3. Discrétisation

Tache 3 :

temp.py tp2.py tp2 EXE .py exe.py tp2 3.py TP 3.py TP4.py TP5.py

```python
from math import *
import numpy as np
import matplotlib.pyplot as plt

def jh(l,h,y,penalisation ,n):    #calcul de l'expression de la fonctionel Jh de probleme discrétisé
    s=0.
    o=0.
    for i in range(0,n-1):
        r=y[i+1]-y[i]
        s=s+y[i+1]*sqrt(h*h+r*r)
    for j in range(0,n-1):
        o=o+sqrt(h*h+pow((y[i+1]-y[i]),2))
    return (s+1./penalisation*(o-l)*(o-l))
```

4.2 Méthode de la section dorée

Tache 4 :

```python
def golden2(y0, a, c, b):         #la methode de golden section search
    phi = (1 + sqrt(5))/2
    resphi = 2 - phi
    if abs(a - b) < 0.0001:
        return (a + b)/2
    d = c + resphi*(b - c)
    z1=jh(l,h,y0-d*djh(y0,h,penalisation,n),penalisation,n)
    z2=jh(l,h,y0-c*djh(y0,h,penalisation,n),penalisation,n)
    if (z1 < z2):
        return golden2(y0, c, d, b)
    else:
        return golden2(y0, d, c, a)
```

4.3 Application et implémentation

Tache 6 :

```
def djh(y,h,penalisation,n,l):                    # calcul de gradient de Jh en y

    m=np.zeros(n)
    a=0.
    r=0.
    s=0.
    for i in range(0,n-1):
        a=a+sqrt(h*h+pow((y[i+1]-y[i]),2))
    m[0]=0.
    m[n-1]=0.
    for j in range(1,n-1):
        s=sqrt(h*h+pow((y[j]-y[j-1]),2))
        r=(y[j]*(y[j]-y[j-1]))/s
        q=sqrt(h*h+pow((y[j+1]-y[j]),2))
        K=y[j+1]*(y[j+1]-y[j])/q
        m[j]=s+r-K+((2./penalisation)*(a-l)*((y[j]-y[j-1])/s-(y[j+1]-y[j])/q))
    return m
```

Tache 7 :

```
def  normevect(x,n): #calcul de la norme  2 de vecteur x
        s=0.
        z=0.
        for i in range(0,n):
            s=s+pow(x[i],2)
        z = sqrt(s)
        return z
```

```
def pasfixe(y0,eps,n,nbmax,pas):        #determination de la solution du probleme avec la methode de gradient a pas fixe
        k=1
        y1=np.zeros(n)
        y1=y0-pas*djh(y0,h,penalisation,n,l)
        diff=normevect(y1-y0,2)
        while((diff>=eps)and(k<=nbmax)):
            y0=y1
            y1=y0-pas*djh(y0,h,penalisation,n,l)
            k=k+1
            diff=normevect(y1-y0,2)
            if ((jh(l,h,y1,penalisation,n)>=jh(l,h,y0,penalisation,n))):
                break
        return y1
```

```python
def pasvar(y0,eps,penalisation,n,nbmax): #determination de la solution du probleme en utilisant la methode du pas variable
    k=1
    y1=np.zeros(n)
    f=0.01
    e=0.009
    alpha=golden2(y0,f,(f+e)/2.,e)
    y1=y0-alpha*djh(y0,h,penalisation,n,l)
    diff=normevect(y1-y0,2)
    while((diff>=eps)and(k<=nbmax)):
            y0=y1
            k=k+1
            alpha=golden2(y0,f,(f+e)/2.,e)
            w=djh(y0,h,penalisation,n,l)
            y1=y0-alpha*w
            diff=normevect(y1-y0,2)
            if ((jh(l,h,y1,penalisation,n)>=jh(l,h,y0,penalisation,n))):
                break

    return (y1)
```

