École Polytechnque de Tunisie

TP Optimisation

Optimisation sous contraintes : Méthode de projection

Travail de :

Mohamed SAIDI

Houssem Majed

Travail encadré par : M. MOAKHER Maher, Mme. CHOUAIEB Nadia

June 11, 2021

0.0.1 Optimisation sous contraintes

On se propose dans ce TP de résoudre numériquement le problème de minimisation sous contrainte suivant :

$$\min_{u \in K} J(u)$$

où

$$J(u) = \int_0^1 \left(\frac{1}{2} u'(x)^2 - f(x)u(x) \right) dx$$
$$K = \left\{ u \in H^1(]0,[)1 : u(0) = u(1) = 0, u(x) \geq g(x) \forall x \in [0,1] \right\}$$

avec f et g sont deux fonctions données.

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from pylab import *
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.colors import LogNorm
import time
import pandas as pd
from math import pi
import scipy as sp
from scipy.optimize import minimize, rosen, rosen_der
from numpy.linalg import eig
```

[ ]:
```
#Definition de la matrice A et du vecteur b
n=int(input("entrer un entier n"))
h=1/n
A = (4/h**2)*np.diag(np.ones(n))-(2/h**2)*np.diag(np.ones(n-1),-1)-(2/h**2)*np.
 ↪diag(np.ones(n-1),1)
L=[]
for i in range (n) :
    b = L.append([1])
b=np.array(L)
v=[]
for i in range(n) :
    v.append([0])
y = np.array(v)
```

[3]:
```
# Implémentation de la solution exacte du problème (1)
def u(x):
    if(0<=x<=0.336526877):
        return(-0.5*x*(x-1))
    if(0.868601328<=x<=1):
        return(-0.5*x*(x-1))
    else:
        return(1.5-20*((x-0.6)**2))
```

[4]:
```
# Implémentation de la solution exacte
def solution(n):
    s=np.zeros(n)
    h=1/(n+1)
    for i in range(0,n):
        s[i]=u((i+1)*h)
    print("La solution exacte du problème est:")
    print(s)
    print("\n")
```

[11]:
```
#Definition de la fonction J
def J(n,x,f) :
    return (1/2)*np.vdot(np.dot(A,x),x)-np.vdot(b,x)
```

[12]:
```
#Definition du gradient de J
def DJ(n,u,f):
    return(np.dot(A,u)-b)
```

[13]:
```
def f(x):
    return (1)
    #return(pi**2*np.sin(pi*x))
def g(x):
    return(max(1.5-20*(x-0.6)**2,0))
def gn(n):
    s=[]
    h=1/(n+1)
    for i in range(0,n):
        s.append([g((i+1)*h)])
    t=np.array(s)
    return(t)
```

1 Partie 1− Discrétisation et résolution directe

Pour trouver une solution approchée de ce problème de minimisation, on procède comme on a fait dans le TP 1 , on discrétise l'intervalle $[0,1]$ en $n+1$ sous intervalles égaux $[x_i, x_{i+1}]$, $i = 0, \ldots, n$, où $x_i = ih$ avec $h = 1/(n+1)$. Puis, on approche la solution $u(x)$ par une fonction $u_h(x)$, continue, affine par morceaux et donnée par

$$u_h(x) = \sum_{i=1}^{n} u_i \phi_i(x)$$

$$\phi_i(x) = \begin{cases} \frac{x-x_{i-1}}{h} & \text{si } x \in [x_{i-1}, x_i] \\ \frac{x_{i+1}-x}{h} & \text{si } x \in [x_i, x_{i+1}] \\ 0 & \text{si non} \end{cases},$$

et u_i sont des valeurs approchées de $u\left(x_i\right)$. On pose alors

$$K_h = \left\{ u_h(x) = \sum_{i=1}^{n} u_i \phi_i(x), u_i \geq g\left(x_i\right) \forall i = 0, \ldots, n+1 \right\}.$$

On approche le problème (1) par le suivant:

$$\min_{u_h \in K_h} J\left(u_h\right)$$

1.0.1 1.1 Expliciter ce problème (2) en utilisant notamment la formule des trapèzes pour calculer les intégrales et montrer qu'il peut se mettre sous la forme suivante

$$\min_{\boldsymbol{u} \in K^n} J_n(\boldsymbol{u})$$

où $K^n = \{\boldsymbol{v} \in \mathbb{R}^n : v_i \geq g\left(x_i\right) \forall 1 \leq i \leq n\}$ et J_n est à expliciter.

Soit $x \in [0,1]$, Nous avons alors: $J(u) = \int_0^1 \left(\frac{1}{2} u'(x)^2 - f(x)u(x)\right) dx$, $\rightarrow J(u) = \int_0^1 \left(\frac{1}{2}\left(\sum_{i=1}^n u_i \Phi_i'(x)\right)^2 - \sum_{i=1}^m f(x) u_i \cdot \Phi_i(x)\right) dx \rightarrow J(u) = \int_0^1 \frac{1}{2}\left(\sum_{i=1}^m u_i \Phi_i'(x)\right)^2 dx - \int_0^1 \sum_{i=1}^n f(x) u_i \Phi_i(x) dx \rightarrow J(u) = \sum_{k=0}^n \int_{x_k}^{x_{k+1}} \frac{1}{2}\left(\sum_{i=1}^n u_i \Phi_i'(x)\right)^2 dx - \sum_{k=0}^n \int_{x_k}^{x_{L+1}} \sum_{i=1}^n f(x) u_i \Phi_i(x) dx \rightarrow J(u) = \sum_{k=0}^n \int_{x_k}^{x_{x+1}} \frac{1}{2} \sum_{i=1}^n u_i^2 \Phi_i'(x)^2 dx + \sum_{k=0}^n \int_{x_k}^{x_{u+1}} \left(\sum_{i=1}^n \sum_{j=i+1}^n u_i \Phi_i'(x) u_j \Phi_j'(x)\right) dx - \sum_{0 \leq i,k \leq n} \int_{x_k}^{x_{\lambda+1}} f(x) u_i \Phi_i(x) dx$ $\rightarrow J(u) = \frac{1}{2} \sum_{1 \leq i \leq n} u_i^2 \frac{2}{h} + \frac{1}{2} \sum_{k=0}^n \sum_{i=1}^n \sum_{j=i+1}^n u_i u_j \int_{x_k}^{x_{h+1}} \Phi_i'(x) \Phi_j'(x) dx - \sum_{0 \leq ik \leq n} u_i \int_{x_k}^{x_{i+1}} f(x) \Phi_i(x) dx \rightarrow J(u) = \frac{1}{h} \sum_{1 \leq i \leq n} u_i^2 + \sum_{1 \leq i \leq n} u_i u_{i-1} \int_{x_{i-1}}^{x_i} -\left(\frac{1}{h}\right)^2 dx - \sum_{0 \leq i,k \leq n} u_i \left(x_{k+1} - x_k\right) \frac{f(x_{k+1}) \Phi_i(x_{k+1}) + f(x_k) \Phi_i(x_k)}{2} \rightarrow J(u) = \frac{1}{h} \sum_{1 \leq i \leq n} u_i^2 - \frac{1}{h} \sum_{1 \leq i \leq n} u_i u_{i-1} - \sum_{1 \leq i \leq n} u_i h \frac{2 f(x_i)}{2} \rightarrow J(u) = \frac{1}{h} \sum_{1 \leq i \leq n} u_i^2 - \frac{1}{h} \sum_{1 \leq i \leq n} u_i u_{i-1} - \sum_{1 \leq i \leq n} u_i h f\left(x_i\right) \rightarrow J(u) = h\left(\frac{1}{2h^2}\left(2 \sum_{1 \leq i \leqslant n} u_i^2 - 2 \sum_{1 \leq i \leq n} u_i u_{i-1}\right) - \sum_{1 \leq i \leq n} u_i f\left(x_i\right)\right) \rightarrow J(u) = h\left(\frac{1}{2}\langle Au, u\rangle - \langle b, u\rangle\right) = h.J_n(u)$
on a aussi $\mathrm{K}_h \subset K^n$

1.0.2 1.2 Montrer que le problème (3) admet une unique solution

on va montrer que Kn est fermé : on sait que la fonction $\rightarrow x - g(x)$ continue alors l'ensemble Kn peut s'écrit sous la forme : $K^n = \{v \in IR^m : k\left(v_i\right) \geq 0 \forall 1 \leq i \leq n\}$ ce qui est bien un fermé . on va montrer aussi que K^m est convexe : Soit v et w deux vecteurs de K^n et soit t $\in [0,1]$. Nous avons alors $v_i \geq g\left(x_i\right)$ et $u_i \geq g\left(x_i\right) \rightarrow (1-t)v_i \geq (1-t)g\left(x_i\right)$ et $tu_i \geq \operatorname{tg}\left(x_i\right) \rightarrow (1-t)v_i + tu_i \geq g\left(x_i\right)$ Alors $(1-t)v_i + tu_i \in K^n$ par suite on a K^n est convexe en outre K^N est un ensemble convexe fermé non vide et que J est continue, coersive et. strictement convexce done le problème (3) admet une unique solution...

