TP7_correction

March 6, 2017

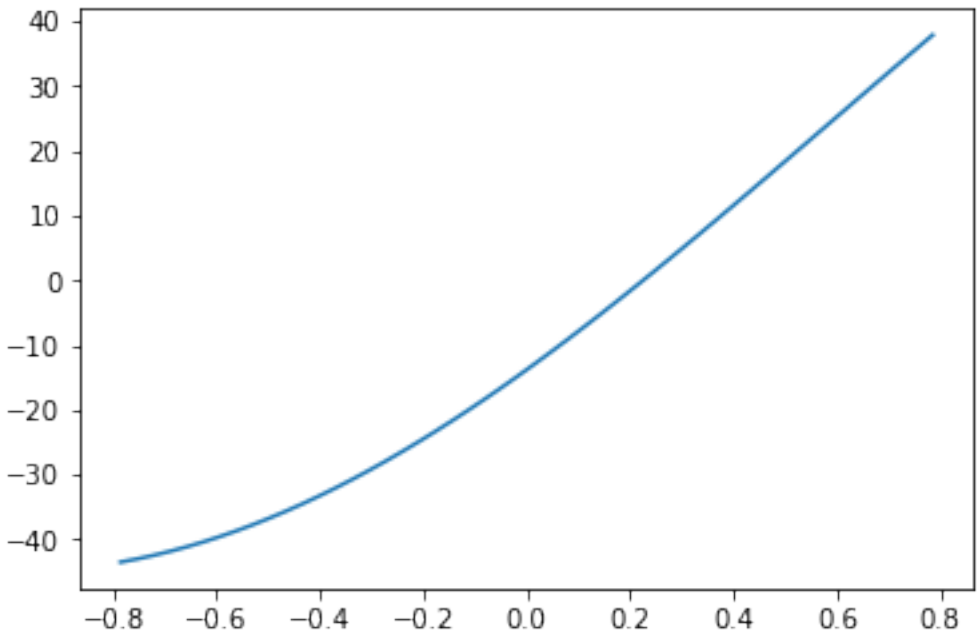
#
CORRECTION DU TP 7 D'INFORMATIQUE
Activité 1
Q1)

In [12]:
```
from pylab import *
from math import sqrt
```

In [13]:
```
A=37.5
B=-57.5
C=23.7
D=sqrt(A*A+B*B)
alpha0=acos(C/D)-acos(A/D)
def f(x):
    return C-(A*cos(x)+B*sin(x))
```

In [14]:
```
X=linspace(-pi/4,pi/4)
plot(X,f(X))
```

Out[14]: [<matplotlib.lines.Line2D at 0x7f43a42da630>]



Q2)
On va procéder par dichotomie, on commence donc par écrire l'algorithme correspondant

```
In [20]: def dicho(f,a,b,n):     #algorithme de dichotomie pour le zéro de f entre a
             u=a
             v=b
             for k in range(1,n):
                 if f(u)*f((u+v)/2)<0:
                    u=u
                    v=(u+v)/2
         #           print u,v
                 else :
                     u=(u+v)/2
                     v=v
         #            print k,u,v
             return [u,v]
```

On peut obtenir les informations sur la précision machine avec le code suivant :

```
In [21]: import sys
         sys.float_info
```

```
Out[21]: sys.float_info(max=1.7976931348623157e+308, max_exp=1024, max_10_exp=308,
```

On prend approximativement 10^{-16} comme valeur de précision machine. On peut tester le code en regardant le résultat de la dichotomie à la 40ème itération :

```
In [22]: print(dicho(f,-pi/2,pi/2,40))
```

```
[0.22540612222974982, 0.22540612223546436]
```

La solution analytique exacte est :

```
In [23]: print(alpha0)
```

```
0.2254061222313578
```

Q3) Pour estimer la précision de la solution, on ve écrire un script qui va nous retourner le nombre d'itérations nécessaires pour obtenir une approximation à la précision demandée, précision que l'on poussera donc jusqu'à la valeur 10^{-16}.

```
In [25]: for k in range(0,17):
             n=0
             while abs(dicho(f,-pi/2,pi/2,n)[0]-alpha0)>10**(-k):        #Notre algo
                 #prend le premeir élément de ce-dernier
                 n=n+1
             print(n,"precision",10**(-k))
```

```
2 precision 1
5 precision 0.1
8 precision 0.01
13 precision 0.001
16 precision 0.0001
16 precision 1e-05
20 precision 1e-06
23 precision 1e-07
28 precision 1e-08
31 precision 1e-09
31 precision 1e-10
39 precision 1e-11
42 precision 1e-12
45 precision 1e-13
45 precision 1e-14
45 precision 1e-15
53 precision 1e-16
```

