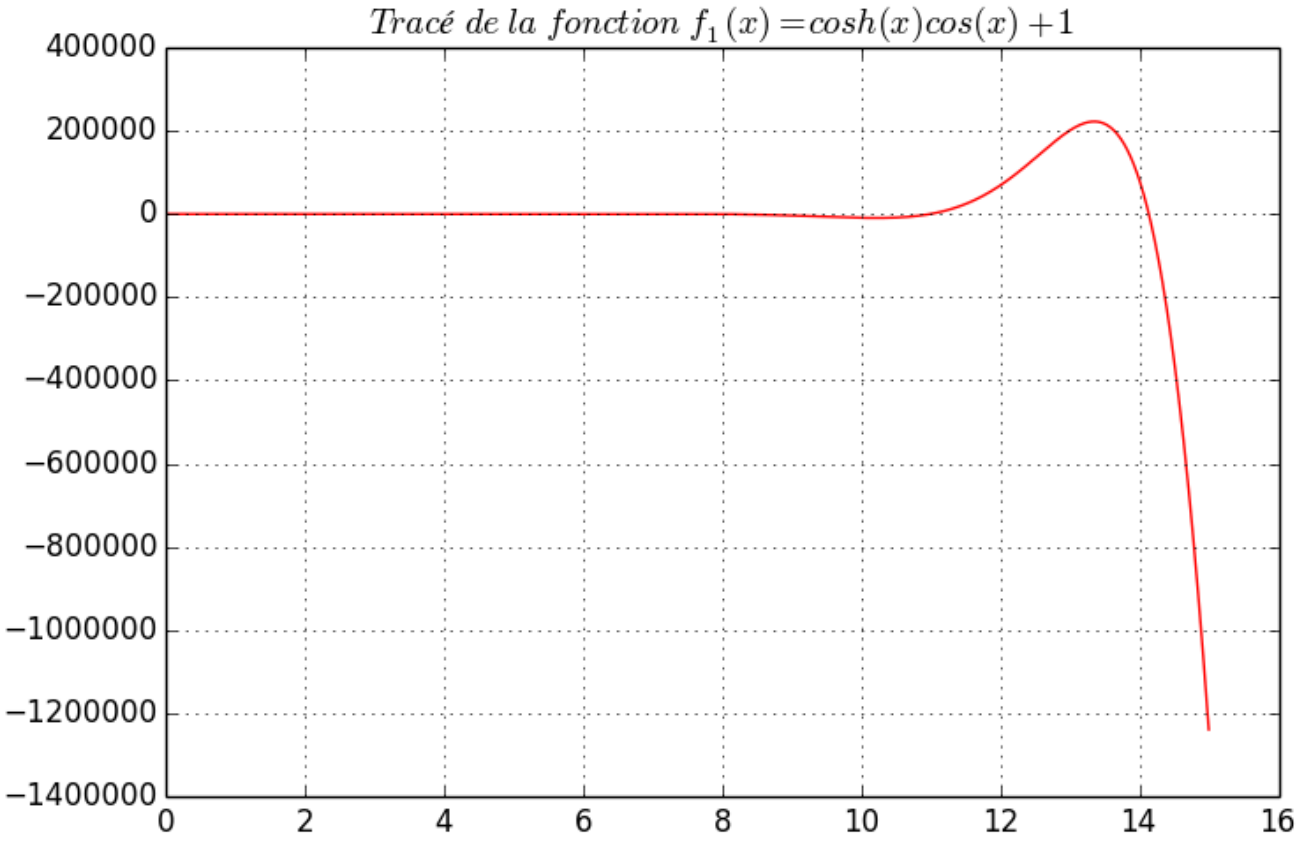
10. Construire une nouvelle variante de la méthode de Newton qui utilisera uniquement les arguments :

