TP n°3 : Résolution numérique d'équations

CORRECTION

Objectif : L'objectif de ce T.P. est de résoudre des équations du type $f(x) = 0$ en utilisant différentes méthodes.

I. Major League Baseball

Soit une sphère de masse $m = 140\ g$ et de diamètre $d = 7{,}5\ cm$ en chute libre dans le champ de pesanteur de norme constante $g = 9{,}8\ m.s^{-2}$ en présence de frottements non négligeables de l'air. **On cherche à connaitre la vitesse limite v supposée positive atteignable par la balle.**

En appliquant le principe fondamental de la dynamique, on peut montrer que la vitesse limite de la sphère est donnée par l'équation :

$$v^2 = \frac{2mg}{\rho S c_D}$$

Avec :

- $\rho = 1{,}2$ la densité de l'air
- $S = \frac{\pi d^2}{4}$ la surface frontale de la sphère
- $c_D = \frac{24}{R_e} + \frac{6}{1+\sqrt{R_e}} + \frac{2}{5}$ pour une sphère, le coefficient de traînée obtenu expérimentalement.
- $R_e = \frac{\rho v d}{\mu}$ le nombre de Reynolds décrivant le comportement du fluide.
- $\mu = 1{,}8.10^{-5} kg.s^{-1}.m^{-1}$ la viscosité dynamique du fluide.

1) En combinant les expressions précédentes montrer que l'on peut aboutir à une équation de la forme :

$$(R_e)^2 c_D = \alpha$$

Avec α indépendant de R_e et v.

2) En déduire l'équation $g(R_e) = 0$ à résoudre pour déterminer R_e. Expliquer ensuite comment procéder pour déterminer v une fois cette équation résolue.

3) En remarquant que l'équation précédente peut aussi s'écrire $c_D R_e = \frac{\alpha}{R_e}$, représenter graphiquement sur une même figure les deux membres de l'équation en fonction de R_e. En déduire sur quel intervalle il sera pertinent de lancer la résolution numérique de l'équation.

Données du problème :	
Expression du coefficient α :	$\alpha = \frac{8\rho m g}{\pi \mu^2}$
Équation à résoudre :	$0 = \frac{2}{5}R_e^2 + 24R_e + \frac{6R_e^2}{1+\sqrt{R_e}} - \alpha$
Intervalle de résolution :	$I = [160000, 180000]$

II. Résolution du problème par la méthode de Newton

La méthode de Newton permet tout comme la méthode dichotomique de résoudre des équations non linéaires du type $f(x) = 0$. Pour appliquer cette méthode on considère une fonction f dérivable sur un intervalle I. Au voisinage de tout point de la courbe représentative de la fonction f on estime ensuite que celle-ci est approximable par sa tangente en ce point.

Au point de coordonnées $(x_n, f(x_n))$ de la courbe représentative de f, sa tangente a pour équation :

$$y = f'(x_n)(x - x_n) + f(x_n)$$

4) À condition que la dérivée ne soit pas nulle quelles sont les coordonnées du point d'intersection de la tangente et de l'axe des abscisses ?

En nommant x_{n+1} l'abscisse du point précédemment déterminé on obtient une relation de récurrence. À chaque itération la nouvelle abscisse calculée est plus proche du zéro de la fonction f. On décide alors d'arrêter la recherche lorsque l'abscisse x_{n+1} est suffisamment proche de x_n. Cela signifie que l'on ne progresse plus et que le zéro de la fonction est bien approximé.

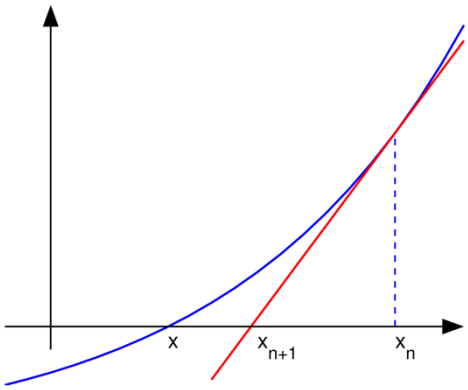


Figure 1 - Courbe d'une fonction f et sa tangente en un point d'abscisse x_n

Relation de récurrence pour la méthode de Newton

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

5) Coder en Python la fonction `Newton(f,der,a,eps)` permettant de trouver la valeur de R_e en utilisant la méthode de Newton.

