Projet d'Analyse Numérique:

RÉSOUDRE LE PROBLÈME DE LA CHAÎNETTE

Travail présenté par :

Mohamed Aziz Tousli

Hakim Jemaa

Riadh Cherni

Année universitaire :

2017 _ 2018

I. **But du projet:**

Considérons un câble homogène, flexible, attaché en deux points A et B. Dans sa position d'équilibre, le câble pend dans un plan vertical et semble prendre une forme parabolique. La chaînette est le nom que porte cette forme.

Le But de ce projet est de déterminer numériquement la longueur de cette forme au moyen de diverses approximations.

II. Tâches à faire :

Tâche1 :

L’équation cartésienne de la forme de la chaînette est :

$$\boxed{y = \alpha ch\left(\frac{x-\beta}{\alpha}\right) + \Upsilon}$$

On déterminera à partir des conditions aux limites et de la contrainte sur la longueur du fil une équation transcendante vérifiée par le coefficient α, et à partir de laquelle on déterminera les autres coefficients β et ϒ.

Conditions aux limites:

- $y_a = \alpha ch\left(\frac{x_a-\beta}{\alpha}\right) + \Upsilon$

$$= \frac{\alpha}{2}\left(e^{\frac{x_a-\beta}{\alpha}} + e^{\frac{-x_a+\beta}{\alpha}}\right) + \Upsilon$$

- $y_b = \alpha ch\left(\frac{x_b-\beta}{\alpha}\right) + \Upsilon$

$$= \frac{\alpha}{2}\left(e^{\frac{x_b-\beta}{\alpha}} + e^{\frac{-x_b+\beta}{\alpha}}\right) + \Upsilon$$

La contrainte sur la longueur du fil:

- $L = \int_{x_a}^{x_b} \sqrt{1 + \mathrm{sh}^2\left(\frac{x-\beta}{\alpha}\right)}\, dx$

$$= \int_{x_a}^{x_b} ch\left(\frac{x-\beta}{\alpha}\right) dx$$

$$= \alpha\left[sh\left(\frac{x_b-\beta}{\alpha}\right) - sh\left(\frac{x_a-\beta}{\alpha}\right)\right]$$

$$= \frac{\alpha}{2}\left(e^{\frac{x_b-\beta}{\alpha}} - e^{\frac{-x_b+\beta}{\alpha}} - e^{\frac{x_a-\beta}{\alpha}} + e^{\frac{-x_a+\beta}{\alpha}}\right)$$

$$= \frac{\alpha}{2}\left(e^{-\frac{\beta}{\alpha}}\left(e^{\frac{x_b}{\alpha}} - e^{\frac{x_a}{\alpha}}\right) + e^{\frac{\beta}{\alpha}}\left(e^{\frac{-x_a}{\alpha}} - e^{\frac{-x_b}{\alpha}}\right)\right)$$

Donc :

$$y_b - y_a - L = \alpha\left(e^{\frac{\beta-x_b}{\alpha}} - e^{\frac{\beta-x_a}{\alpha}}\right) = e^{\frac{\beta}{\alpha}}\left(e^{\frac{-x_b}{\alpha}} - e^{\frac{-x_a}{\alpha}}\right)$$

***D'o*ù :**

$$e^{\frac{\beta}{\alpha}} = \frac{y_b - y_a - L}{e^{\frac{-x_b}{\alpha}} - e^{\frac{-x_a}{\alpha}}}$$

En remplaçant la valeur de $e^{\frac{\beta}{\alpha}}$ ***dans L , on obtient*** :

$$L = \frac{\alpha}{2}\left(\frac{y_b - y_a - L}{e^{\frac{-x_b}{\alpha}} - e^{\frac{-x_a}{\alpha}}}\left(e^{\frac{-x_a}{\alpha}} - e^{\frac{-x_b}{\alpha}}\right) + \frac{y_b - y_a - L}{e^{\frac{-x_b}{\alpha}} - e^{\frac{-x_a}{\alpha}}}\left(e^{\frac{x_b}{\alpha}} - e^{\frac{x_a}{\alpha}}\right)\right)$$

$$= \frac{\alpha}{2}\left(L + y_a - y_b + \frac{y_b - y_a - L}{2 - e^{\frac{x_a - x_b}{\alpha}} - e^{\frac{x_b - x_a}{\alpha}}}\right)$$

$$= \frac{\alpha}{2}\left(L + y_a - y_b + \frac{y_b - 2y_a - L}{1 - ch\left(\frac{x_b - x_a}{\alpha}\right)}\right)$$