1.0.3 1.4 Pour $n = 100, f(x) = 1, g(x) = \max\left(1.5 - 20(x-0.6)^2, 0\right)$, résoudre le problème (3) a l'aide de scipi.optimize.minimize.

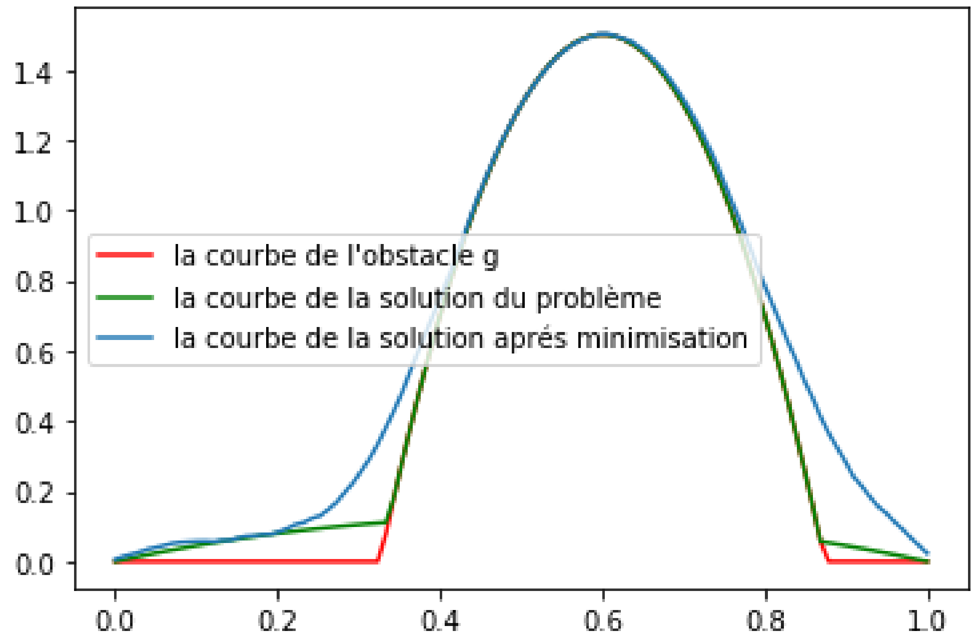
[3]:
```
# représentation graphique de la solution aisni que l'obstacle
def graphe():
    x=np.linspace(0,1,100)
    y=np.array([g(x[i]) for i in range(np.size(x))])
    z=np.array([u(x[i]) for i in range(np.size(x))])
    a=minimize()
    w=np.array([a[i] for i in range(np.size(x))])
    plt.plot(x,y,label="la courbe de l'obstacle g",color='r')
    plt.plot(x,z,label="la courbe de la solution du problème",color='g')
    plt.plot(x,w,label="la courbe de la solution aprés minimisation")
    plt.legend()
    plt.show()
```

[19]:
```
def minimize():
    x = np.linspace(0,1, n + 2)
    xv = x[1:-1]
    fv = f(xv)
    gv=np.array([g(xv[i]) for i in range(np.size(xv))])
    Jf = lambda u : J(n,u,fv)
    DJf = lambda u : DJ(n, u, fv)
    const = ({'type': 'ineq', 'fun' : lambda u: u - gv,
              'jac' : lambda u: np.eye(np.size(u))})
    u = np.zeros(n)
    res =sp.optimize.minimize(Jf, u, method ='COBYLA', jac=DJf, constraints=const
                  , options={'tol': 1e-8,
                             'disp': True ,
                             'maxiter':5000})
    return(res.x)
```

1.0.4 Et représenter sur le même graphique la solution ainsi que l'obstacle. Tester aussi avec $f(x) = \pi^2 \sin(\pi x)$.

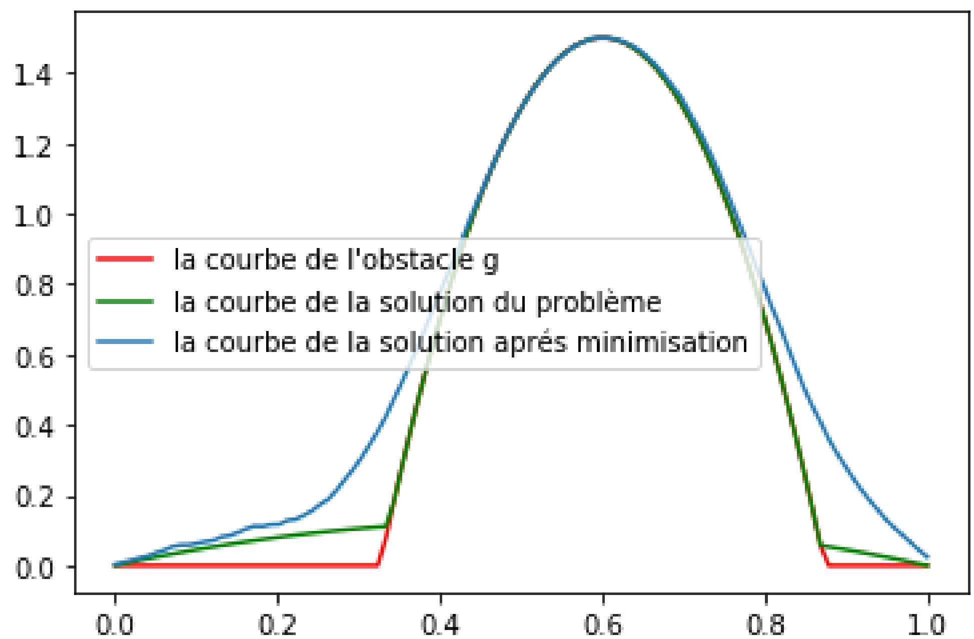
1.0.5 Pour $f(x) = 1$

[22]:
```
graphe()
```



1.0.6 Pour $f(x) = \pi^2 \sin(\pi x)$

[9]:
```
graphe()
```



1.0.7 1.5 Vérifier que la solution de (1) vérifie $u(0) = u(1) = 0$ et pour tout $x \in]0, 1[$:

$$-u''(x) \geq f(x),$$
$$u(x) \geq g(x),$$
$$\left(-u''(x) - f(x)\right)(u(x) - g(x)) = 0.$$

Linéquation (4.a) traduit une concavité maximale de la fonction u. L'inéquation (4.b) représente l'obstacle: on veut que la solution u soit au dessus de g. L'équation (4.c) traduit le fait que l'on a au moins égalité dans une des deux inéquations (4.a) et (4.b) : soit on résout $-u''(x) = f(x)$, soit $u(x) = g(x)$, et u est sur l'obstacle.

Soit $x|0, 1|$ on a : u la solution du problème (1), alors d'après l'énoncée de ce problème, nous avons immédiatement u(0) = u(1) = 0 et $u(x) \geq g(x)$, l'équation (4.b) est vérifiée. casn 1: si$u(x) = g(x)$ l'équation (4.c) est vérifiée. casn 2 : si$u(x) > g(x)$ Alors d'après l'équation d'Euler-Lagrange, nous avons :

$$\frac{d}{dx}\left(\frac{dL}{du'}\right) - \frac{dl}{du} = 0, \text{ avec L(u)} = \frac{1}{2}u'(x)^2 - f(x)u(x)$$

$\rightarrow \frac{d}{dx}\left(u'(x)\right) + f(x) = 0 \rightarrow u''(x) + f(x) = 0 \rightarrow \left(u''(x) + f(x)\right)(u(x) - g(x)) = 0$Et par la suite, l'équation (4.c) est vérifiée. D'après l'équation (4.c), nous avons :

$$u''(x) = \begin{cases} -1 \text{ si } x \in [0, 0.326] \cup [0.874, 1] \\ -40 \text{ si } x \in [0.326, 0.874] \end{cases}$$

Donc, d'après ce qui précde : $-u''(x)1 = f(x)$ et par la suite l'équation (4.a) est vérifiée. L'inéquation (4.a) traduit une concavité maximale de la fonction u. L'inéquation (4.b) représente l'obstacle : on veut que la solution u soit au dessus de g. Léquation (4.c) traduit le fait que l'on a au moins égalité dans une des deux inéquations (4.a) et (4.b) : soit on résout $-u''(x) = f(x)$, soit $u(x) = g(x)$, et u est sur l'obstacle.

1.0.8 Partie 2 - Méthode du gradient projeté

1.0.9 2.1 Montrer que K^n est un ensemble convexe fermé non vide et que l'opérateur projection P_{K^n} sur K^n est défini pour tout $v \in \mathbb{R}^n$ par

$$\left(P_{K^n}(\boldsymbol{v})\right)_i = \max\left(g_i, v_i\right), \quad i = 1, \ldots, n.$$

soit k_0 un sous-espace vectoriel de $IR^n, a \in R^n, K^n = a + k_0$ et $h \in k_0$ arbitraire et fixé. On va prendre : $\mathrm{K}^n = Pv \pm h$ qui est un élément de kn car $Pv \in k$ et $h \in k_0$. Ceci donne

$$\pm < v - Pv, h > \leq 0$$

d'où on déduit

$$\langle v - Pv, h >= 0$$

$(\mathrm{P}_{K^n}(v))_i = \begin{cases} gi & \text{si} \quad v > gi \\ vi & \text{si} \quad v \in K^{\mathrm{n}} \end{cases}$ On peut aussi noter dans ce cas :

$$\left(P_{K^n}(v)\right)_i = \max\left(g_i, v_i\right), \quad i = 1, \ldots, n$$

1.0.10 2. 2 De manière générale, montrer que si

$$K^n = \{\boldsymbol{v} \in \mathbb{R}^n : a_i \leq v_i \leq b_i\}$$

on a pour tout $\boldsymbol{v} \in \mathbb{R}^n$

$$(P_{K^n}(\boldsymbol{v}))_i = \max(a_i, \min(v_i, b_i)), \quad i = 1, \ldots, n,$$

et que cette définition est valable même si l'on a $a_i = -\infty$ ou $b_i = +\infty$ pour certains $1 \leq i \leq n$

2.2 De manière générale, montrer que si

$$K^n = \{\boldsymbol{v} \in R^n : a_i \leq v_i \leq b_i\}$$

on a pour tout $v \in R^n$

$$(P_{K^n}(v))_i = \max(a_i, \min(v_i, b_i)), \quad i = 1, \ldots, n$$

et que cette définition est valable mème sill'on a $a_i = -\infty$ ou $b_i = +\infty$ pour certains $1 \leq i \leq n$ 5

$$K^n = \{\boldsymbol{v} \in R^n : a_i \leq v_i \leq b_i\} = k1 \cup k2$$

d'apres la question précédente on pour l'égalité dans k1:

$$(P_{K^n}(v))_i = \max(a_i, v_i), \quad i = 1, \ldots, n$$
$$\min(b_i, v_i) \leq v_i$$

d'ou

$$(P_{K^n}(v))_i = \max(\min(b_i, v_i), v_i), \quad i = 1, \ldots, n$$

donc on a:

$$(P_{K^n}(v))_i = \max(a_i, \min(b_i)), \quad i = 1, \ldots, n$$

1.0.11 2. 3 Ecrire un programme projK. py qui prend en argument un point $\boldsymbol{u}$ et $g_n = (g(x_i))_{i=1,\ldots,n}$ et qui renvoie $P_{K^n}(\boldsymbol{u})$.

[9]:
```
def projK(u,gn):
    s=np.maximum(u,gn)
    return s
```

1.1 Question 2.4

[8]:
```
#Definition de la fonction gradient projeté
X=[]
Y=[]
X1=[]
Y1=[]
def gradproj(Max,q,tol,gn,n) :
    u=y
    k=0
    r=tol
```