Algorithme de résolution de $f(x) = 0$ par la méthode de Newton

```
Fonction Newton
Entrées f : une fonction.
        a : une des bornes de l'intervalle de recherche
        eps : la précision
        der : la dérivée de la fonction f
Sortie La solution de l'équation ou un encadrement de celle-ci
-----------------------------------------------------------------------------------------

Créer une variable ecart qui vaut 10 fois la précision souhaitée
Créer une variable x qui vaut initialement a

Tant que l'écart est supérieur à la précision souhaitée Faire
    Définir la variable x1 grâce à la relation de récurrence
    Calculer le nouvel écart correspondant à la différence absolue entre x1 et x
    Transférer la valeur de x1 dans x

Retourner la solution de l'équation x1
```

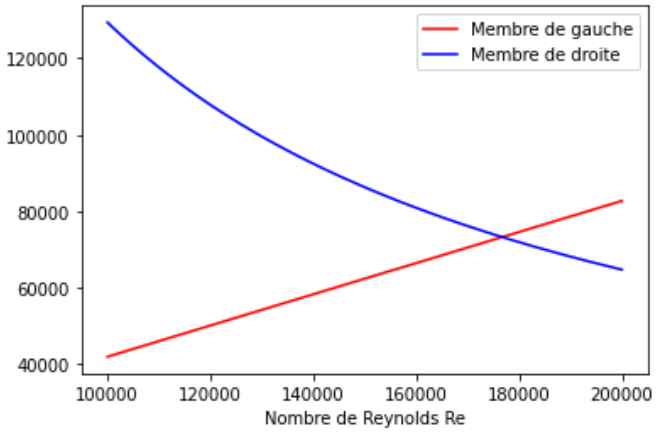
Détermination de l'intervalle de résolution :

```
from math import *
import matplotlib.pyplot as plt

def trainee(u):
    return (24/u)+6/(1+sqrt(u))+(2/5)

t = [i for i in range(100000,200000)]
m1 = [trainee(i)*i for i in t]
m2 = [alpha/i for i in t]

plt.figure()
plt.plot(t,m1,'r')
plt.plot(t,m2,'b')
plt.legend(['Membre de gauche','Membre de droite'])
plt.xlabel("Nombre de Reynolds Re")
```



On constate que la solution se trouve entre les graduations 160000 et 180000.

Programmation de la méthode de Newton :

```
def Newton(f,der,a,eps):
    ecart = 10*eps
    x = a
    while ecart > eps:
        x1 = x - f(x)/der(x)
        ecart = abs(x1-x)
        x = x1
    return x1
```

6) Calculer la dérivée de la fonction g déterminée à la question 2 puis coder en Python les fonctions `g(u)` et `derivee(u)` qui renvoient respectivement la valeur de la fonction g et de sa dérivée appliquées à une valeur quelconque `u`.

$g(R_e) = \frac{2}{5}R_e^2 + 24R_e + \frac{6R_e^2}{1+\sqrt{R_e}} - \alpha$	$\frac{dg(R_e)}{dR_e} = \frac{4}{5}R_e + 24 + \frac{12R_e+9R_e^{\frac{3}{2}}}{(1+\sqrt{R_e})^2}$

Programmation des fonctions adaptées à notre cas :

```
def g(u):
    return (2/5)*u*u+24*u+(6*u*u)/(1+sqrt(u))-alpha
```

```
def derivee(u):
    return (4/5)*u+24+(12*u*-9*u**(3/2))/((1+sqrt(u))**2)
```

7) Utiliser votre fonction Newton sur les données du problème pour déterminer la valeur numérique de R_e à 10^{-2} près. En déduire la valeur de v la vitesse limite de la balle.

$R_e = 176712{,}9832859736$	$v = 127{,}23$ km/h

Appel à la fonction pour résoudre le problème :

```
rho = 1.2
m = 0.14
d = 0.075
g = 9.8
mu = 1.8*10**(-5)
alpha = (8*rho*m*g)/(pi*mu*mu)

solution = Newton(g,derivee,180000,0.01)
vms = mu*solution/(rho*d)
vkmh = 3.6*vms
print("La vitesse limite de la balle est de "+str(vkmh)+" km/h.")
```

8) Comparer la vitesse obtenue à celle du record au lancer lors d'un match professionnel de base-ball.