On obtient finalement l'équation , $f(\alpha) = 0$ avec :

$$\boxed{f(x) = L - \frac{x}{2}\left(L + y_a - y_b + \frac{y_b - 2y_a - L}{1 - ch\left(\frac{x_b - x_a}{x}\right)}\right) = 0}$$

Déterminant β et Υ:

On a obtenu : $e^{\frac{\beta}{\alpha}} = \frac{y_b - y_a - L}{e^{\frac{-x_b}{\alpha}} - e^{\frac{-x_a}{\alpha}}}$

D'où :

$$\beta = \alpha Ln(\quad -bx-bya-=Lx\ a$$

$$) \, \overline{y}$$

$$e\ \alpha - e\ \alpha$$

Et **:**

$$\Upsilon = y_a - \alpha ch \, (\, \underline{\quad\quad}$$

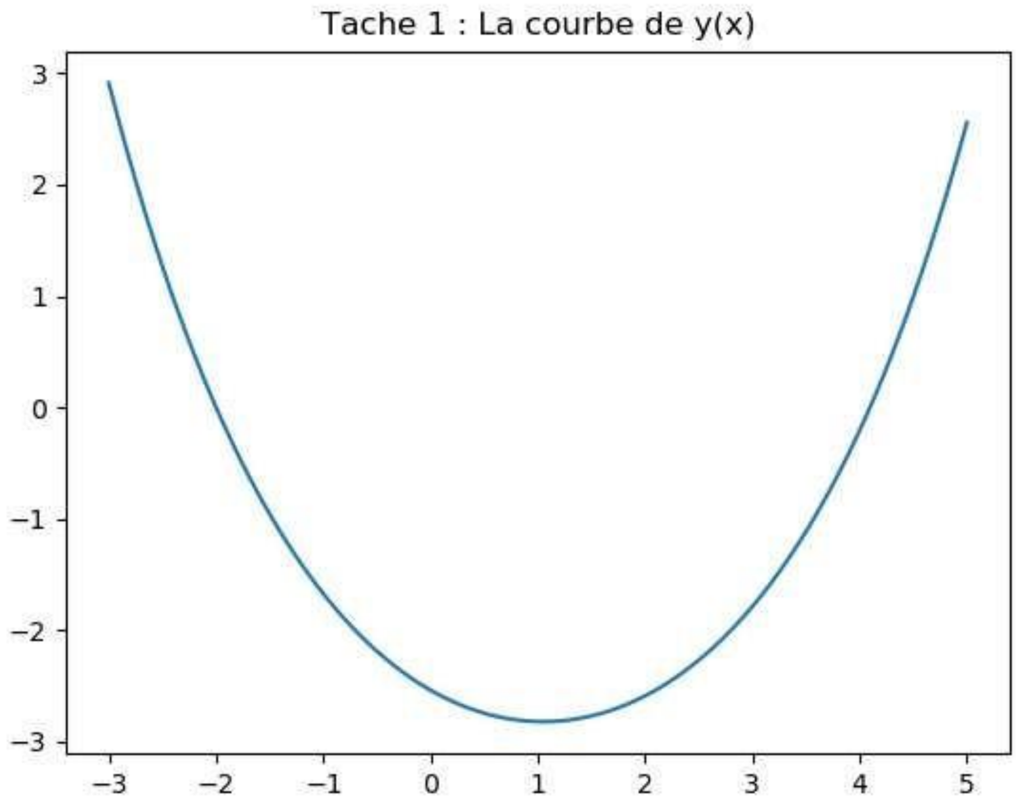
$$_{a}\alpha - \beta) \ x$$

Implémentation en python :