```
    X.append(u[0])
    Y.append(u[1])
    while (r>=tol) and (k<=Max) :
        w=-np.dot(A,u)+b
        v=u
        p=v+q*w
        u=np.maximum(p,gn(n))
        r=np.linalg.norm(u-v)
        k=k+1
        X.append(u[0])
        Y.append(u[1])
    for m in range(len(X)) :
        X1.append(X[m][0])
        Y1.append(Y[m][0])
    return u
```

[108]:
```
#La solution est:
gradproj(100,0.1,0.00001,gn)
```

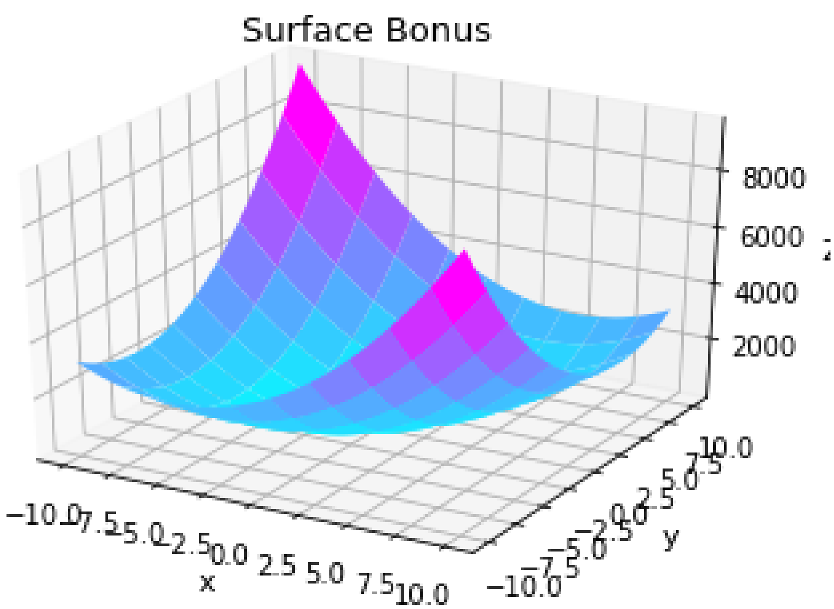
[108]:
```
array([[0.76805322],
       [1.41111111]])
```

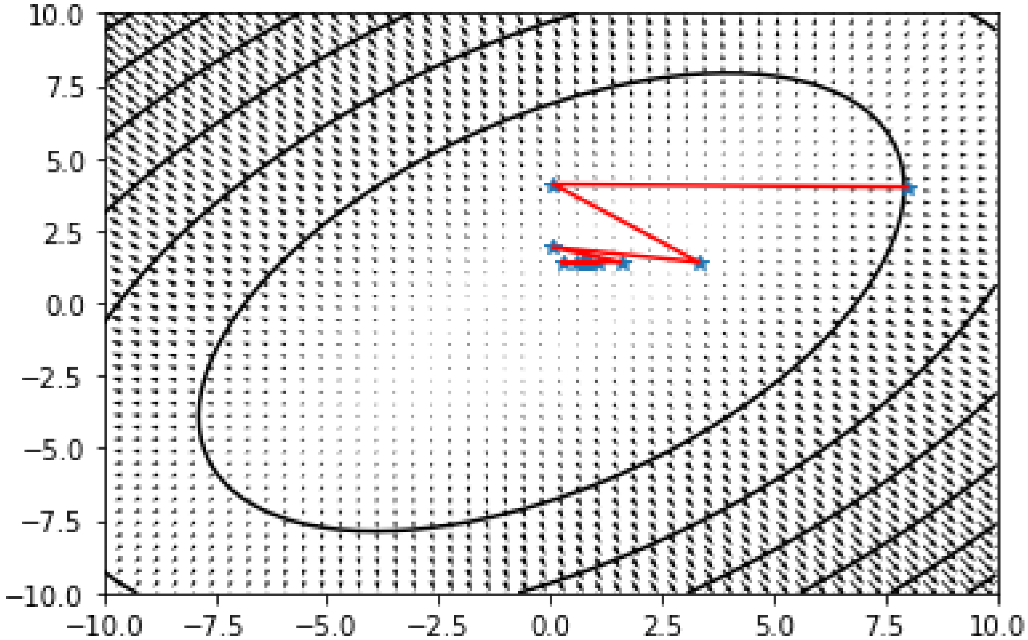
1.1.1 2.5 Pour $n = 2$, tracer sur une même figure les courbes de niveaux de J_2 ainsi que le ch amp de vecteurs ∇J_2 sur le pavé $[-10, 10] \times [-10; 10]$, calculer les itérations $u^{(k)} = \left(u_1^{(k)}, u_2^{(k)}\right)$ données par l' algorithme de gradient projeté à pas fixe, et tracer sur la même figure que précédemment la ligne qui relie les $u^{(k)}$. On prendra $u^{(0)} = (8, 4), \rho = 0.1$ et $\mathrm{Tol} = 10^{-5}$

[10]:
```
#la fonction qui permet de faire le tracage
def tracer(N,n) :
    x = np.linspace(-10,10,N)
    y= np.linspace(-10,10,N)
    x,y= np.meshgrid(x,y)
    r=np.zeros((N,N))
    lvls = np.logspace(-4,0,20)
    dx=np.zeros((50,50))
    dy=np.zeros((50,50))
    for i in range(N) :
        for j in range(N) :
            w=np.array([[x[i,j]],[y[i,j]]])
            r[i,j]=(1/2)*np.vdot(np.dot(A/(h**2),w),w) - np.vdot(b,w)