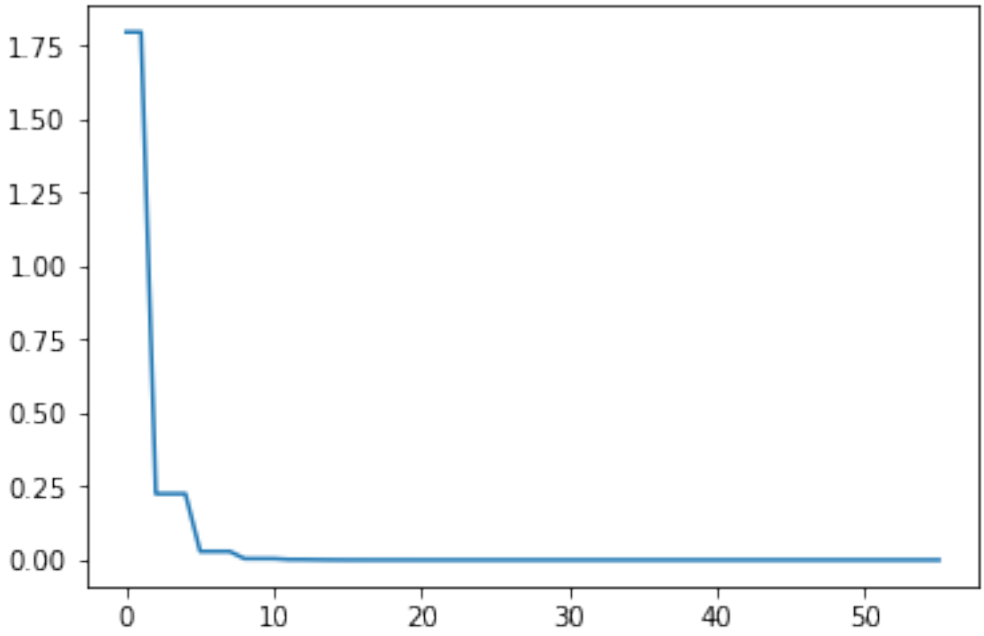
On peut constater que le nombre d'itérations supplémentaires nécessaires pour gagner une décimale de précision supplémentaire tourne en moyenne autour de 3-4. C'est un comportement en fait caractéristique d'une méthode de convergence dite linéaire.

Q4) On crée une liste L qui regroupe l'erreur entre la solution approchée et la solution exacte à chaque itération jusqu'à 55 (itération limite correspondant en gros au niveau de tolérance machine) :

```
In [28]: L=[]
         for k in range(0,56):
             erreur=abs(dicho(f,-pi/2,pi/2,k)[0]-alpha0)
             L.append(erreur)
```

On met en abscisses les valeurs de n correspondant aux itérations et en ordonnée l'erreur correspondante :

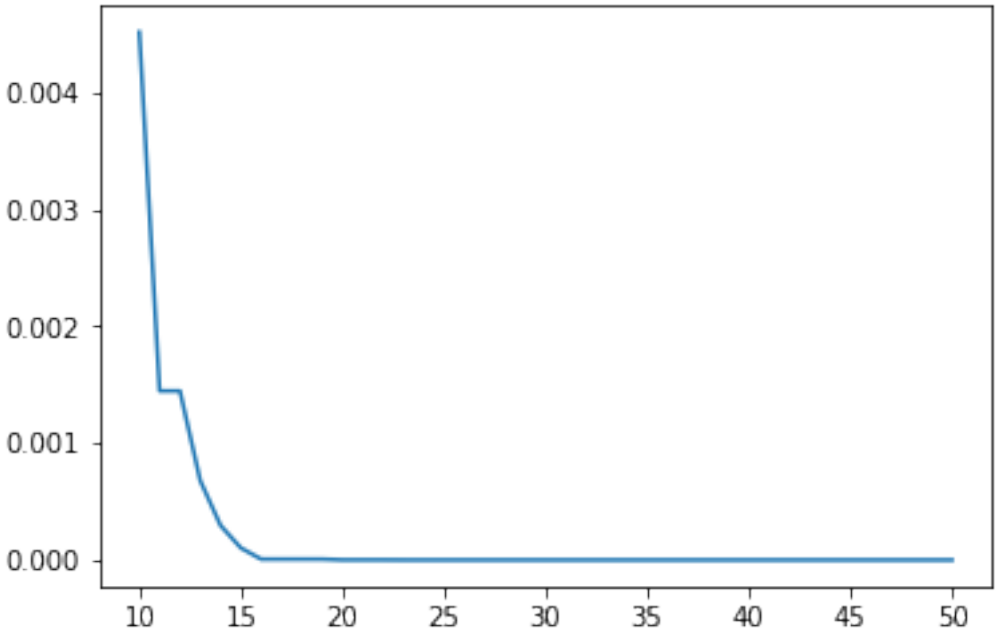
```
In [31]: plot(range(0,56),L)

Out[31]: [<matplotlib.lines.Line2D at 0x7f43a411c080>]
```