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


Ya=0
Yb=0
Xa=-2
Xb=2
L=4

#equation transcendante :
def f(x):
    return(L-x/2*(L+Ya-Yb+2/(Yb-Ya-1)*(1-cosh((Xb-Xa)/x))))

#calcul de alpha :
alpha=resol.fsolve(f,1000)
print("alpha=",alpha)

#calcul de beta :
beta=alpha*log((Yb-Ya-L)/(exp(-Xb/alpha)-exp(-Xa/alpha)))
print("beta=",beta)

#calcul de gama :
gama=Ya-alpha*cosh((Xa-beta)/alpha)
print("gama=",gama)

#fonction y :
def Y(x):
    return(alpha*cosh((x-beta)/alpha)+gama)

x=np.linspace(-3,5,1000)
y=[Y(x[i]) for i in range(0,np.size(x))]

plt.title("Tache 1 : La courbe de y(x)")
plt.plot(x,y)
plt.show()
```

Exécution :

```
alpha= [ 1.99201777]
beta= [ 1.04870201]
gama= [-4.81743046]
```



<u>Tache 2 :</u>

A partir de :

- la fonction : $y_h(x) = y_i + \frac{x - x_i}{h}(y_{i+1} - y_i)$

- La formule de quadrature du point milieu :

$$\int_{x_A}^{x_B} f(x)dx = h \sum_{j=1}^{N-1} \left(\frac{x_j + x_{j+1}}{2}\right)$$

- Le problème pénalisé :

$$\min \int_{x_a}^{x_b} y_\varepsilon(x)\sqrt{1 + y'_\varepsilon(x)^2}\,dx + \frac{1}{\varepsilon}\left(\int_{x_a}^{x_b} y_\varepsilon(x)\sqrt{1 + y'_\varepsilon(x)^2}\,dx - l\right)^2$$

On obtient :

$$\boxed{J_h(Y) = \sum_{i=1}^{N-1} \frac{(x_{i+1} - x_i)(y_{i+1} - y_i)}{2}\sqrt{1 + \left(\frac{y_{i+1} - y_i}{h}\right)^2} + hy_i + \frac{h^2}{\varepsilon}\left(\sum_{i=1}^{N-1}\sqrt{1 + \left(\frac{y_{i+1} - y_i}{h}\right)^2} - L\right)^2}$$

La longueur effective du fil :

$$\boxed{L = \boldsymbol{h}\sum_{i=1}^{N-1}\sqrt{1 + \left(\frac{y_{i+1} - y_i}{h}\right)^2}}$$

Avec :

- $x_i = x_a + (i-1)h$
    - $h = \frac{x_b - x_a}{N-1}$

Tâche 3 :

Evaluation de la valeur de J_h *pour un vecteur à* N *composantes*

Y, *un paramètre de pénalisation* ε *et un pas de discrétisation* h:

```python
import numpy as np

def Jh(l,eps,h,Y):
    s1=0
    for i in range(1,N-1):
        s1+=h*Y[i]+((X[i+1]-X[i])*(Y[i+1]-Y[i])*(1+((Y[i+1]-Y[i])/h)**2)**0.5)/2
    s2=0
    for i in range(1,N-1):
        s2+=(1+((Y[i+1]-Y[i])/h)**2)**0.5-l
    return( s1+((h*s2)**2)/eps )

Xa=0
Xb=1
Ya=2
Yb=3
l=2
eps=0.25

Y=[2,2.08,2.1,2.3,2.6,2.7,2.9,3]
N=np.size(Y)
h=(Xb-Xa)/(N-1)
X=[Xa+(i-1)*h for i in range(1,N+1)]
print("la valeur de Jh(Y)= ",Jh(l,eps,h,Y))
```

Tâche 4 :

On cherche le minimum d'une fonction f définie en tout

$$point\ d'un\ intervalle\ [a,b]:$$

