TP5correction

February 23, 2021

1 TP5 - Equations non linéaires

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

1.1 Exercice 1 : Ordre de convergence des méthodes numériques

[2]:
```
def f(x):
    return np.cosh(x-1) - np.sqrt(1 + x**2)
def fprime(x):
    return np.sinh(x-1) - x / np.sqrt(1 + x**2)
```

[3]:
```
print(f(0), f(1), f(2), f(3))
```

```
0.5430806348152437 -0.41421356237309515 -0.6929873426845461 0.5999180309152519
```

1.1.1 Question a.

Première analyse : f est décroissante, puis croissante, donc d'après les valeurs ci-dessus admet deux racines, l'une dans [0,1] et l'autre dans [2,3].

1.1.2 Question b.

Méthode de la corde, on part de $x_0 \in [a, b]$ et

$$x_{k+1} = x_k - \frac{b-a}{f(b)-f(a)} f(x_k)$$

[4]:
```
def corde(x, a, b, n, f):
    list = []
    fact = (b - a) / (f(b) - f(a))
    for i in range(n):
        x = x - fact * f(x)
        list.append(x)
    return x, list
```

[5]:
```
x1, list1 = corde(0.5,0,1,10,f)
print('première racine : ',x1,', résidu : ',f(x1))
```

```
x2, list2 = corde(2.5,2,3,10,f)
print('deuxième racine : ',x2,', résidu : ',f(x2))
```

```
première racine :  0.5099269315194523 , résidu :  0.0
deuxième racine :  2.7291102308392508 , résidu :  -1.1935640630511557e-05
```

Méthode de la sécante, on part de x_{-1} et $x_0 \in [a, b]$ et

$$x_{k+1} = x_k - \frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})} f(x_k)$$

[6]:
```
def secante(xm1, x, n, f, fprime):
    list = []
    for i in range(n):
        xinter = x
        if np.abs(f(x) - f(xm1)) <= 1.e-16:
            taux = fprime(x)
        else:
            taux = (f(x) - f(xm1)) / (x - xm1)
        x = x - f(x) / taux
        xm1 = xinter
        list.append(x)
    return x, list
```

[7]:
```
x3, list3 = secante(0.1,0.7,10,f,fprime)
print('première racine : ',x3,', résidu : ',f(x3))
x4, list4 = secante(2.1,2.9,10,f,fprime)
print('deuxième racine : ',x4,', résidu : ',f(x4))
```

```
première racine :  0.5099269315194522 , résidu :  0.0
deuxième racine :  2.7291168982143748 , résidu :  -4.440892098500626e-16
```

[8]:
```
list3
```

[8]:
```
[0.5256754030507611,
 0.5092065751370912,
 0.5099293941851798,
 0.5099269319015381,
 0.509926931519452,
 0.5099269315194522,
 0.5099269315194522,
 0.5099269315194522,
 0.5099269315194522,
 0.5099269315194522]
```

Méthode de Newton

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

[9]:
```python
def newton(x, n, f, fprime):
    list = []
    for i in range(n):
        x = x - f(x) / fprime(x)
        list.append(x)
    return x, list
```

[10]:
```python
x5, list5 = newton(0.5, 10, f, fprime)
print('première racine : ',x5,', résidu : ',f(x5))
x6, list6 = newton(2.5, 10, f, fprime)
print('deuxième racine : ',x6,', résidu : ',f(x6))
```

```
première racine :  0.5099269315194522 , résidu :  0.0
deuxième racine :  2.729116898214375 , résidu :  8.881784197001252e-16
```

1.1.3 Question c.

Dans la méthode de la sécante, on a un soucis de division par zéro. En effet, pour une certaine itération k, on a $f(x_k) = f(x_{k-1})$ à erreur machine près. Dans le programme de la sécante, on a donc triché pour écarter ce cas. On a remplacé $\dfrac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}$ par $f'(x_k)$ dans le cas où $f(x_k)$ et $f(x_{k-1})$ sont trop proches.