On zoome peut être un peu sur la fin :

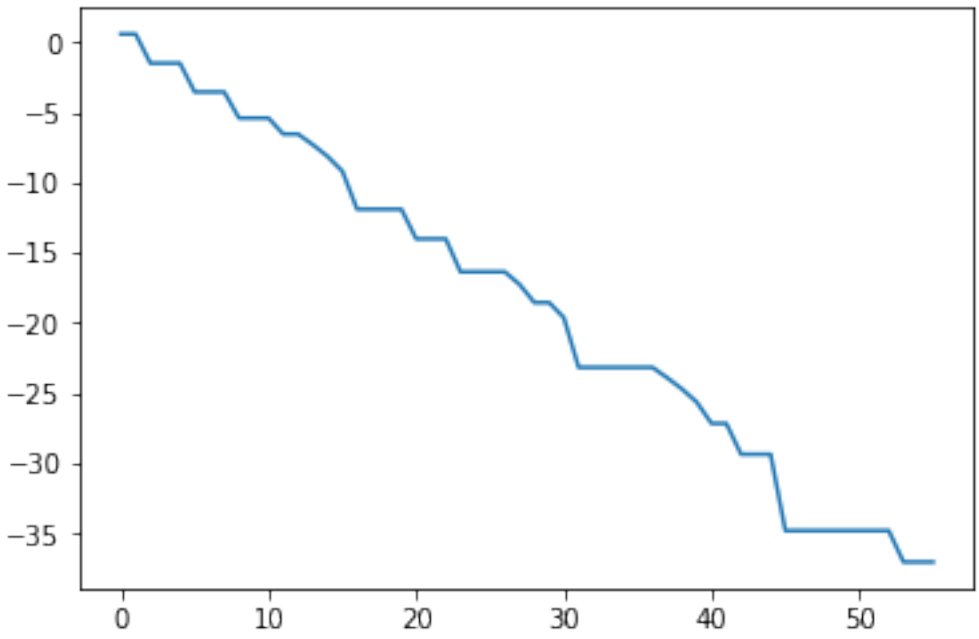
```
In [32]: plot(range(10,51),L[10:51])
```

```
Out[32]: [<matplotlib.lines.Line2D at 0x7f43a4080d30>]
```



Ce n'est pas demandé mais on peut regarder le logarithme des précisions en ordonnée :

```
In [33]: plot(range(0,56),log(L))

Out[33]: [<matplotlib.lines.Line2D at 0x7f43a406b978>]
```



Il y a quelques sauts mais nous ne sommes pas loin d'une droite (c'est pour cel que l'on parle de convergence linéaire).

Q5) On reprend tout ceci avec la méthode de Newton, que l'on commence par écrire :