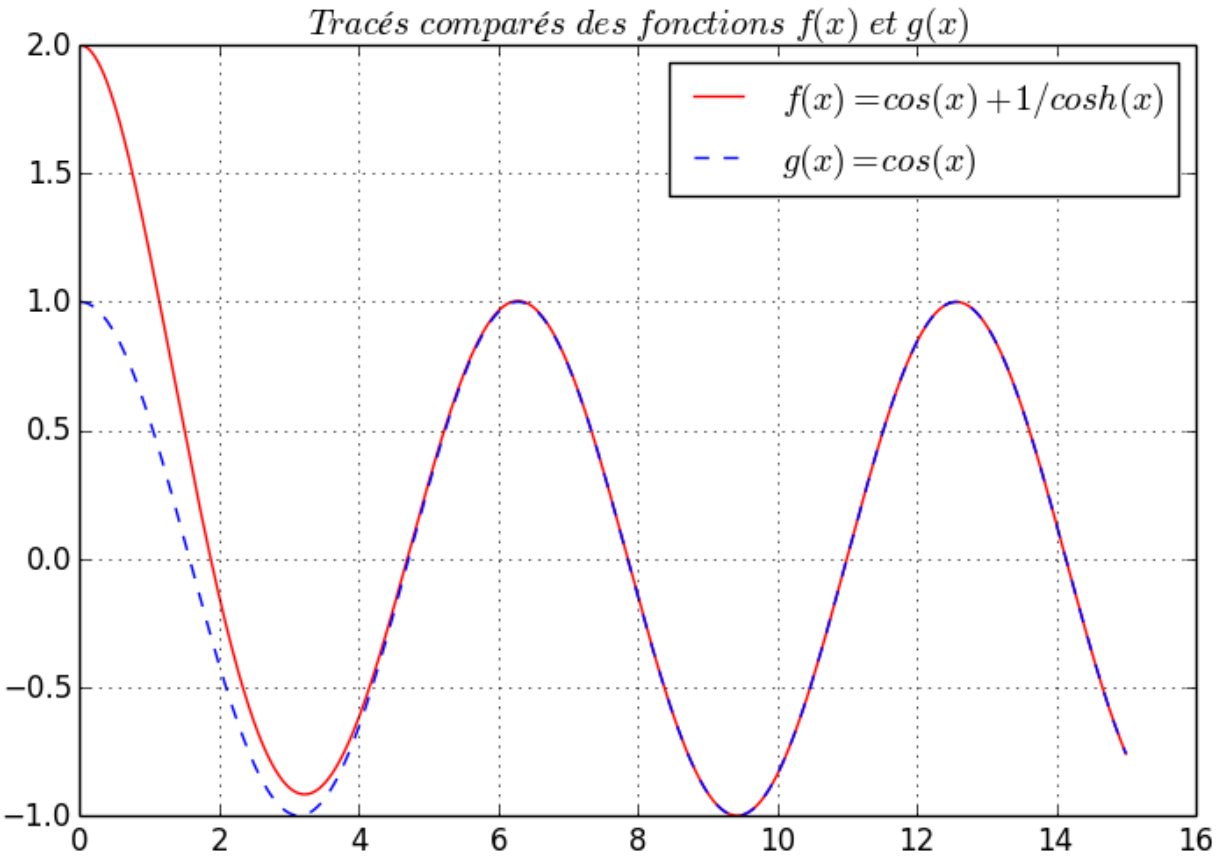
```
def recherche_newton_variante (f , x , eps):
```

dans laquelle vous implémenterez l'utilisation de la dérivée approximée.

Faire en sorte que votre script donne après exécution les valeurs approchées à $\varepsilon(=\texttt{eps})$ près de la racine avec les trois méthodes (*dichotomie, Newton et variante de Newton*).

Recompiler avec une recherche dans l'intervalle $[g = 1, d = 2]$
et constater que les performances de la méthode de Newton avec ou sans variante sont semblables.