1.1.4 Question d.

[11]:
```python
print('Première racine')
print('Solutions successives, méthode de la corde')
for x in list1:
    print(x)
print('Solutions successives, méthode de la sécante')
for x in list3:
    print(x)
print('Solutions successives, méthode de Newton')
for x in list5:
    print(x)
```

```
Première racine
Solutions successives, méthode de la corde
0.5100198836310282
0.5099262627177411
0.5099269363451499
0.5099269314846333
0.5099269315197035
0.5099269315194505
0.5099269315194523
0.5099269315194523
0.5099269315194523
0.5099269315194523
```

```
Solutions successives, méthode de la sécante
0.5256754030507611
0.5092065751370912
0.5099293941851798
0.5099269319015381
0.509926931519452
0.5099269315194522
0.5099269315194522
0.5099269315194522
0.5099269315194522
0.5099269315194522
Solutions successives, méthode de Newton
0.5099059054880547
0.5099269314242016
0.5099269315194522
0.5099269315194522
0.5099269315194522
0.5099269315194522
0.5099269315194522
0.5099269315194522
0.5099269315194522
0.5099269315194522
```

[12]:

```python
print('Deuxième racine')
print('Solutions successives, méthode de la corde')
for x in list2:
    print(x)
print('Solutions successives, méthode de la sécante')
for x in list4:
    print(x)
print('Solutions successives, méthode de Newton')
for x in list6:
    print(x)
```

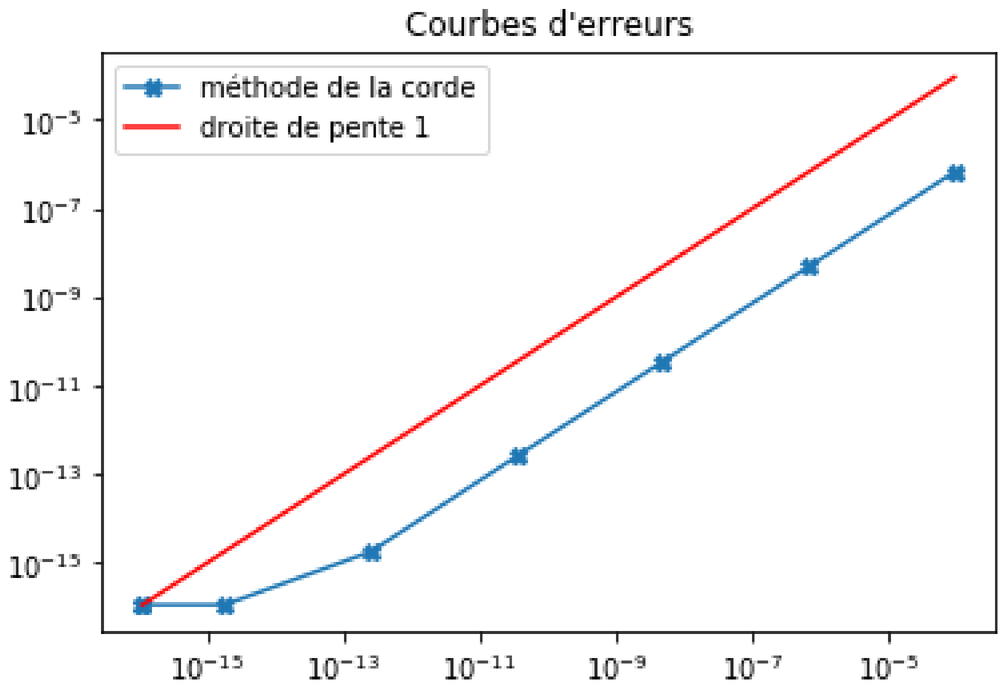
```
Deuxième racine
Solutions successives, méthode de la corde
2.7631072584816256
2.714749380669009
2.734415016196211
2.727048043712181
2.7299078529600838
2.728811998204833
2.72923406177224
2.729071821113478
2.729134232916333
2.7291102308392508
Solutions successives, méthode de la sécante
2.6219803941407096
```

```
2.714941520688669
2.7304073786057446
2.7291021650421885
2.729116883005035
2.729116898214554
2.7291168982143748
2.729116898214375
2.7291168982143748
2.7291168982143748
Solutions successives, méthode de Newton
2.783287812505934
2.73135059276508
2.7291208833971052
2.7291168982270873
2.7291168982143748
2.729116898214375
2.7291168982143748
2.729116898214375
2.7291168982143748
2.729116898214375
```