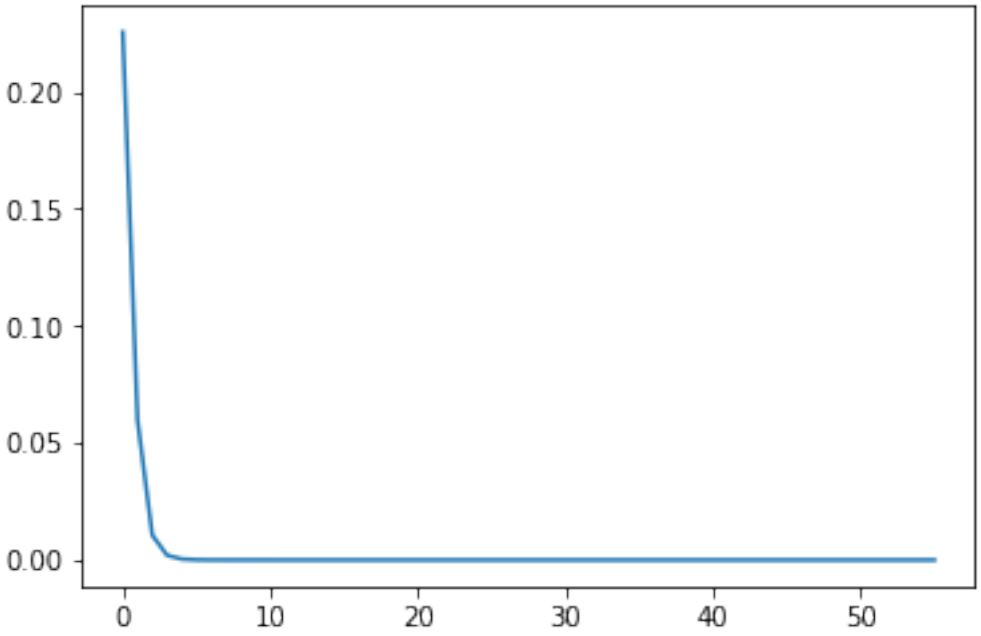
```
In [34]: from scipy import misc

         def newton(f,a,n):
             u=a
             for k in range(1,n+1):
                 u=u-f(u)/misc.derivative(f,u)#fonction derivative du sous_module m
             return(u)
```

puis que l'on teste :

```
In [35]: newton(f,0,5)

Out[35]: 0.22547765414758747

In [36]: print(alpha0)

0.2254061222313578
```

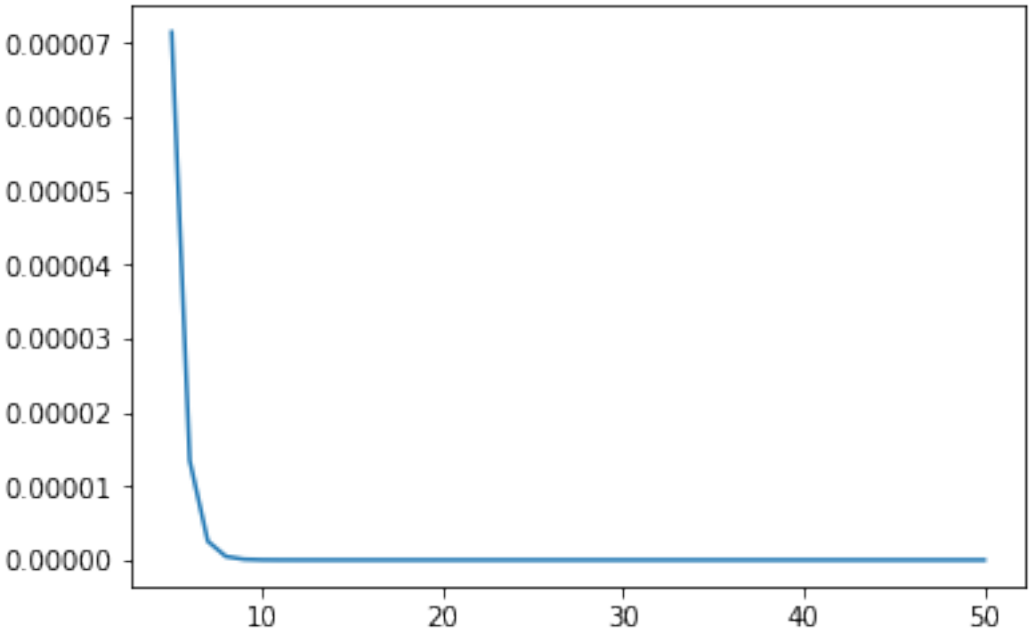
On peut remarquer qu'il y a déjà 4 décimales exactes au bout de 5 itérations, alors qu'il nous en fallait autour de 16 pour la dichotmie. Poursuivons notre analyse :

```
In [37]: for k in range(0,17):
             n=0
             while abs(newton(f,0,n)-alpha0)>10**(-k):
                 n=n+1
             print(n,"precision",10**(-k))

0 precision 1
1 precision 0.1
3 precision 0.01
4 precision 0.001
5 precision 0.0001
7 precision 1e-05
8 precision 1e-06
9 precision 1e-07
11 precision 1e-08
12 precision 1e-09
14 precision 1e-10
15 precision 1e-11
16 precision 1e-12
18 precision 1e-13
19 precision 1e-14
20 precision 1e-15
22 precision 1e-16


In [38]: L2=[]
         for k in range(0,56):
             erreur=abs(newton(f,0,k)-alpha0)
             L2.append(erreur)

In [40]: plot(range(0,56),L2)

Out[40]: [<matplotlib.lines.Line2D at 0x7f43d7ca09e8>]
```