5. Etude numérique

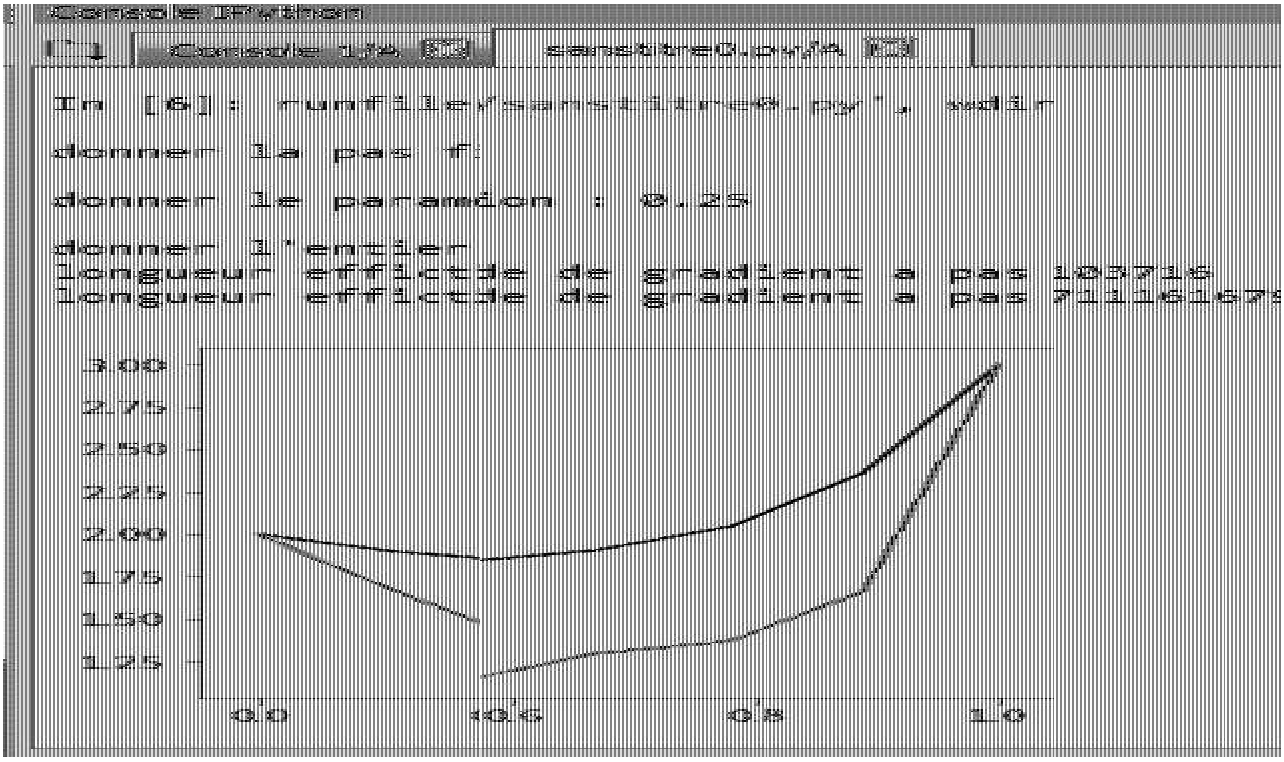
Tache 8:

```python

def longueureffective(y,h): #calcul de longueur effective
    k=0.
    v=0.
    for j in range(0,n-1):

        k=k+sqrt(h*h+(y[j+1]-y[j])*(y[j+1]-y[j]))


    return (k)
```

```python
xa=0.      #L'abscisse de la premiere point d'attache a
xb=1.      #L'abscisse de la deuxieme point d'attache b
ya=2.      #L'ordonné de la premiere point d'attache a
yb=3.      #L'oronné de la deuxiéme point d'attache b
l=2.       #La longueur du fil

while(1==1):   #la controle de saisie de la pas fixe
    pasf=float(input("donner la pas fixe : "))
    if(pasf<0.1):
        break

penalisation=float(input("donner le parametre de penalisation : ")) #la saaisie de paramétre de penilisation
n=int(input("donner l'entier n : "))

h=(xb-xa)/n
y00=np.zeros(n)  #le vecteur initial y0
y00[0]=ya
y00[n-1]=yb

nbmax=200        #nombre maximum d'iteration
eps=0.0001       # la tolerance

y1=pasfixe(y00,eps,n,nbmax,pasf)  #Le resultat obtenue par la methode du pas fixe

y2=pasvar(y00,eps,penalisation,n,nbmax)   # le resultat  obtenue par la methode de pas variable

f=longueureffective(y1,h)
print("longueur effictive pour la methode de gradient a pas fixe = ",f)
f=longueureffective(y2,h)
print("longueur effictive pour la methode de gradient a pas variable = ",f)

t=np.linspace(xa,xb,n)
plt.plot(t,y1,'g')     # l graphe pour la methode de gradient a pas fixe
plt.plot(t,y2,'r')    #Le graphe pour la methode de gradient a pas variable
plt.show()
```