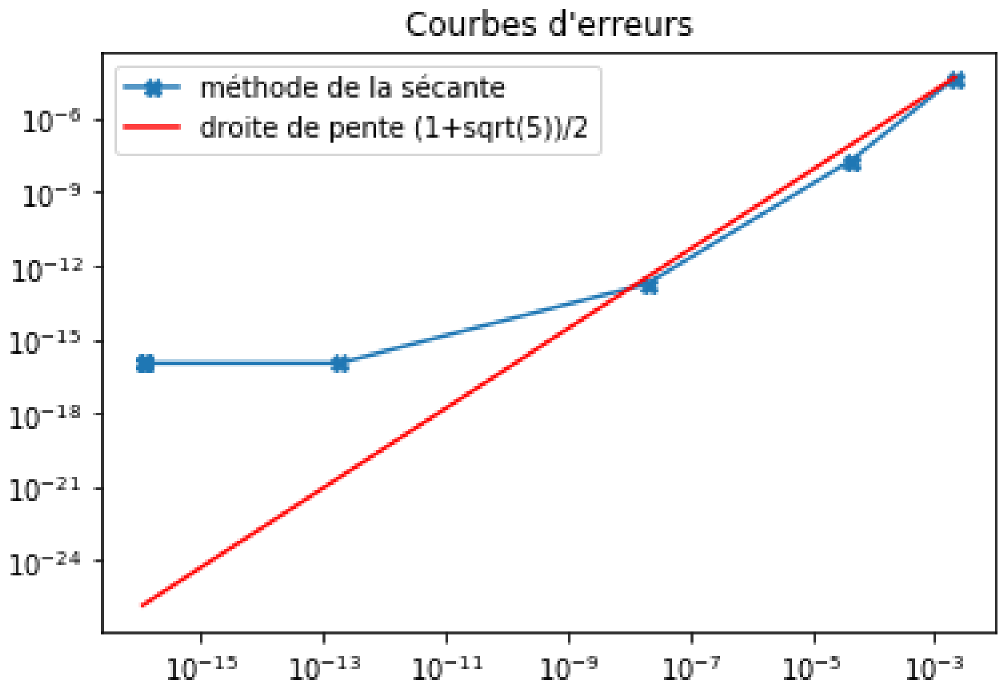
1.1.5 Question e.

[13]:

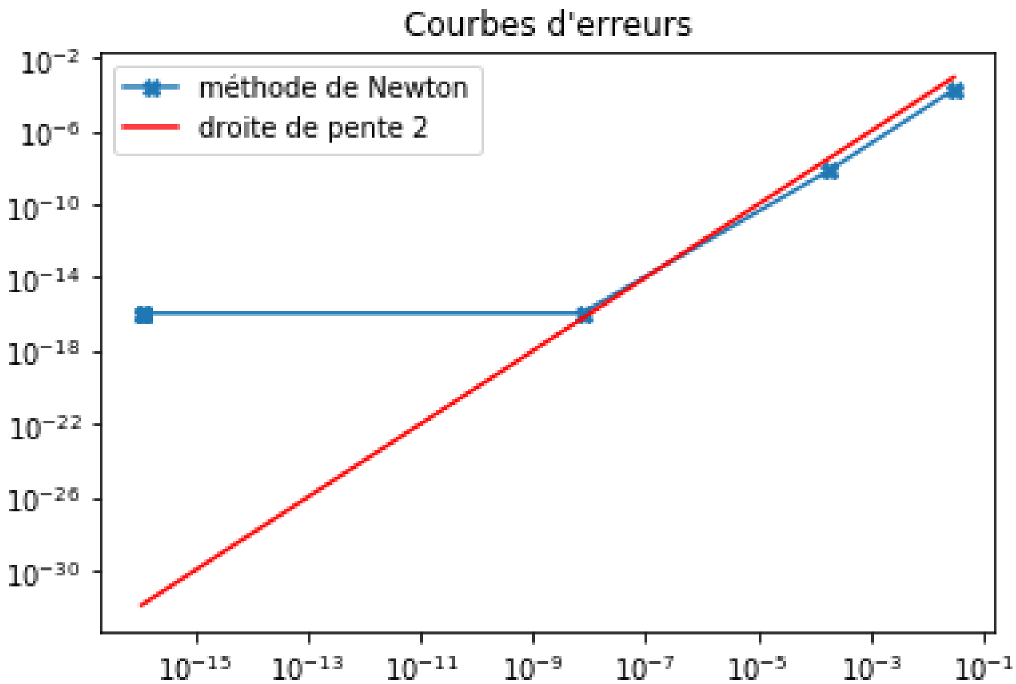
```python
x, list = corde(0.5,0,1,10,f)
xref, list2 = newton(0.5,30,f,fprime)
erreurs = np.abs(np.array(list) - xref)
plt.loglog(erreurs[:-1],erreurs[1:],'X-',label = 'méthode de la corde')
plt.loglog(erreurs[:-1],erreurs[:-1],'r',label = 'droite de pente 1')
plt.legend(loc = "best")
plt.title("Courbes d'erreurs")
plt.show()
```