```python
def glodensearch(f,a,b):
    erreur=b-a
    taux=(1+5**0.5)/2
    nbre_iteration=0
    while(erreur>taux):
        a1=a+(b-a)/(taux**2)
        b1=a+(b-a)/taux
        if(f(a1)>f(b1)):
           a=a1
        elif(f(a1)<f(b1)):
            b=b1
        elif(f(a1)==f(b1)):
            a=a1
            b=b1
        erreur=b-a
        nbre_iteration+=1
    return(a,nbre_iteration)
```

Tâche 5 :

On cherche à déterminer les $N-2$ *composantes de* $\frac{\partial J_h(Y)}{\partial y_j}$, *avec* $2 \leq j \leq N\_1$

$$\frac{\partial J_h(Y)}{\partial y_j} = h + \frac{(x_{j+1} - x_j)}{2}\left(-\sqrt{1 + \left(\frac{y_{j+1} - y_j}{h}\right)^2} - \frac{\left(\frac{y_{j+1} - y_j}{h}\right)^2}{\sqrt{1+\left(\frac{y_{j+1} - y_j}{h}\right)^2}} \right) +$$

$$\frac{(x_j - x_{j-1})}{2}\left(\sqrt{1 + \left(\frac{y_j - y_{j-1}}{h}\right)^2} + \frac{\left(\frac{y_j - y_{j-1}}{h}\right)^2}{\sqrt{1+\left(\frac{y_j - y_{j-1}}{h}\right)^2}} \right) +$$

$$\frac{2h^2}{\varepsilon}\left(\sum_{i=1}^{N-1} \sqrt{1 + \left(\frac{y_{i+1} - y_i}{h}\right)^2} - L \right)\left(\frac{\left(\frac{y_j - y_{j-1}}{h}\right)^2}{\sqrt{1+\left(\frac{y_j - y_{j-1}}{h}\right)^2}} - \frac{\left(\frac{y_{j+1} - y_j}{h}\right)^2}{\sqrt{1+\left(\frac{y_{j+1} - y_j}{h}\right)^2}} \right)$$

Tâche 6 :

On renvoie la valeur en Y du gradient de J_h:

```python
import numpy as np

def DJh(Y):
    grad=np.zeros(np.size(Y))
    s=0
    for i in range(1,N-1):
        s+=(1+((Y[i+1]-Y[i])/h)**2)**0.5-1
    for j in range(2,N-1):
        c=(1+((Y[j+1]-Y[j])/h)**2)**0.5
        c2=c+(((Y[j+1]-Y[j])/h)**2)/c
        c3=2*((Y[j+1]-Y[j])**2)/(eps*c)*s
        c4=h+(X[j+1]-X[j])/2*(-c2)-c3

        c1=(1+((Y[j]-Y[j-1])/h)**2)**0.5
        c21=c1+(((Y[j]-Y[j-1])/h)**2)/c1
        c31=2*((Y[j]-Y[j-1])**2)/(eps*c1)*s
        c41=(X[j]-X[j-1])/2*c21+c31

        grad[j]=c4+c41
    return( grad )
```

Tâche 7 :

- *On implémente la méthode de gradient* à *pas fixe*:

➢ *On implémente la méthode de gradient à pas variable*:

```
def gradient_pas_fixe(p,Y0):
    Y1=Y0-p*DJh(Y0)
    nb_iteration=0
    while(np.linalg.norm((Y1-Y0))>eps and Jh(Y1)>Jh(Y0)):
        Y0=Y1
        Y1=Y0-p*DJh(Y0)
        nb_iteration+=1
    return [Y0,nb_iteration]
```