    fig = plt.figure()
    ax = plt.axes(projection='3d')
    fig1,ax1=plt.subplots(1,1)
```

```
    for i in range(N) :
        for j in range(N) :
            w=np.array([[x[i,j]],[y[i,j]]])
            f=np.dot(A,w)-b
            dx[i,j]=f[0]
            dy[i,j]=f[1]

    plt.quiver(x,y,dx,dy,cmap=cm.cool)
    cp = ax1.contour(x, y, r,colors='k')
    plt.plot(X1,Y1,'*')
    plt.plot(X1,Y1,'r')
    ax.plot_surface(x, y, r, rstride=5, cstride=5,
                cmap='cool')
    ax.set_title("Surface Bonus", fontsize = 13)
    ax.set_xlabel('x', fontsize = 11)
    ax.set_ylabel('y', fontsize = 11)
    ax.set_zlabel('Z', fontsize = 11)
    plt.show()
```

[29]:
```
tracer(50,2)
```



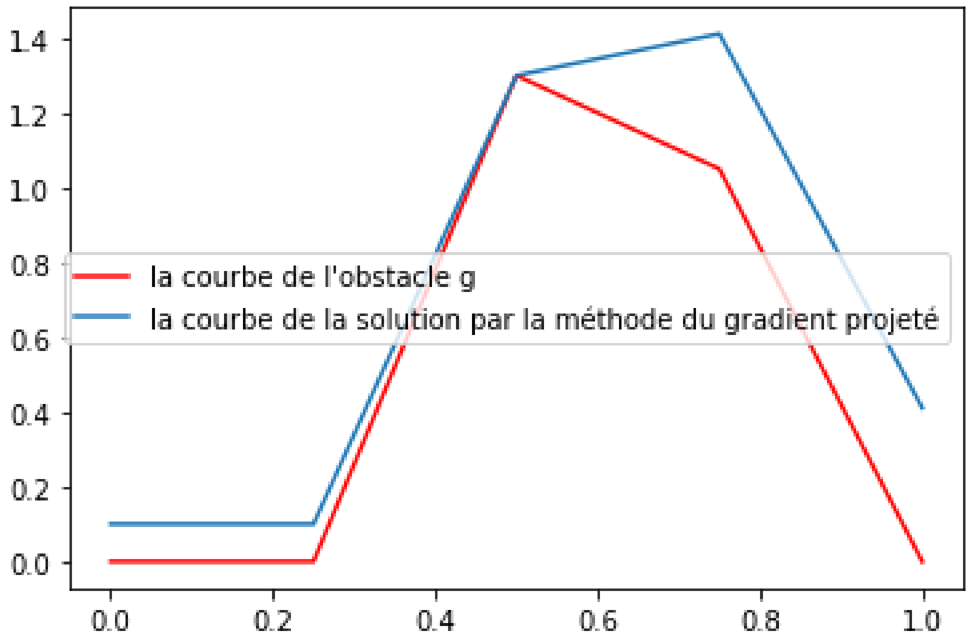


1.1.2 2.6 Pour $n = 5, 20, 50, 100$, afficher a l'aide de la fonction print le nombre d itérations ainsi que le temps de calcul pour ch aque n. Tracer sur une même figure les solutions approchées u_n, ainsi que le graphe de la fonction g. On prendra $\rho = 0.1, \mathrm{Tol} = 10^{-5}$.

[37]:
```
ps1 = time.perf_counter()
print("Le nombre d'itération est")
print(gradproj(10000,0.1,0.00001,gn,5))
tps2 = time.perf_counter()
print("le temps d'execution pour n=5")
print(tps2-tps1)
```

```
Le nombre d'itération est
331
le temps d'execution pour n=5
0.012329299999976229
```

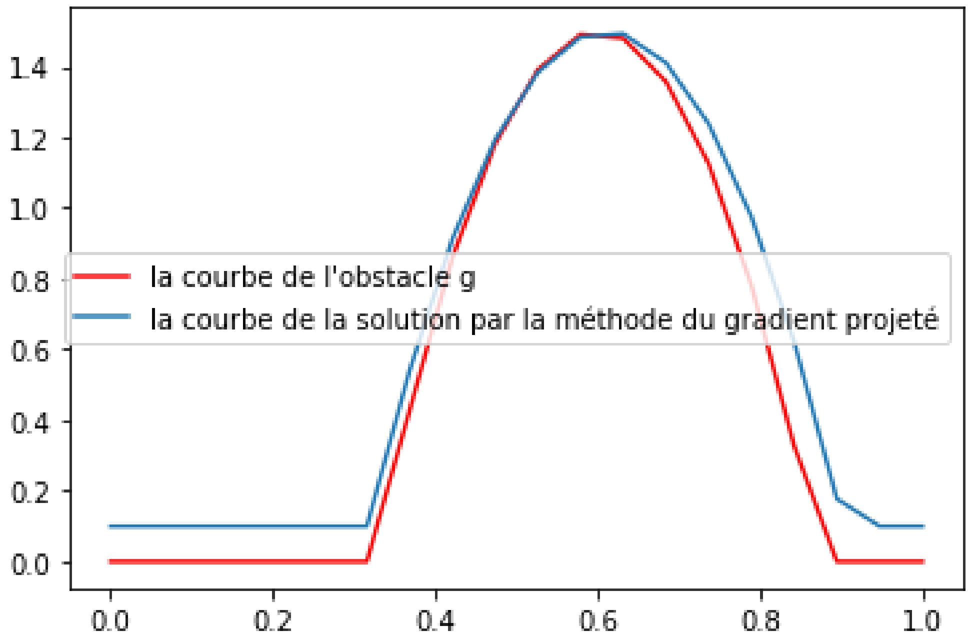
[50]:
```
graphe()
```



[32]:
```
tps3 = time.perf_counter()
print("Le nombre d'itération est")
print(gradproj(10000,0.1,0.00001,gn,20))
tps4 = time.perf_counter()
print("le temps d'execution pour n=20")
print(tps4-tps3)
```