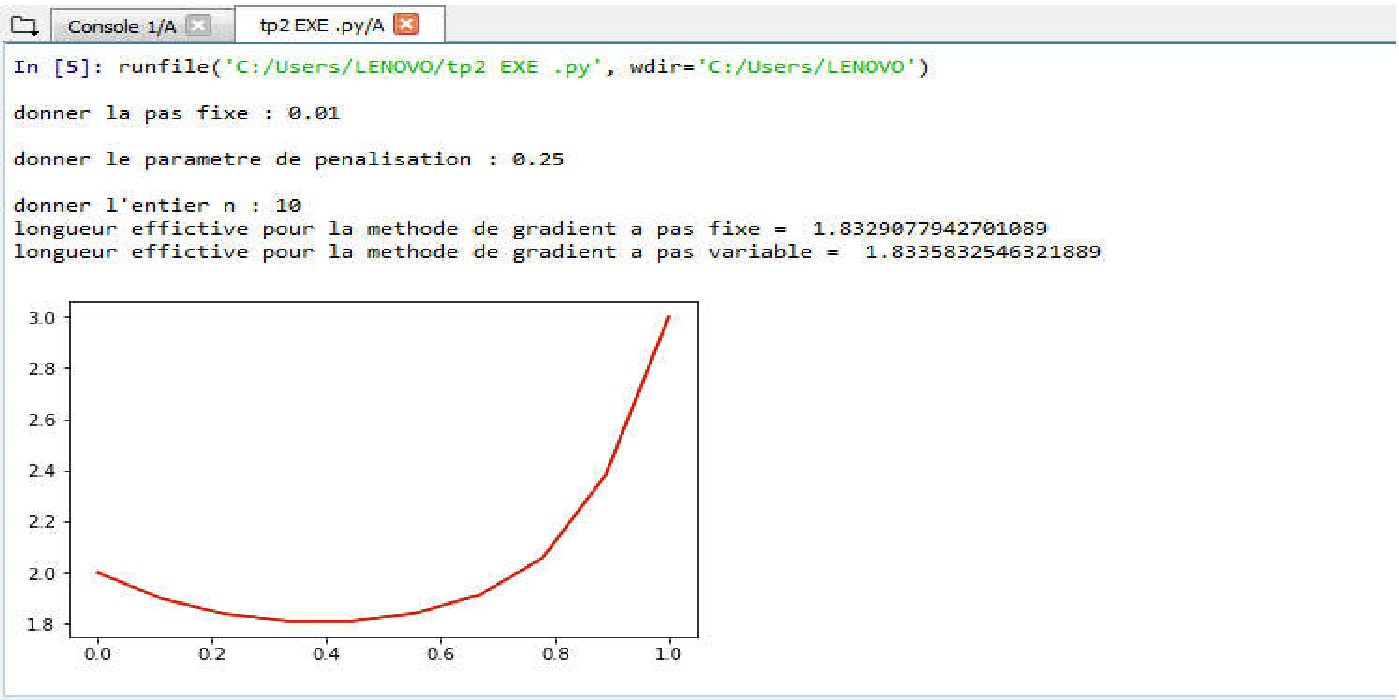
Essai num 1 :

a=(0 ,2) ; b=(1,3) ; l=2 ; parametre de penalisation=0.25 ; la tolerence pour le critère d'arret =0.0001 et le nb max d'itération=200



Essai num 2 :