```
def gradient_pas_variable(Y0):
    def f(x):
            return(Jh(Y0-x*DJh(Y0)))
    pas=glodensearch(f,Xa,Xb)[0]
    Y1=Y0-pas*DJh(Y0)
    nb_iteration=0
    while((np.linalg.norm((Y1-Y0))>eps) and (Jh(Y1)<Jh(Y0))):
        def f(x):
            return(Jh(Y0-x*DJh(Y0)))
        pas=glodensearch(f,Xa,Xb)[0]
        Y0=Y1
        Y1=Y0-pas*DJh(Y0)
        nb_iteration+=1
    return [Y1,nb_iteration]
```

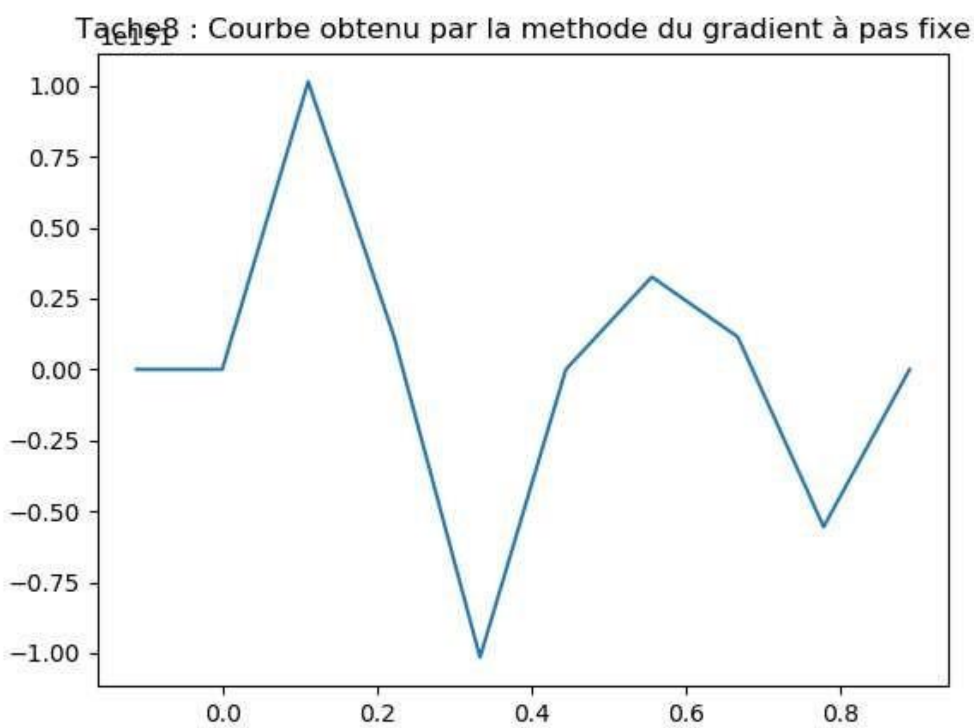
Tâche 8 :

➢ *méthode de gradient à pas fixe*:

```
import numpy as np
import matplotlib.pyplot as plt

#calcul de Jh :
def Jh(Y):
    s1=0
    for i in range(1,N-1):
        s1+=h*Y[i]+((X[i+1]-X[i])*(Y[i+1]-Y[i])*(1+((Y[i+1]-Y[i])/h)**2)**0.5)/2
    s2=0
    for i in range(1,N-1):
        s2+=(1+((Y[i+1]-Y[i])/h)**2)**0.5-1
    return( s1+((h*s2)**2)/eps )


# cacul du gradient :
def DJh(Y):
    grad=np.zeros(np.size(Y))
    s=0
    for i in range(1,N-1):
        s+=(1+((Y[i+1]-Y[i])/h)**2)**0.5-1
    for j in range(2,N-1):
        c=(1+((Y[j+1]-Y[j])/h)**2)**0.5
        c2=c+(((Y[j+1]-Y[j])/h)**2)/c
        c3=2*((Y[j+1]-Y[j])**2)/(eps*c)*s
        c4=h+(X[j+1]-X[j])/2*(-c2)-c3

        c1=(1+((Y[j]-Y[j-1])/h)**2)**0.5
        c21=c1+(((Y[j]-Y[j-1])/h)**2)/c1
        c31=2*((Y[j]-Y[j-1])**2)/(eps*c1)*s
        c41=(X[j]-X[j-1])/2*c21+c31

        grad[j]=c4+c41
    return( grad )
```

```
def gradient_pas_fixe(p,Y0):
    Y1=Y0-p*DJh(Y0)
    nb_iteration=0
    while(np.linalg.norm((Y1-Y0))>eps and Jh(Y1)>Jh(Y0)):
        Y0=Y1
        Y1=Y0-p*DJh(Y0)
        nb_iteration+=1
    return [Y0,nb_iteration]
```