Tracé de la fonction $f_1(x) = cosh(x)cos(x) + 1$
400000
200000
0
−200000
−400000
−600000
−800000
−1000000
−1200000
−1400000
0
2
4
6
8
10
12
14
16


Tracés comparés des fonctions $f(x)$ et $g(x)$
$f(x) = cos(x) + 1/cosh(x)$
$g(x) = cos(x)$
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
0
2
4
6
8
10
12
14
16

```python
# -*- coding: utf-8 -*-
#-------------------------------------------------------------------------------
# TP :   Recherche des racines d'une fonction par dichotomie et Newton
#        Application aux modes propres en flexion d'une pale d'hélicoptère
#-------------------------------------------------------------------------------

from math import *  # pour pouvoir manipuler 'pi'
import pylab as plt
import numpy as np

# définition de la fonction dont on cherche une racine et fonction approchée
def f1(x):
    return np.cos(x)*np.cosh(x)+1
def f(x):
    return np.cos(x) + 1/np.cosh(x)
def fd(x):
    return -np.sin(x) -  np.sinh(x)/np.cosh(x)**2
def g(x):
    return np.cos(x)

# tracé de la fonction sur un intervalle jugé pertinent pour prévisualisation
print("défintion d'un intervalle pour prévisualisation de la fonction")
a = float(input("Entrer la borne inférieure de l'intervalle (a) : "))
b = float(input("Entrer la borne supérieure de l'intervalle (b) : "))
X = np.arange(a,b,0.001)

plt.figure(0)
plt.plot(X , f1(X) , '-r')
plt.title(r"$Tracé\ de\ la\ fonction\ cosh(x)cos(x)+1$")
plt.grid(True)

plt.figure(1)
plt.plot(X , f(X) , '-r' , label=r"$f(x)=cos(x)+1/cosh(x)$")
plt.plot(X , g(X) , '--b', label=r"$g(x)=cos(x)$")
plt.title(r"$Tracés\ comparés\ des\ fonctions\ f(x)\ et\ g(x)$")
plt.legend()
plt.grid(True)
plt.show()


# construction d'une procédure de dichotomie dans l'intervalle (d,g)
def recherche_dicho (f , g , d , eps):
    j = 0
    if f(g)*f(d) >0 :
        return print(" la méthode ne peut pas aboutir dans l'intervalle proposé ")
    while abs(g-d) > eps :
        j += 1
        if f(g)*f((d+g)/2) > 0:
            g = (d+g)/2
        else :
            d = (d+g)/2
    return (d,j)
```

```python
# construction d'une procédure de Newton et de sa variante à partir de la valeur (x)

def recherche_newton (f , fd , x , e):
    j = 0
    while abs(f(x) / fd(x)) > e:
        j += 1
        if fd(x) ==0:
            return ("La méthode de Newton ne peut aboutir, la dérivée s'annule !")
        x=x-f(x)/fd(x)
    return (x,j)


def recherche_newton_variante (f , x , e):
    j = 0
    while abs( 2*f(x)/(f(x+e)-f(x-e)) ) > 1:
        j += 1
        if (f(x+e)-f(x-e)) == 0:
            return ("La méthode de Newton ne peut aboutir, la dérivée s'annule !")
        x=x-2*e*f(x)/ (f(x+e)-f(x-e))
    return (x,j)


# application des méthodes de dichotomie et de Newton

g = float(input("Entrer la borne inférieure de l'intervalle (g) : "))
d = float(input("Entrer la borne supérieure de l'intervalle (d) : "))
e = 1e-6


print( "intervalle de recherche : ", (g , d) , "précision souhaitée = ", e )

print( "dichotomie , racine obtenue et nb d'itérations :" )
print( recherche_dicho (f , g , d ,e) )

print( "Newton, racine obtenue et nb d'itérations :" )
print( recherche_newton (f , fd , g ,e) )

print( "Newton (variante sans dérivée), racine obtenue et nb d'itérations :" )
print( recherche_newton_variante (f , g , e) )
```