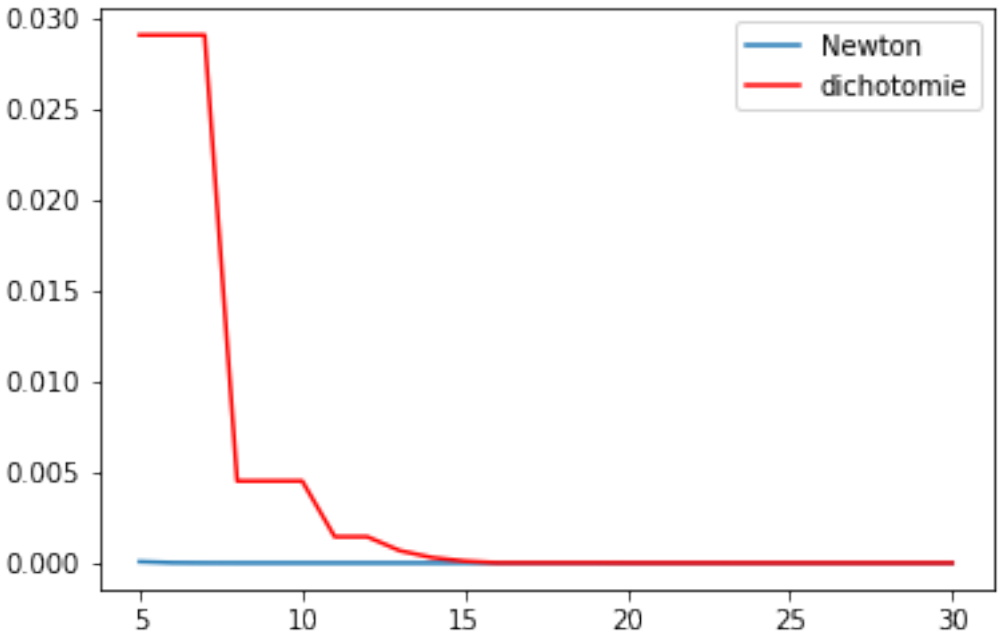
On zoome :

```
In [41]: plot(range(5,51),L2[5:51])

Out[41]: [<matplotlib.lines.Line2D at 0x7f43d7c08278>]
```



Pour finir, nous mettons les deux erreurs obtenues sur un même graphique :

```
In [51]: plot(range(5,31),L2[5:31],label="Newton")
         plot(range(5,31),L[5:31],'r',label="dichotomie")
         legend()

Out[51]: <matplotlib.legend.Legend at 0x7f439a377eb8>
```



Le graphique semble donc confirmer une vitesse de convergence supérieure pour la méthode de Newton.