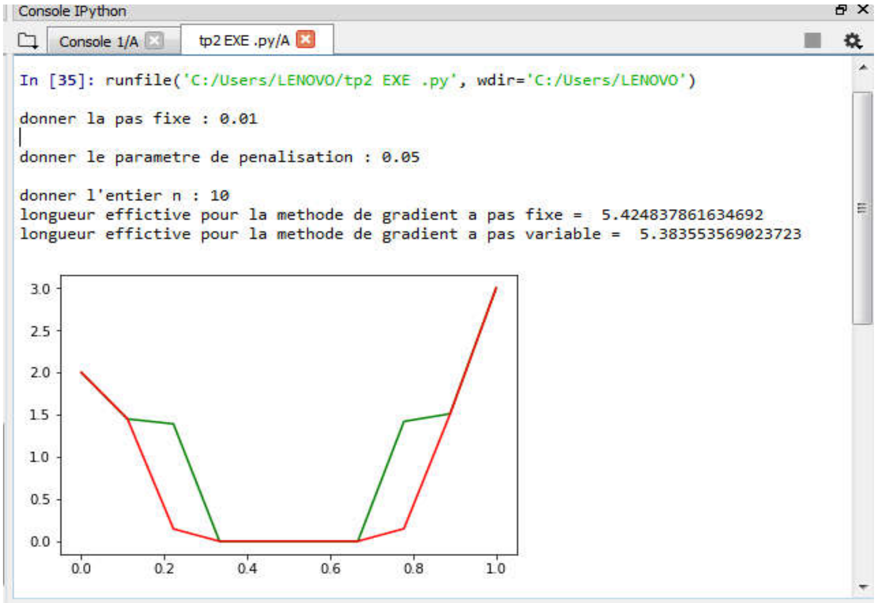
a=(0 ,2) ; b=(1,3) ; l=2 ; parametre de penalisation=0.25 ; la tolerence pour le critere d'arret =0.000001 et le nb max d'itération=200



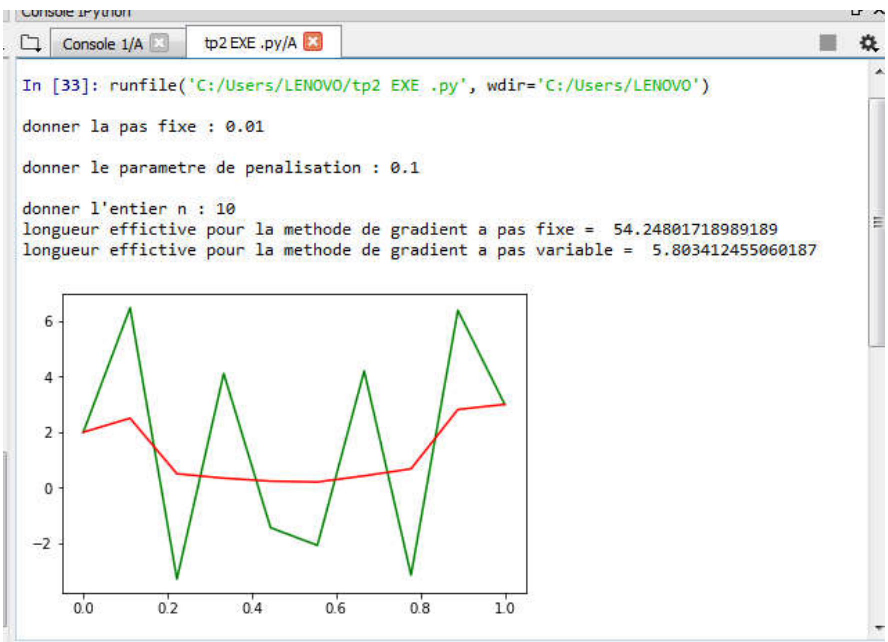
**on remarque que pour une valeur de epsilon = 0.0001, la méthode de gradient à pas fixe nous a donner une valeur de longueur effective plus prés à la valeur de la longueur du fil choisie que la méthode de gradient à pas variable . Et lorsque on a diminué la valeur de epsilon =0.0000001, on a obtenue presque la même valeur avec les 2 méthodes. En conclusion, la méthode de gradient à pas fixe est plus précise que celle à pas variable.