```
Le nombre d'itération est
143
le temps d'execution pour n=20
0.008600999999998749
```

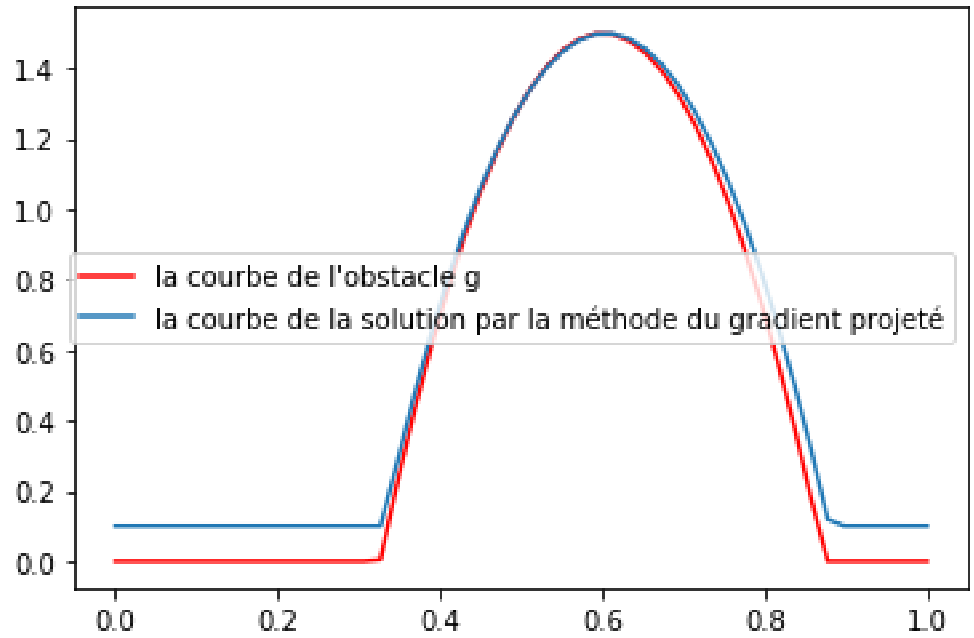
[54]:
```
graphe()
```



[23]:
```
tps5 = time.perf_counter()
print("Le nombre d'itération est")
print(gradproj(10000,0.1,0.00001,gn,50))
tps6 = time.perf_counter()
print("le temps d'execution pour n=50")
print(tps6-tps5)
```

```
Le nombre d'itération est
106
le temps d'execution pour n=50
0.008995600000019977
```

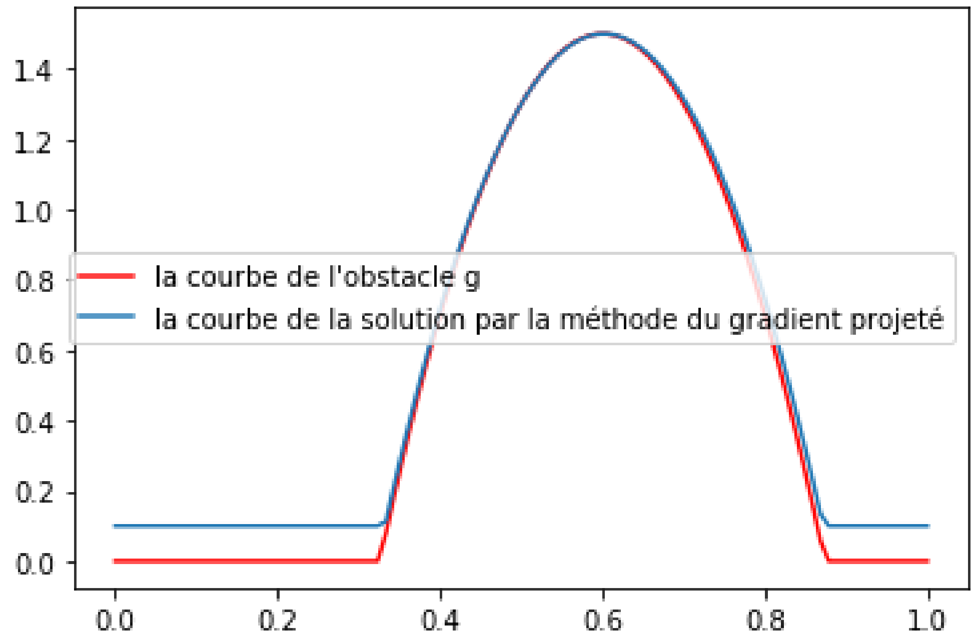
[56]:
```
graphe()
```



[25]:
```
tps7 = time.perf_counter()
print("le nombre d'itération")
print(gradproj(10000,0.1,0.00001,gn,100))
tps8 = time.perf_counter()
print("le temps d'execution pour n=100")
print(tps8-tps7)
```

```
le nombre d'itération
88
le temps d'execution pour n=100
0.013118300000002137
```

[58]:
```
graphe()
```



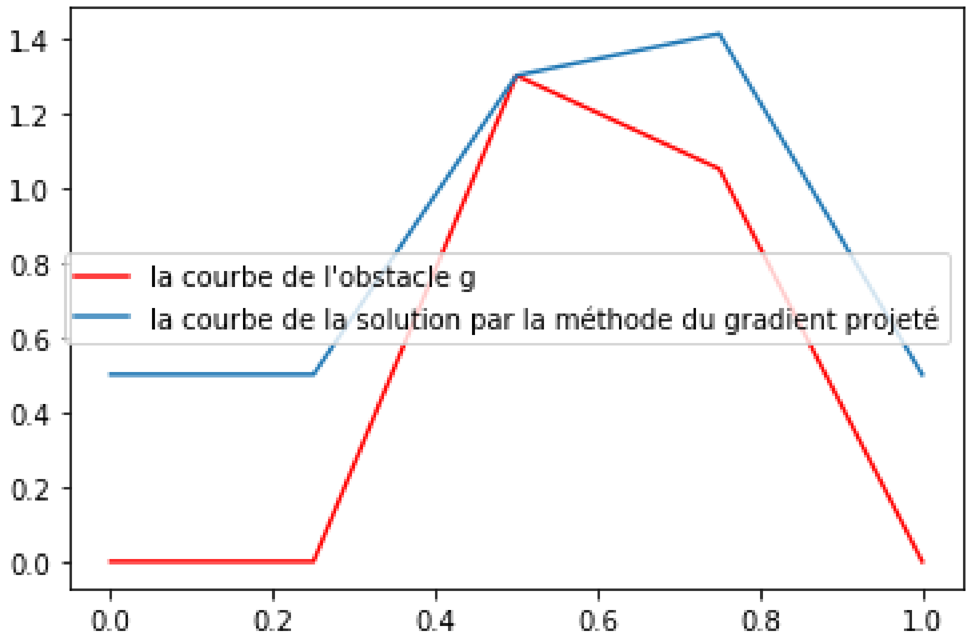
1.1.3 2.7 Reprendre l'expérience précédente pour $\rho = 0.5$, puis $\rho = 1$. Que constate-t-on ? Peut-on choisir le pas ρ arbitrairement?

1.1.4 Pour $\rho = 0.5$

[39]:
```
tps1 = time.perf_counter()
print("Le nombre d'itération est")
print(gradproj(10000,0.5,0.00001,gn,5))
tps2 = time.perf_counter()
print("le temps d'execution pour n=5")
print(tps2-tps1)
```

```
Le nombre d'itération est
191
le temps d'execution pour n=5
0.007262699999955657
```

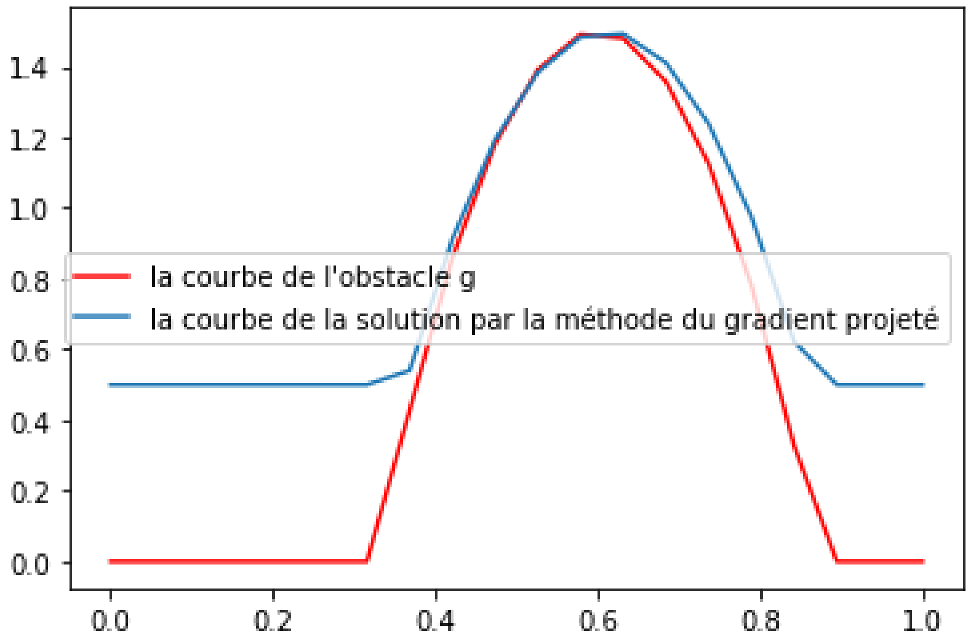
[65]:
```
graphe()
```



[42]:
```
tps3 = time.perf_counter()
print("Le nombre d'itération est")
print(gradproj(10000,0.5,0.00001,gn,20))
tps4 = time.perf_counter()
print("le temps d'execution pour n=20")
print(tps4-tps3)
```