On finit notre analyse par une comparaison des temps de calculs en faisant évoluer les itérations :

```
In [58]: from time import time
         for k in range(5,30):
             print("Itération",k)
             print("dicho")
             t1=time()
             dicho(f,-pi/2,pi/2,k)
             t2=time()
             print(t2-t1)

             print("newton")
             t1=time()
             newton(f,0,k)
             t2=time()
             print(t2-t1)
```

```
Itération 5
dicho
5.340576171875e-05
newton
0.00023603439331054688
Itération 6
dicho
3.147125244140625e-05
newton
0.00019788742065429688
Itération 7
dicho
3.457069396972656e-05
newton
0.0002219676971435547
Itération 8
dicho
3.8623809814453125e-05
newton
0.00046896934509277344
Itération 9
dicho
8.058547973632812e-05
newton
0.0005130767822265625
Itération 10
dicho
9.012222290039062e-05
newton
0.0005271434783935547
Itération 11
dicho
9.751319885253906e-05
newton
0.0005609989166259766
Itération 12
dicho
0.00010395050048828125
newton
0.0006110668182373047
Itération 13
dicho
0.00011205673217773438
newton
0.0006811618804931641
Itération 14
dicho
0.00012755393981933594
```

```
newton
0.0006933212280273438
Itération 15
dicho
0.00012755393981933594
newton
0.0007672309875488281
Itération 16
dicho
0.0001354217529296875
newton
0.0007991790771484375
Itération 17
dicho
0.00015139579772949219
newton
0.0008401870727539062
Itération 18
dicho
0.00015354156494140625
newton
0.0008828639984130859
Itération 19
dicho
0.00016069412231445312
newton
0.0009403228759765625
Itération 20
dicho
0.00016427040100097656
newton
0.0010132789611816406
Itération 21
dicho
0.00017762184143066406
newton
0.0010294914245605469
Itération 22
dicho
0.00018525123596191406
newton
0.0011124610900878906
Itération 23
dicho
0.0001983642578125
newton
0.00119781494140625
Itération 24
```

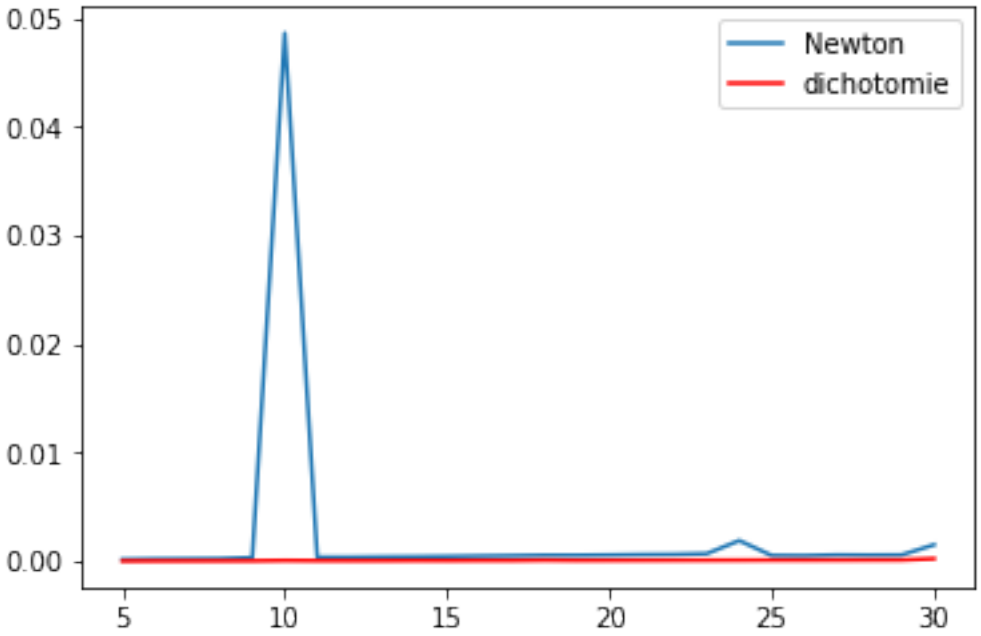
```
dicho
0.0002148151397705078
newton
0.0011756420135498047
Itération 25
dicho
0.0002155303955078125
newton
0.001285552978515625
Itération 26
dicho
0.0002377033233642578
newton
0.0013124942779541016
Itération 27
dicho
0.0002460479736328125
newton
0.0013813972473144531
Itération 28
dicho
0.0001192454528808594
newton
0.0007741451263427734
Itération 29
dicho
0.00012159347534179688
newton
0.0007901191711425781
```

On peut remarquer que Newton est plus rapide au départ mais semble plus long pour de grandes itérations. Le graphique suivant serait peut-être plus parlant :

```
In [57]: Tdicho,Tnewton=[],[]    #on stocke les temps d'exécutions des deux méthode
         for k in range(1,51):

             t1=time()
             dicho(f,-pi/2,pi/2,k)
             t2=time()
             Tdicho.append(t2-t1)


             t1=time()
             newton(f,0,k)
             t2=time()
             Tnewton.append(t2-t1)

In [60]: plot(range(5,31),Tnewton[5:31],label="Newton")
```

```
plot(range(5,31),Tdicho[5:31],'r',label="dichotomie")
legend()
```

```
Out[60]: <matplotlib.legend.Legend at 0x7f439a1204a8>
```