Tache9 :

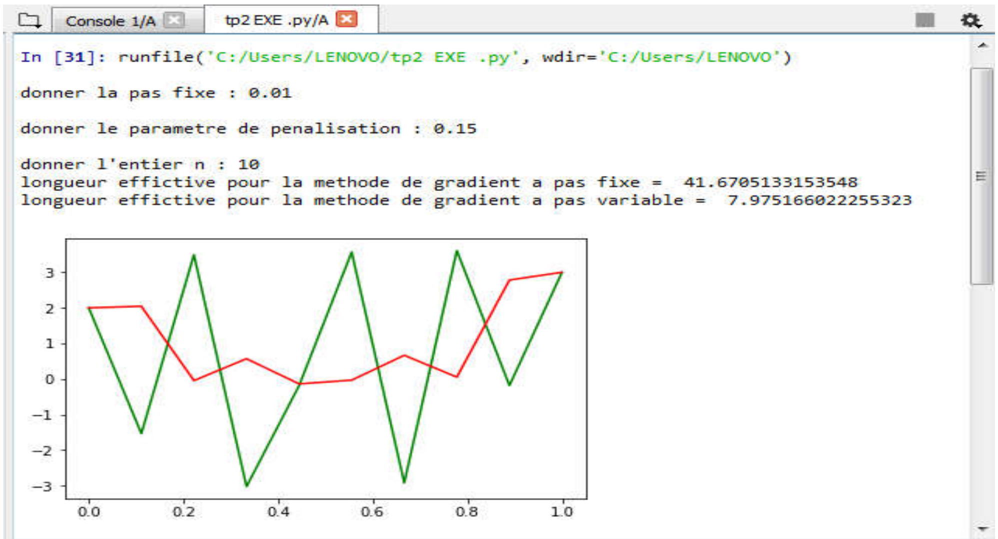
Variation de la valeur de paramètre de pénalisation :