[14]:
```
x, list = secante(0.4,0.6,10,f,fprime)
xref, list2 = newton(0.5,30,f,fprime)
erreurs = np.abs(np.array(list) - xref)
plt.loglog(erreurs[:-1],erreurs[1:],'X-',label = 'méthode de la sécante')
taux = (1 + np.sqrt(5)) / 2
plt.loglog(erreurs[:-1],erreurs[:-1]**taux,'r',label = 'droite de pente␣
 ↪(1+sqrt(5))/2')
plt.legend(loc = "best")
plt.title("Courbes d'erreurs")
plt.show()
```



[15]:
```
x, list = newton(0.1,10,f,fprime)
xref, list2 = newton(0.5,30,f,fprime)
erreurs = np.abs(np.array(list) - xref)
plt.loglog(erreurs[:-1],erreurs[1:],'X-',label = 'méthode de Newton')
plt.loglog(erreurs[:-1],erreurs[:-1]**2,'r',label = 'droite de pente 2')
plt.legend(loc = "best")
plt.title("Courbes d'erreurs")
plt.show()
```



1.2 Exercice 2 : Systèmes d'équations non linéaires

1.2.1 Question a.

$$F(x,y) = (x^2 + 4y^2 - 4, \exp(y + y^2) - 2 - \sin(x))$$

$$dF(x,y) : (h,v) \mapsto (2xh + 8yv, -\cos(x)h + (1+2y)\exp(y+y^2)v)$$

1.2.2 Question b.

```
[16]: def F(x):#x est un vecteur de taille 2 contenant les variables x et y de␣
       ↪l'énoncé
          return np.array([x[0]**2 + 4*x[1]**2 - 4, np.exp(x[1] + x[1]**2) - 2 - np.
       ↪sin(x[0])])
      def DF(x):
          return np.array([[2*x[0],       8*x[1]                                  ],
                           [-np.cos(x[0]), (1+2*x[1]) * np.exp(x[1] + x[1]**2)]])
```

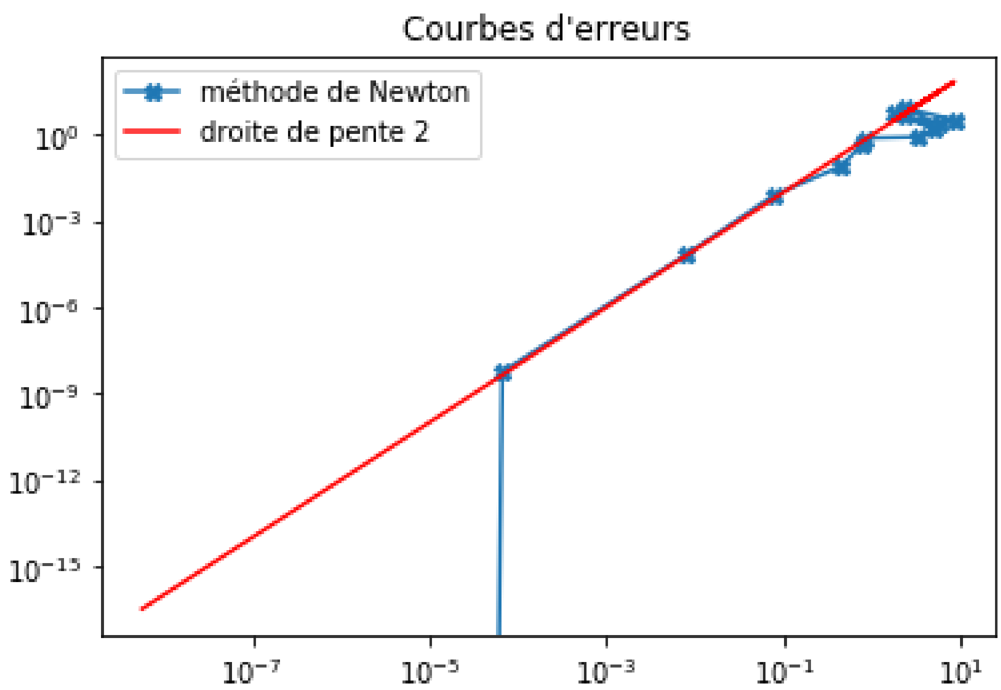
1.2.3 Question c.

```
[17]: def newton2D(x, F, DF, n):
          list = []
          for i in range(n):
              x = x - np.linalg.solve(DF(x), F(x))
              list.append(x)
```

```
    return x, list
```