Activité 2

Q)

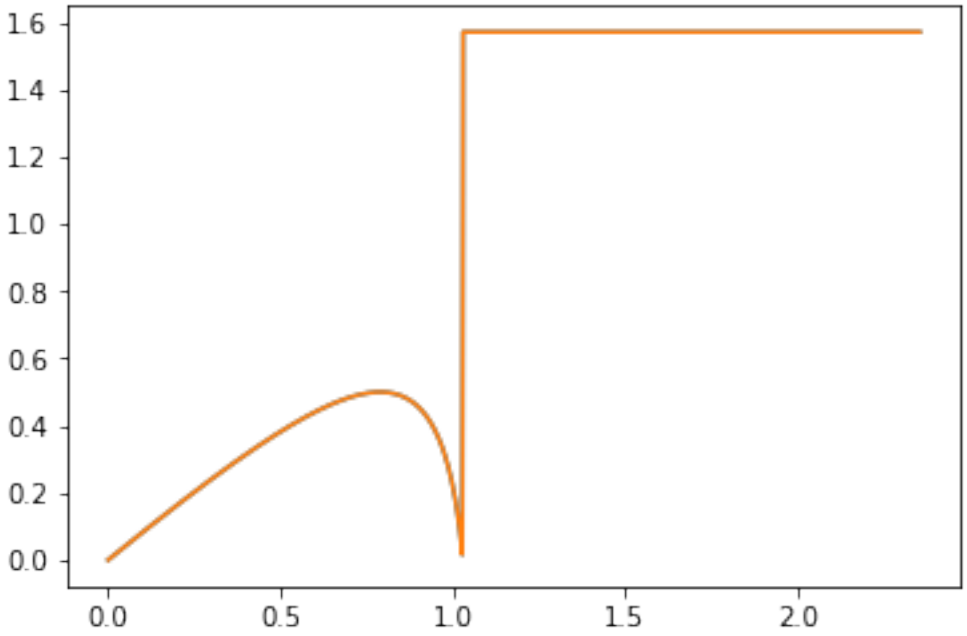
```
In [63]: a=100
         b=260
         c=20
         d=324.22
         l=280
```

```
def f2(x):
    return (b*c-a*d+d*l*cos(theta_m)-c*l*sin(theta_m))*cos(x)+(-b*d-a*c+c*
```

```
X=linspace(0,3*pi/4,1000)     #Liste des valeurs de thetam
Y=[]                          #Liste des résulats obtenus par dichotomie po
for k in range(0,1000):
    theta_m=X[k]
    Y.append(dicho(f2,0,pi/2,30))
    #print(theta_m,dicho(f2,0,pi/2,30))
```

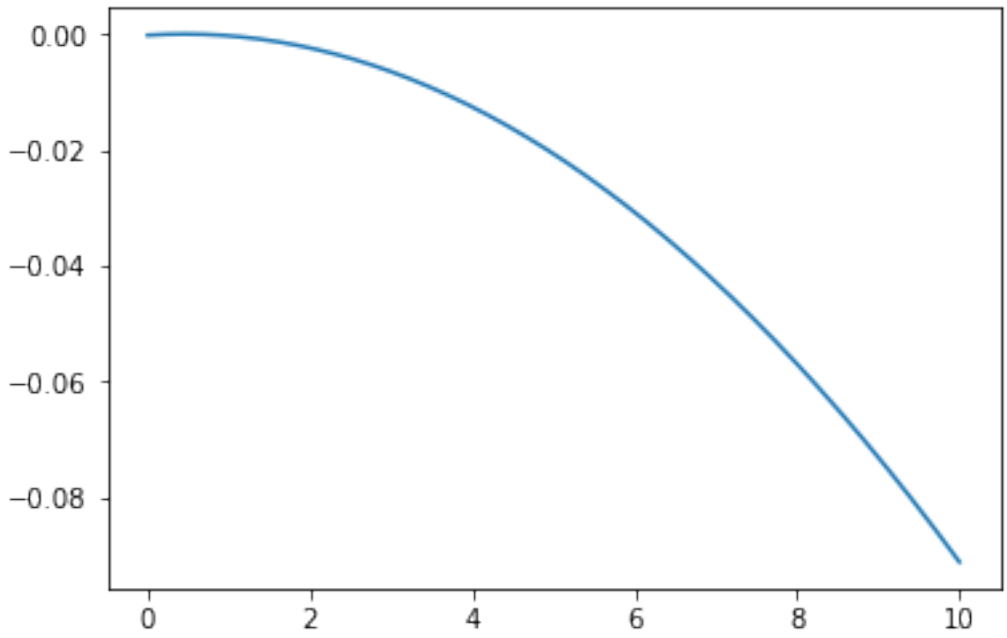
On trace la courbe obtenue :

```
In [64]: plot(X,Y)

Out[64]: [<matplotlib.lines.Line2D at 0x7f439a032f28>,
          <matplotlib.lines.Line2D at 0x7f439a0381d0>]
```