Le record de vitesse pour un lanceur de ligue majeur de baseball est de 169,1 km/h. Plus rapide que la vitesse limite précédente puisque le lanceur donne une vitesse initiale à sa balle.

9) Reprendre votre fonction de résolution d'équation par la méthode dichotomique avec limitation du nombre d'itérations codée lors de la séance précédente. A l'aide d'un compteur judicieusement placé dans vos fonctions Newton et Dichotomie comparer alors les nombres d'itérations nécessaires à chaque méthode pour résoudre le problème avec la même précision que précédemment. Refaire l'opération pour une précision $\varepsilon = 10^{-11}$. Remettre la précision initiale et modifier la borne de départ à 1000000. Conclure.

Newton	Dichotomie

`def Newton_compteur(f,der,a,eps):`
`  ecart = 10*eps`
`  x = a`
`  n = 0`
`  while ecart > eps:`
`  x1 = x - f(x)/der(x)`
`  ecart = abs(x1-x)`
`  x = x1`
`  n+=1`
`  return x1,n`
	`def dichotomie_compteur(f,a,b,eps):`
	`  n = 0`
	`  while b-a>eps and n < 1000000:`
	`  n +=1`
	`  c = (a+b)/2`
	`  if f(a)*f(c)<0:`
	`  b = c`
	`  elif f(a)*f(c)>0:`
	`  a = c`
	`  else:`
	`  return c,n`
	`  return b,n`

Pour une précision de 10^{-2} on obtient la solution en 3 itérations. Pour une précision de 10^{-2} et un départ à 1000000 on obtient la solution en 7 itérations. Pour une précision de 10^{-11} on obtient la solution en 4 itérations.	Pour une précision de 10^{-2} on obtient la solution en 21 itérations. Pour une précision de 10^{-2} et un départ à 1000000 on obtient la solution en 27 itérations. Pour une précision de 10^{-11} dépasse le nombre maximal d'itérations de 10^6.

Conclusion : La méthode de Newton est plus rapide que la méthode dichotomique.

Bonus : Copier et coller votre fonction Newton et renommez la nouvelle fonction Newton2. Modifier alors cette nouvelle fonction pour qu'à chaque itération elle trace sur une même figure la nouvelle tangente à la courbe au point d'abscisse x_n et ainsi illustrer la progression de la méthode vers la solution. Appliquer votre nouvelle fonction Newton2 à la fonction $h(x) = x^2 - 16$ avec 100 pour point de départ et une précision de 10^{-5}pour simplifier le visionnage.

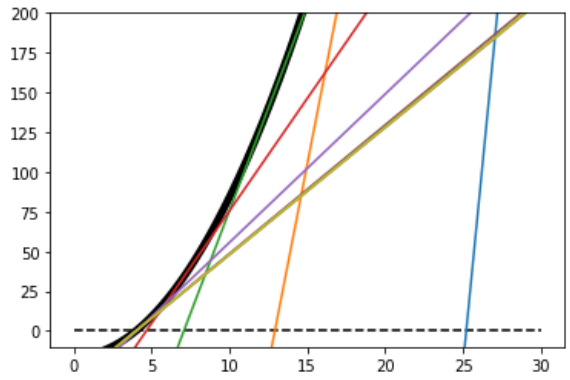
```
def Newton2(f,der,a,eps):
    t = [0.01*i for i in range(0,3000)]
    c = [der(a)*(i-a)+f(a) for i in t]
    cc = [f(i) for i in t]
    plt.figure()
    plt.ylim((-10,200))
    plt.plot(t,cc,'k',linewidth=5)
    plt.plot([0,30],[0,0],'k--')
    ecart = 10*eps
    x = a
    while ecart > eps:
        x1 = x - f(x)/der(x)
        ecart = abs(x1-x)
        x = x1
        c = [der(x1)*(i-x1)+f(x1) for i in t]
        plt.plot(t,c)

    return x1


def fonction_test(u):
    return u**2-16

def derivee_test(u):
    return 2*u

Newton2(fonction_test,derivee_test,100,0.00001)
```



Comme attendu on constate que les tangentes coupent l'axe des abscisses de plus en plus près du zéro réel de la fonction.