[18]:
```
x = np.array([1,0])
y, list = newton2D(x,F,DF,15)
erreurs = np.zeros(len(list))
xref = list[-1]
for i in range(len(list)):
    x = list[i]
    print(x, np.linalg.norm(F(x)))
    erreurs[i] = np.linalg.norm(x - xref)
plt.loglog(erreurs[:-1],erreurs[1:],'X-',label = 'méthode de Newton')
plt.loglog(erreurs[:-1],erreurs[:-1]**2,'r',label = 'droite de pente 2')
plt.legend(loc = "best")
plt.title("Courbes d'erreurs")
plt.show()
```

```
[2.5        2.65192444] 16066.11641513928
[-2.90335938  2.49335959] 6062.7718527224915
[1.56794231 2.3262521 ] 2290.282365258251
[-3.79455552  2.14956962] 869.3752060377693
[-0.41658306  1.96084337] 330.81072664777423
[9.74994064 1.764323  ] 165.84626631020672
[4.59340242 1.55455402] 58.52046056199047
[2.00116723 1.3172239 ] 19.531656206751318
[0.87471222 1.08600507] 7.025975496868753
[1.11778412 0.86639451] 2.153785159939993
[1.45363315 0.72171265] 0.5107276253274882
[1.49547745 0.66660442] 0.04246466141710295
[1.50082491 0.66099903] 0.0003981886671982314
[1.50086569 0.6609467 ] 3.3618260810076556e-08
[1.5008657 0.6609467] 8.881784197001252e-16
```



1.3 Exercice 3 : Méthode d'homotopie

1.3.1 Question a.

[19]:
```
def fmu(x, mu):
    return x * np.exp(x) - mu
def fmuprime(x, mu):
    return (1 + x) * np.exp(x)
def newton2(x, n, mu):
    list = []
    for i in range(n):
        x = x - fmu(x,mu) / fmuprime(x,mu)
        list.append(x)
    return x, list
```

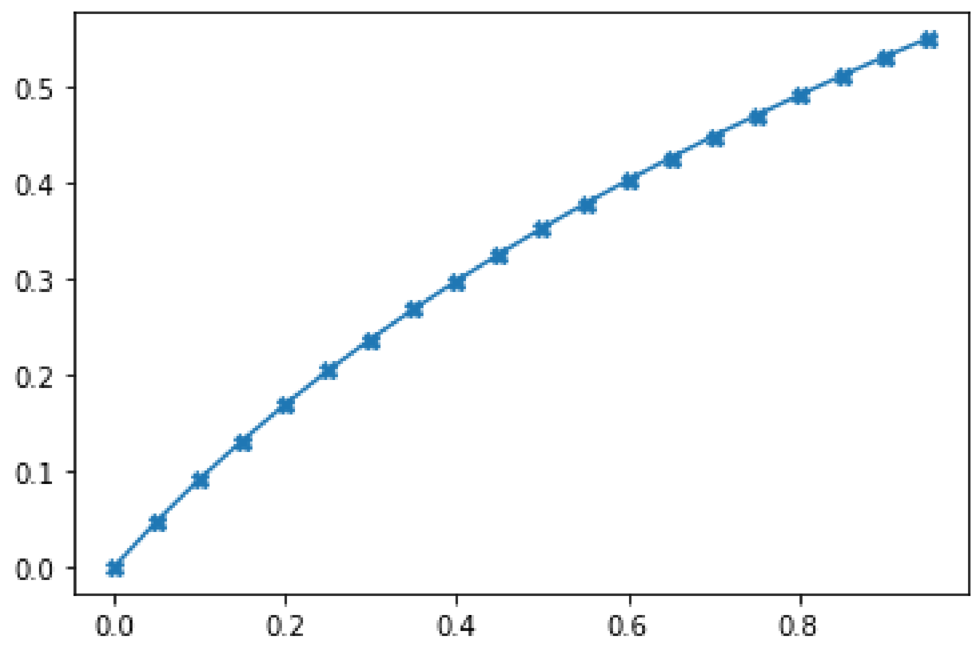
1.3.2 Question b.