```
# programme principale :
Xa=0
Xb=1
Ya=2
Yb=3
l=2
eps=1/4
N=10

h=(Xb-Xa)/(N-1)
X=[(i-1)*h+Xa for i in range(N)]
Y0=[h*(i-1) for i in X]
Y0[0]=Ya
Y0[N-1]=Yb

plt.title("Tache8 : Courbe obtenu par la methode du gradient à pas fixe")
plt.plot(X,gradient_pas_fixe(0.5,Y0)[0])
plt.show()
```

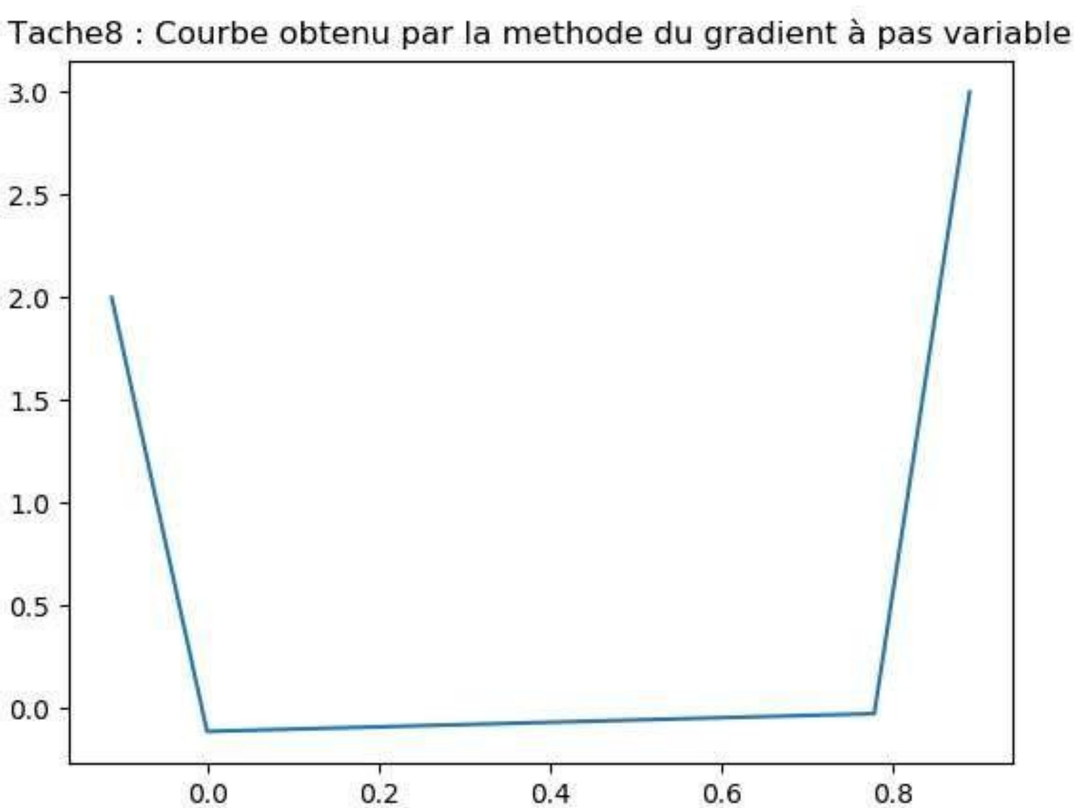


<u>Commentaire</u> : cette méthode présente des erreurs dûes au choix aléatoire du pas.

➢ *méthode de gradient* à *pas variable*:

```
import numpy as np
import matplotlib.pyplot as plt

#calcul de Jh :
def Jh(Y):
    s1=0
    for i in range(1,N-1):
        s1+=h*Y[i]+((X[i+1]-X[i])*(Y[i+1]-Y[i])*(1+((Y[i+1]-Y[i])/h)**2)**0.5)/2
    s2=0
    for i in range(1,N-1):
        s2+=(1+((Y[i+1]-Y[i])/h)**2)**0.5-1
    return( s1+((h*s2)**2)/eps )

# cacul du gradient :
def DJh(Y):
    grad=np.zeros(np.size(Y))
    s=0
    for i in range(1,N-1):
        s+=(1+((Y[i+1]-Y[i])/h)**2)**0.5-1
    for j in range(2,N-1):
        c=(1+((Y[j+1]-Y[j])/h)**2)**0.5
        c2=c+(((Y[j+1]-Y[j])/h)**2)/c
        c3=2*((Y[j+1]-Y[j])**2)/(eps*c)*s
        c4=h+(X[j+1]-X[j])/2*(-c2)-c3

        c1=(1+((Y[j]-Y[j-1])/h)**2)**0.5
        c21=c1+(((Y[j]-Y[j-1])/h)**2)/c1
        c31=2*((Y[j]-Y[j-1])**2)/(eps*c1)*s
        c41=(X[j]-X[j-1])/2*c21+c31

        grad[j]=c4+c41
    return( grad )

#fonction goldensearch :
def glodensearch(f,a,b):
    erreur=b-a
    taux=(1+5**0.5)/2
    nbre_iteration=0
    while(erreur>taux):
        a1=a+(b-a)/(taux**2)
        b1=a+(b-a)/taux
        if(f(a1)>f(b1)):
           a=a1
        elif(f(a1)<f(b1)):
            b=b1
        elif(f(a1)==f(b1)):
            a=a1
            b=b1
        erreur=b-a
        nbre_iteration+=1
    return[a,nbre_iteration]
```

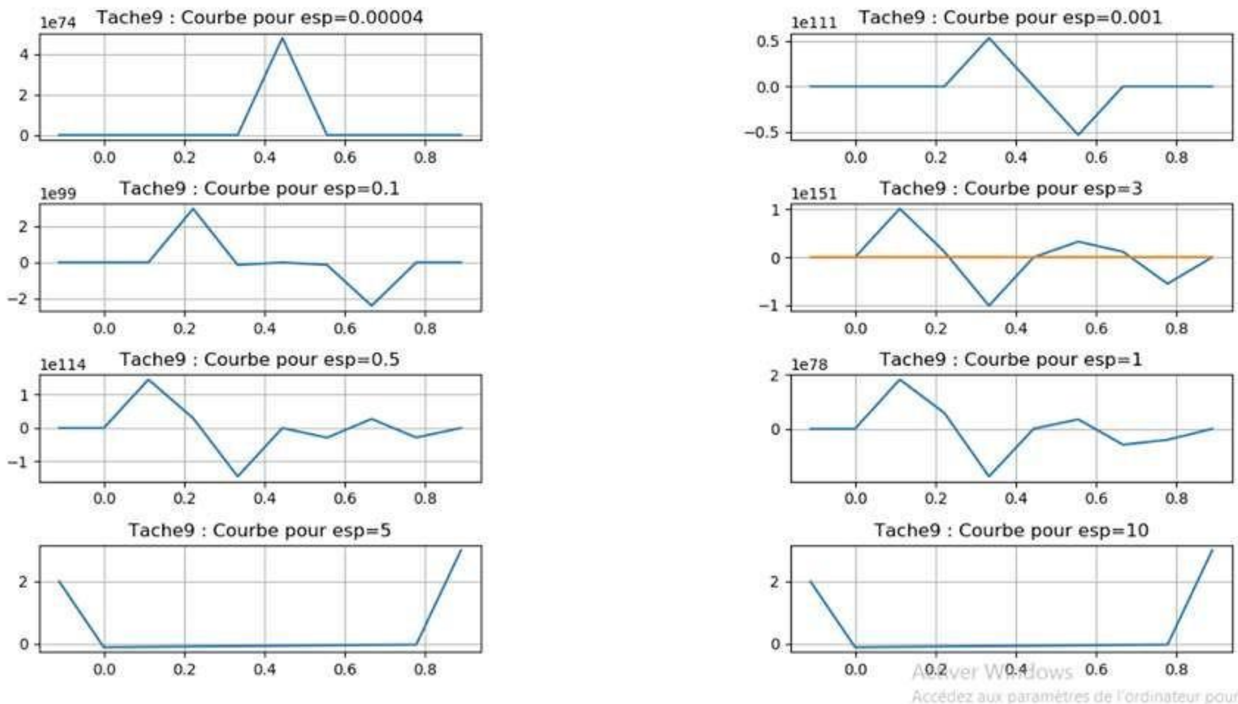
```python
# programme principale :
Xa=0
Xb=1
Ya=2
Yb=3
l=2
eps=1/4
N=10