```
Le nombre d'itération est
109
le temps d'execution pour n=20
0.008421400000088397
```

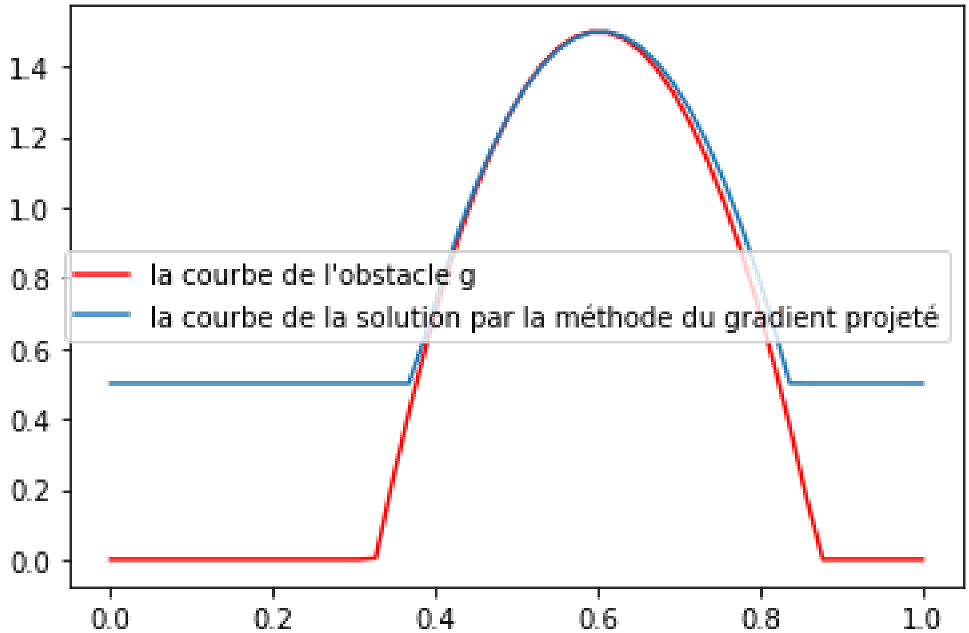
[68]:
```
graphe()
```



[45]:
```python
tps5 = time.perf_counter()
print("Le nombre d'itération est")
print(gradproj(10000,0.5,0.00001,gn,50))
tps6 = time.perf_counter()
print("le temps d'execution pour n=50")
print(tps6-tps5)
```

```
Le nombre d'itération est
86
le temps d'execution pour n=50
0.009168600000066363
```

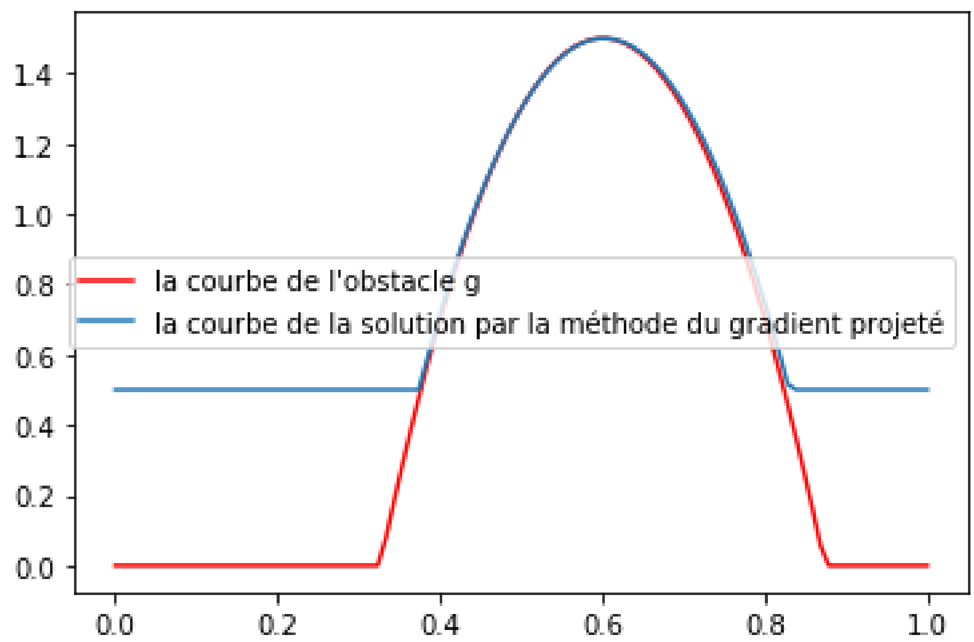
[77]:
```python
graphe()
```



[49]:
```
tps7 = time.perf_counter()
print("Le nombre d'itération")
print(gradproj(10000,0.5,0.00001,gn,100))
tps8 = time.perf_counter()
print("le temps d'execution pour n=100")
print(tps8-tps7)
```

```
Le nombre d'itération
75
le temps d'execution pour n=100
0.013152499999932843
```

[79]:
```
graphe()
```

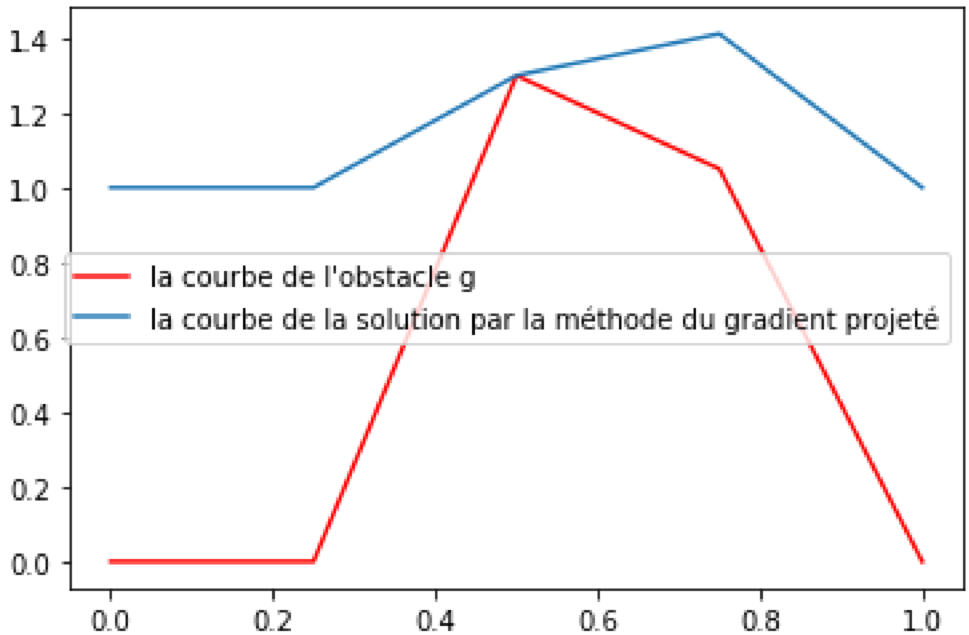


1.1.5 Pour $\rho = 1$

[40]:
```
tps1 = time.perf_counter()
print("Le nombre d'itération est")
print(gradproj(10000,1,0.00001,gn,5))
tps2 = time.perf_counter()
print("le temps d'execution pour n=5")
print(tps2-tps1)
```