[20]:
```
h = 0.05
niter = int(1/h)
mu = 0
tabx = [0]
tabmu = [0]
tabf = [fmu(tabx[0], tabmu[0])]
for i in range(1,niter):
```

```
    mu += h
    x, list = newton2(tabx[-1],15,mu)
    tabmu.append(mu)
    tabx.append(x)
    tabf.append(fmu(x,mu))
print('mu positif')
print(tabx)
print(tabf)
plt.plot(tabmu,tabx,'X-')
```

```
mu positif
[0, 0.04767230860012938, 0.09127652716086226, 0.13151492800103448,
0.16891597349910956, 0.20388835470224018, 0.23675531078855933,
0.2677773400403608, 0.2971677506731385, 0.32510362011804045, 0.3517337112491958,
0.3771843139172231, 0.4015636367870726, 0.4249651641143922, 0.44747025926965506,
0.4691502106949883, 0.49006785880157994, 0.5102789035462333, 0.5298329656334345,
0.5487744554528027]
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 5.551115123125783e-17, 0.0,
-5.551115123125783e-17, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
-1.1102230246251565e-16, 0.0, 0.0]
```

[20]: [<matplotlib.lines.Line2D at 0x7f8937fc4e50>]

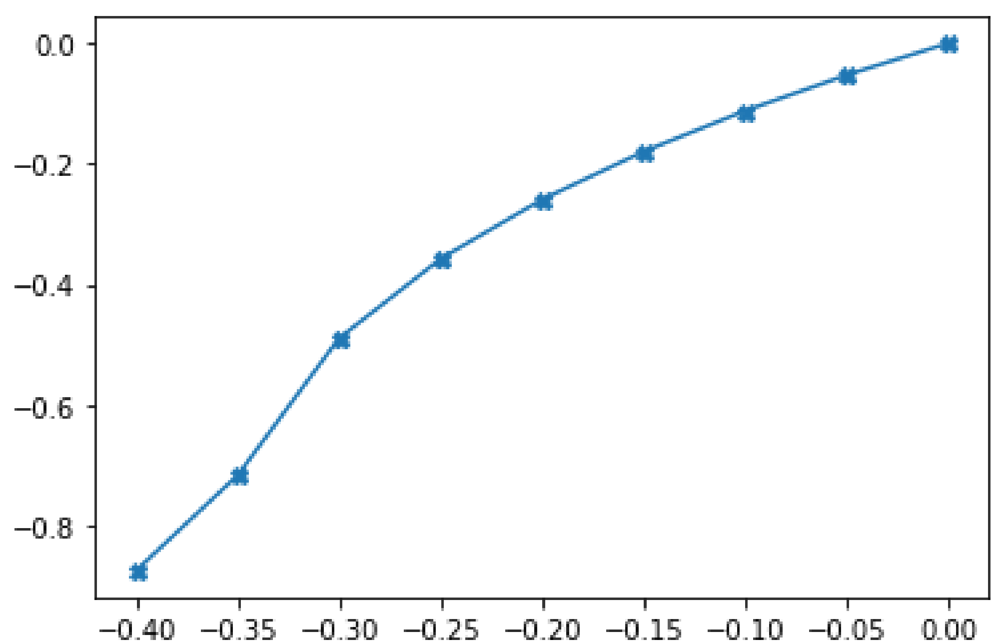


1.3.3 Question c.

[21]:
```
mu = 0
tabx2 = [0]
tabmu2 = [0]
tabf2 = [fmu(tabx2[0],tabmu2[0])]
for i in range(1,niter):
    mu -= h
    x,list = newton2(tabx2[-1],15,mu)
    tabmu2.append(mu)
    tabx2.append(x)
    tabf2.append(fmu(x,mu))
print('mu négatif')
print(tabx2)
print(tabf2)
plt.plot(tabmu2,tabx2,'X-')
```

```
mu négatif
[0, -0.05270598355154635, -0.11183255915896297, -0.17949126834798476,
-0.25917110181907377, -0.3574029561813889, -0.489402227180215,
-0.7166388164560739, -0.8746941295604564, nan, nan, nan, nan, nan, nan, nan,
nan, nan, nan, nan]
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.03526169294431719, nan, nan, nan,
nan, nan, nan, nan, nan, nan, nan, nan]
```

[21]:
```
[<matplotlib.lines.Line2D at 0x7f893807ad50>]
```



Note : pour des valeurs trop négatives de μ, l'algorithme ne converge pas (même en diminuant h)