h=(Xb-Xa)/(N-1)
X=[(i-1)*h+Xa for i in range(N)]
Y0=[h*(i-1) for i in X]
Y0[0]=Ya
Y0[N-1]=Yb

plt.title("Tache8 : Courbe obtenu par la methode du gradient à pas variable")
plt.plot(X,gradient_pas_variable(Y0)[0])
plt.show()
```



Tâche 9 :

Variation de ε pour la méthode de gradient à pas fixe:

```python
#on fait varier epsilon :

    #pour : eps=0.00004
eps=0.00004
plt.subplot(4,2,1)
plt.title("Tache9 : Courbe pour esp=0.00004")
plt.plot(X,gradient_pas_fixe(0.5,Y0)[0])
plt.grid(True)

    #pour : eps=0.001
eps=0.001
plt.subplot(4,2,2)
plt.title("Tache9 : Courbe pour esp=0.001")
plt.plot(X,gradient_pas_fixe(0.5,Y0)[0])
plt.grid(True)

    #pour : eps=0.1
eps=0.1
plt.subplot(4,2,3)
plt.title("Tache9 : Courbe pour esp=0.1")
plt.plot(X,gradient_pas_fixe(0.5,Y0)[0])
plt.grid(True)

  #pour : eps=1/4
eps=1/4
plt.subplot(4,2,4)
plt.title("Tache9 : Courbe pour esp=1/4")
plt.plot(X,gradient_pas_fixe(0.5,Y0)[0])
plt.grid(True)

  #pour : eps=0.5
eps=0.5
plt.subplot(4,2,5)
plt.title("Tache9 : Courbe pour esp=0.5")
plt.plot(X,gradient_pas_fixe(0.5,Y0)[0])
plt.grid(True)


  #pour : eps=1
eps=1
plt.subplot(4,2,6)
plt.title("Tache9 : Courbe pour esp=1")
plt.plot(X,gradient_pas_fixe(0.5,Y0)[0])
plt.grid(True)

  #pour : eps=3
eps=3
plt.subplot(4,2,4)
plt.title("Tache9 : Courbe pour esp=3")
plt.plot(X,gradient_pas_fixe(0.5,Y0)[0])
plt.grid(True)

  #pour : eps=5
eps=5
plt.subplot(4,2,7)
plt.title("Tache9 : Courbe pour esp=5")
plt.plot(X,gradient_pas_fixe(0.5,Y0)[0])
plt.grid(True)

  #pour : eps=10
eps=10
plt.subplot(4,2,8)
plt.title("Tache9 : Courbe pour esp=10")
plt.plot(X,gradient_pas_fixe(0.5,Y0)[0])
plt.grid(True)


plt.subplots_adjust(top=0.9, bottom=0.1, left=0.10, right=0.95, hspace=0.6,
                    wspace=0.7)
plt.show()
```



<u>Commentaire :</u>

Lorsqu'on fait diminuer la valeur de ε, on remarque que la courbe de la méthode du pas fixe est différente de celle de la méthode du pas variable.

Alors que lorsqu'on augmente la valeur de ε, la courbe de la méthode du pas fixe devient de même allure que celle de la méthode du pas variable.