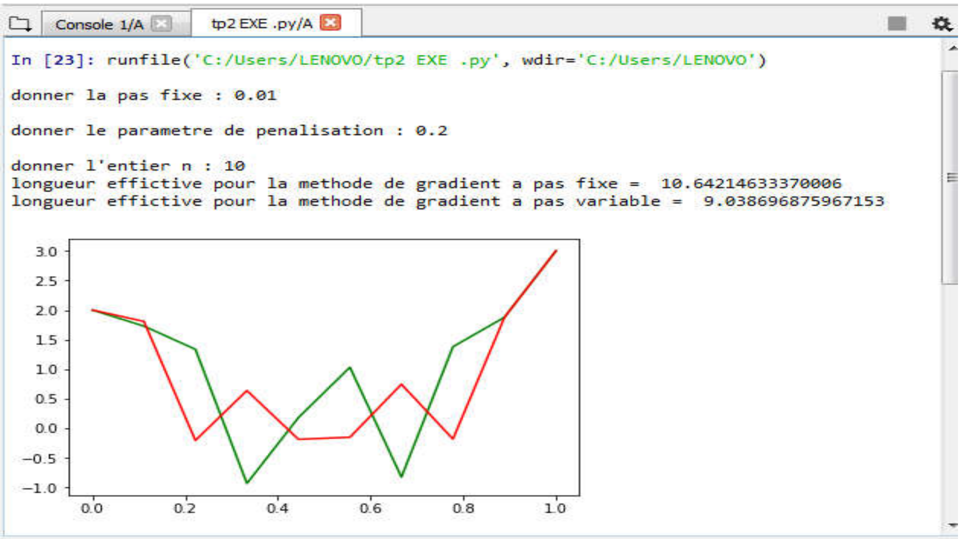
Pour eps=0.05



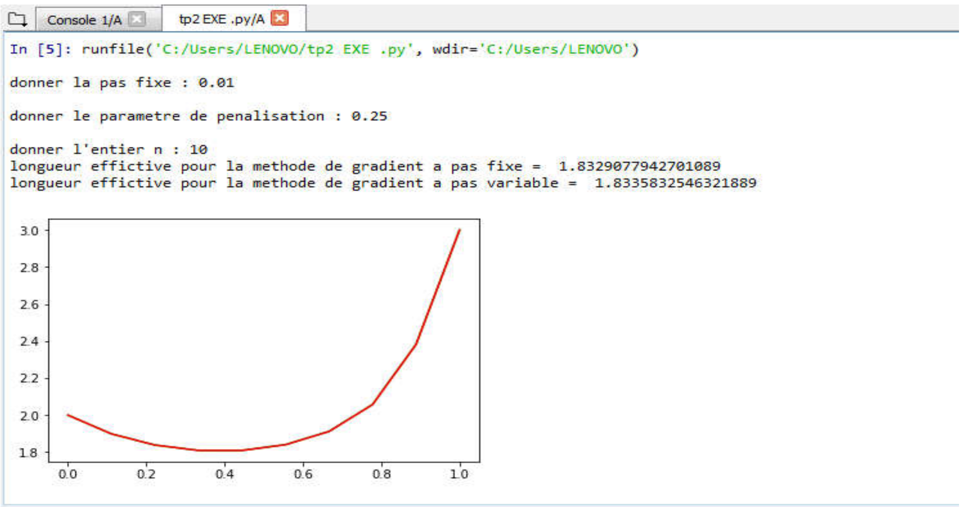
Pour eps=0.1



Pour eps=0.15



Pour eps=0.2

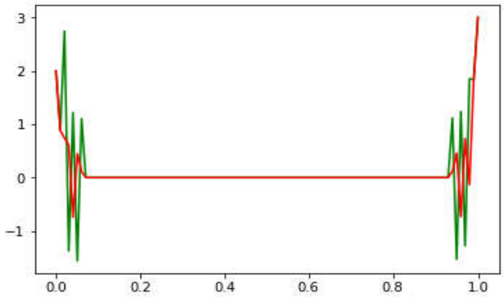


Pour eps=0.25

Pour des valeurs d'eps très inferieure à 0.25 les 2 méthodes nous donnent des valeurs de longueur effective très supérieure à 2. Mais pour des valeurs pas très différentes de 0.25 on obtient presque la meme valeur de longueur effective

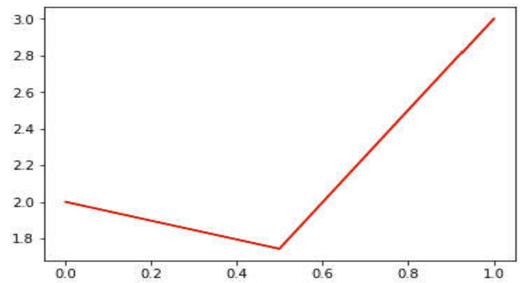
Variation de la valeur de N de points de discrétisation :

```
In [21]: runfile('C:/Users/LENOVO/tp2 EXE .py', wdir='C:/Users/LENOVO')

donner la pas fixe : 0.01

donner le parametre de penalisation : 0.25

donner l'entier n : 100
longueur effictive pour la methode de gradient a pas fixe =  30.47722071217617
longueur effictive pour la methode de gradient a pas variable =  12.33933098277307
```



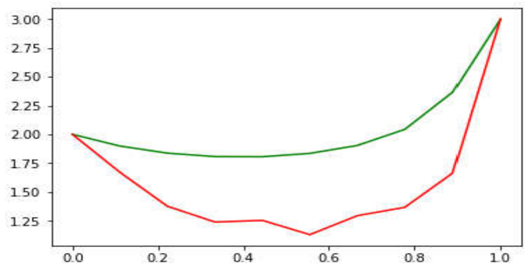
N=100

```
In [20]: runfile('C:/Users/LENOVO/sanstitre0.py', wdir='C:/Users/LENOVO')

donner la pas fixe : 0.01

donner le parametre de penalisation : 0.25

donner l'entier n : 3
longueur effictive pour la methode de gradient a pas fixe =  1.7196199488187591
longueur effictive pour la methode de gradient a pas variable =  1.7196402678159228
```



N=3

```
In [6]: runfile('C:/Users/LENOVO/sanstitre0.py', wdir='C:/Users/LENOVO')

donner la pas fixe : 0.01

donner le parametre de penalisation : 0.25

donner l'entier n : 10
longueur effictive pour la methode de gradient a pas fixe =  1.8410279864103716
longueur effictive pour la methode de gradient a pas variable =  3.0531917111616793
```



N=10

Variation de la valeur de paramètre de pénalisation :