```
Le nombre d'itération est
162
le temps d'execution pour n=5
0.0075322000000142
```

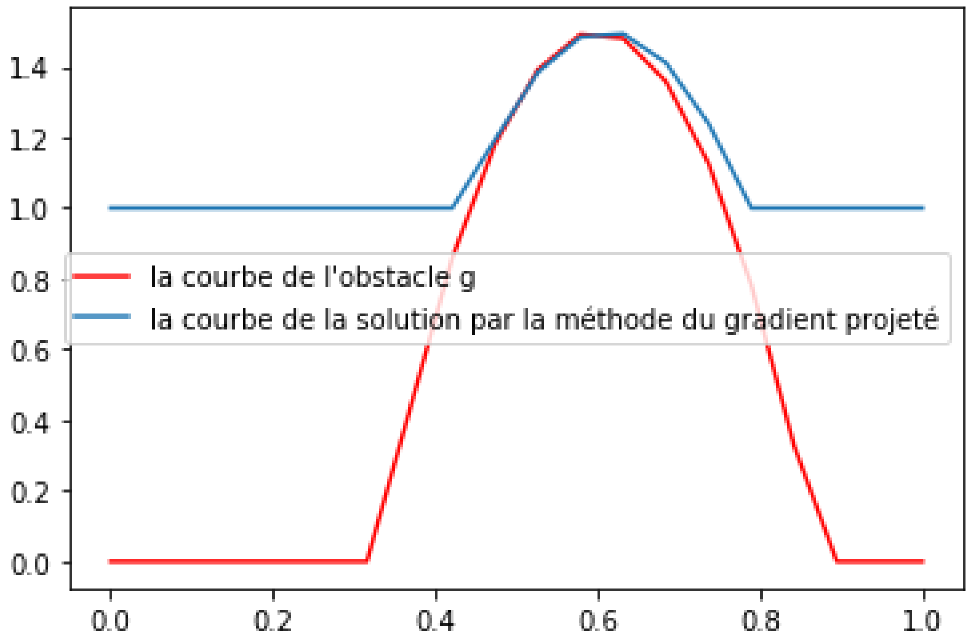
[83]:
```
graphe()
```



[43]:
```
tps3 = time.perf_counter()
print("Le nombre d'itération est")
print(gradproj(10000,1,0.00001,gn,20))
tps4 = time.perf_counter()
print("le temps d'execution pour n=20")
print(tps4-tps3)
```

```
Le nombre d'itération est
100
le temps d'execution pour n=20
0.008256500000015876
```

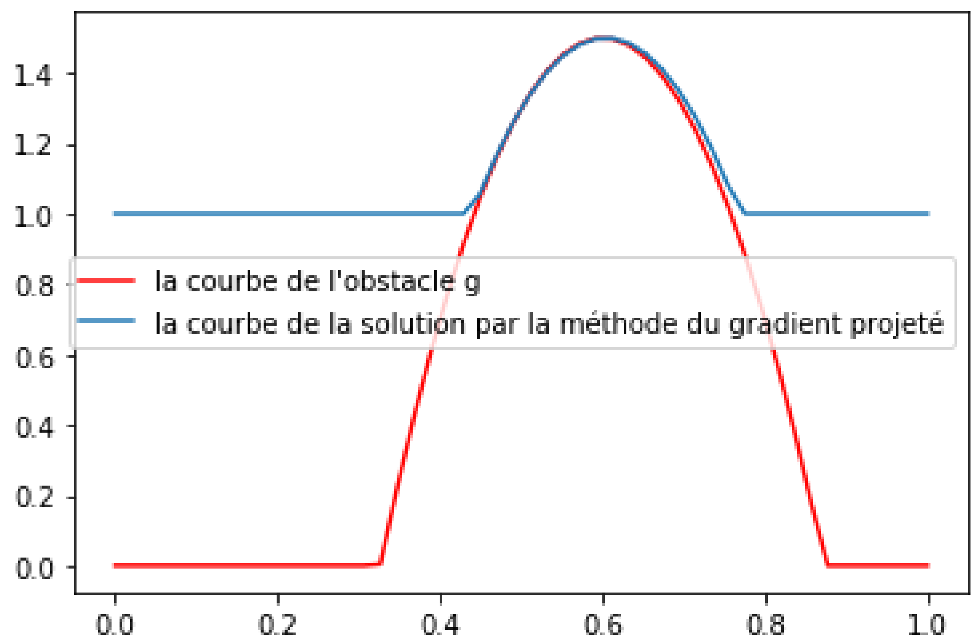
[85]:
```
graphe()
```



[46]:
```
tps5 = time.perf_counter()
print("Le nombre d'itération est")
print(gradproj(10000,1,0.00001,gn,50))
tps6 = time.perf_counter()
print("le temps d'execution pour n=50")
print(tps6-tps5)
```

```
Le nombre d'itération est
80
le temps d'execution pour n=50
0.008817899999939982
```

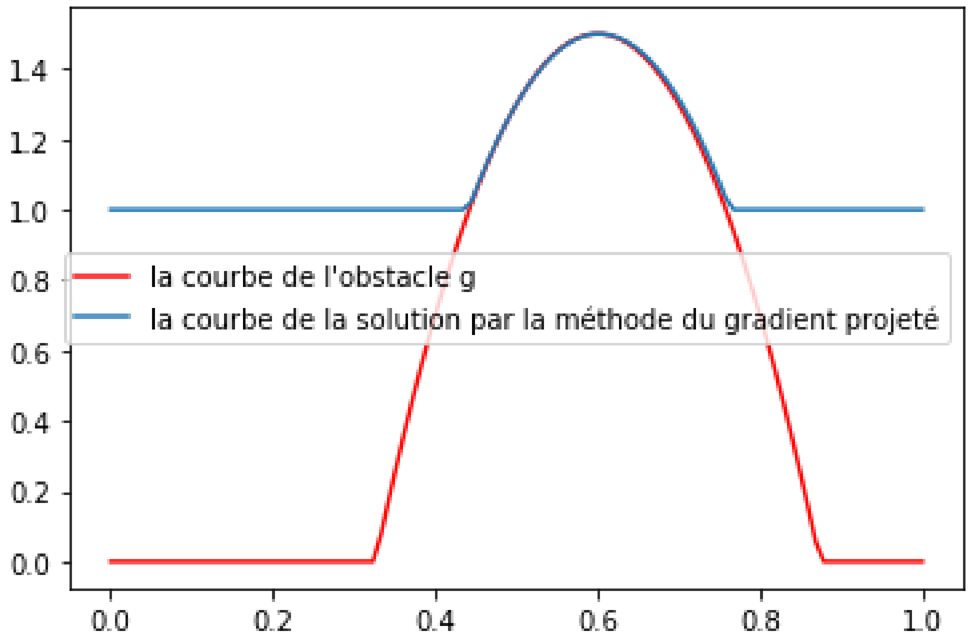
[87]:
```
graphe()
```



[48]:
```
tps7 = time.perf_counter()
print("Le nombre d'itération")
print(gradproj(10000,1,0.00001,gn,100))
tps8 = time.perf_counter()
print("le temps d'execution pour n=100")
print(tps8-tps7)
```