Activité 3

```
In [65]: alpha=10**(-12)
         mu=10**(-3)
         gamma=0.4
         beta=40
         E=0.35
         R=2000


         def f3(x):
             return alpha*(exp(x/beta)-1)-mu*x*(x-gamma)-E/R+x/R

In [67]: X=linspace(0,10,1000)
         plot(X,f3(X))

Out[67]: [<matplotlib.lines.Line2D at 0x7f4399fb3a20>]
```



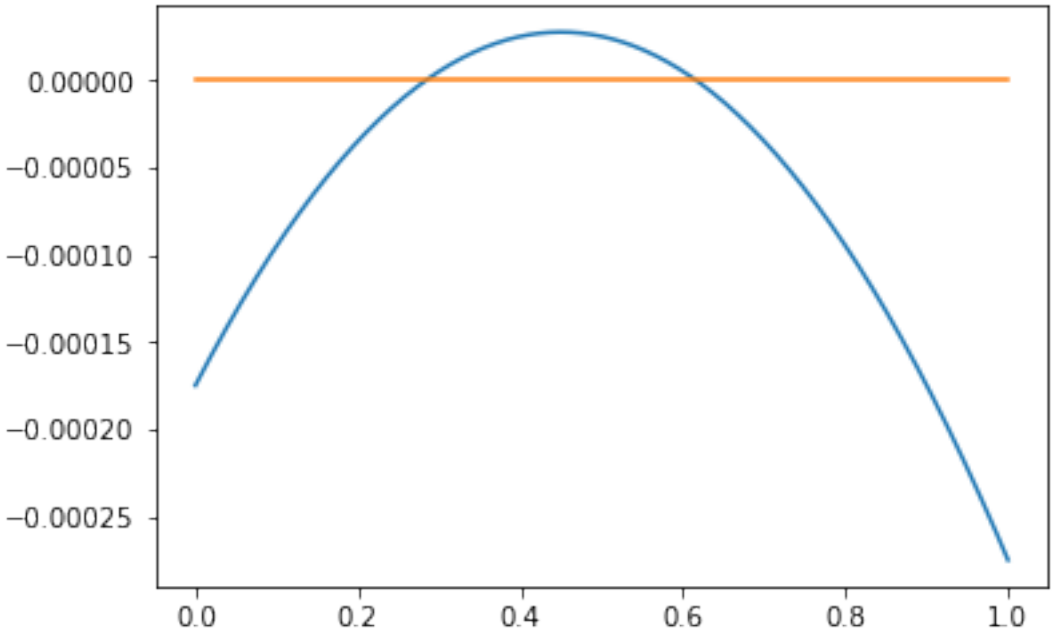

```
In [68]: newton(f3,0,100)

Out[68]: 0.28416876046073369

In [ ]: On obtient donc la valeur du point de fonctionnement, en accord avec le gra

In [69]: newton(f3,5,100)

Out[69]: 0.61583123956454922
```

Curieusement, si l'on choisit d'autres valeurs initiales, on trouve une autre valeur.

```
In [70]: newton(f3,8,100)

Out[70]: 0.61583123956454922
```

Il s'agit bien d'un zéro comme le suggère la courbe représentative de la fonction zoomée sur $[0, 1]$

```
In [74]: X=linspace(0,1,1000)
         plot(X,f3(X))
         plot(X,[0 for k in range(0,1000)])

Out[74]: [<matplotlib.lines.Line2D at 0x7f4399d9d940>]
```



Néanmoins, au vu de la figure fournie, on estime le point de fonctionnement pour une valeur inférieure à 0.5.