```
Le nombre d'itération
69
le temps d'execution pour n=100
0.011938699999973323
```

[89]:
```
graphe()
```



1.1.6 2.8 **Reprendre les questions** 2.5 **et** 2.6 **avec** ρ **optimal c'est-à-dire**

$$\rho = \frac{2}{\lambda_1 + \lambda_n}$$

où λ_1 et λ_n sont respectivement la plus petite et la plus grande valeur propre de A (associée au gradient de J_n).

[44]:
```
L=np.linalg.eigvals(A)
```

[45]:
```
L
L1=list(L)
```

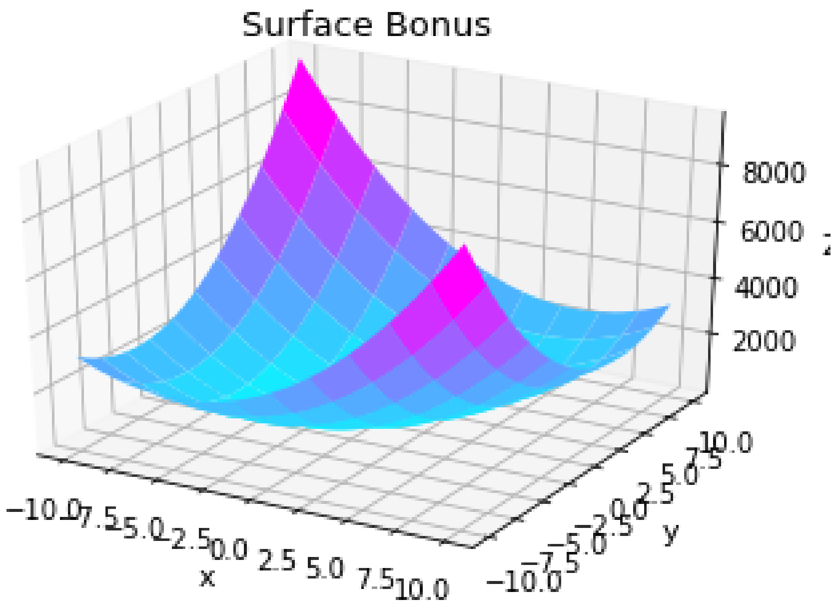
[46]:
```
L1=sort(L1)
```

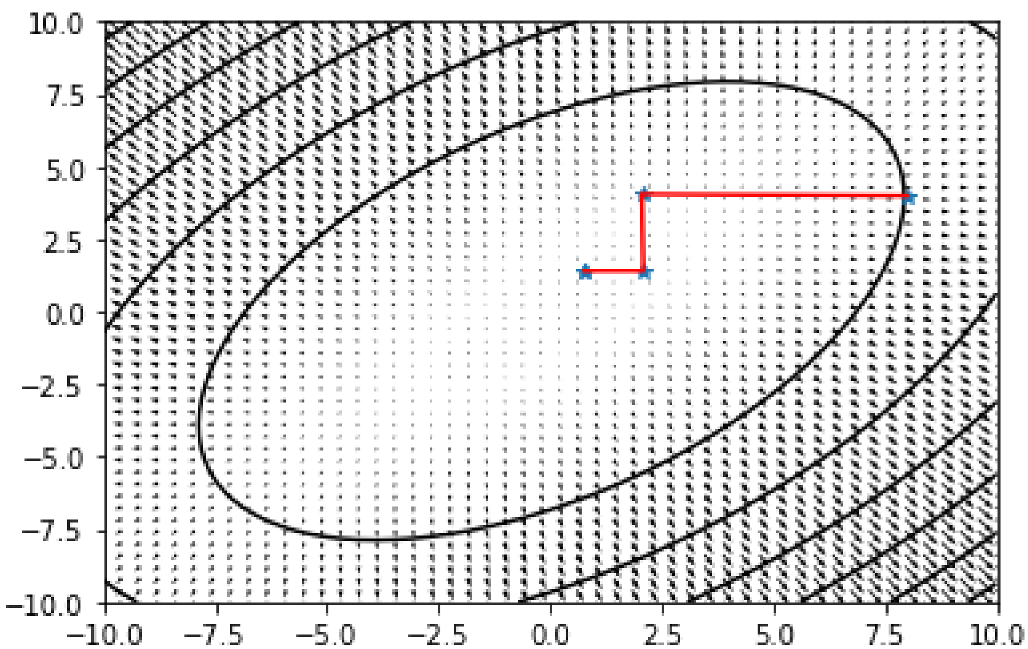
[47]:
```
w=2/(L1[0]+L1[-1])
```

[66]:
```
gradproj(100,w,0.00001,gn,n)
```

[66]:
```
array([[0.76805556],
       [1.41111111]])
```

[17]:
```
tracer(50,2)
```





[102]:
```
tps1 = time.perf_counter()
print("Le nombre d'itération est")
print(gradproj(10000,w,0.00001,gn,5))
tps2 = time.perf_counter()
print("le temps d'execution pour n=5")
```

```python
print(tps2-tps1)
```

```
Le nombre d'itération est
18
le temps d'execution pour n=5
0.0043009999999412685
```

[112]:
```python
tps3 = time.perf_counter()
print("Le nombre d'itération est")
print(gradproj(10000,w,0.00001,gn,20))
tps4 = time.perf_counter()
print("le temps d'execution pour n=20")
print(tps4-tps3)
```

```
Le nombre d'itération est
215
le temps d'execution pour n=20
0.015640499999790336
```

[117]:
```python
tps5 = time.perf_counter()
print("Le nombre d'itération est")
print(gradproj(10000,w,0.00001,gn,50))
tps6 = time.perf_counter()
print("le temps d'execution pour n=50")
print(tps6-tps5)
```

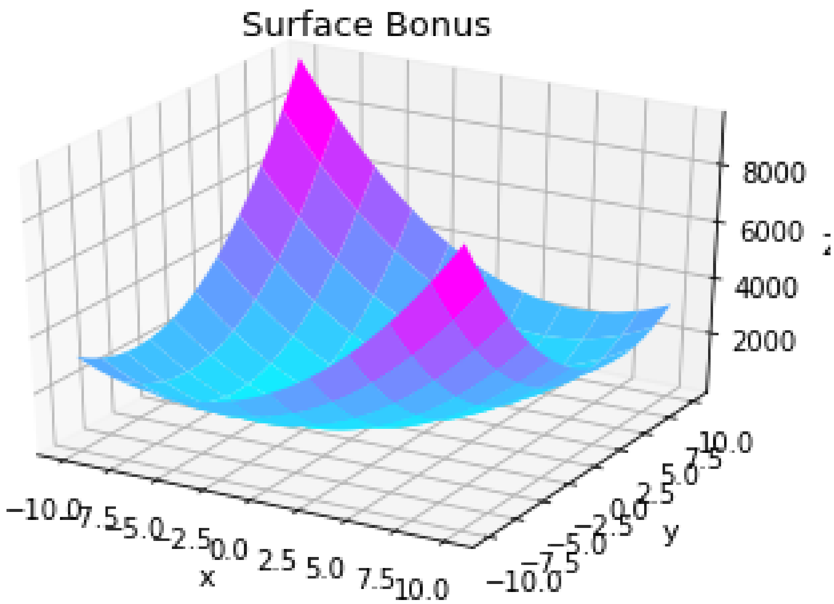
```
Le nombre d'itération est
1074
le temps d'execution pour n=50
0.0903757000000951
```

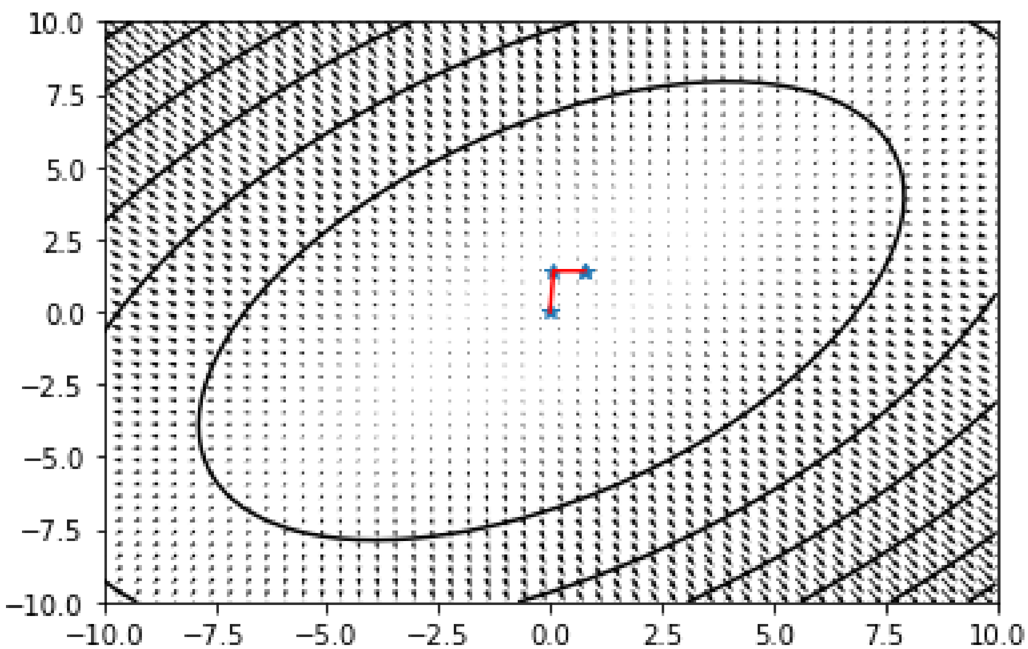
[122]:
```python
tps7 = time.perf_counter()
print("Le nombre d'itération")
print(gradproj(10000,w,0.00001,gn,100))
tps8 = time.perf_counter()
print("le temps d'execution pour n=100")
print(tps8-tps7)
```

```
Le nombre d'itération
3815
le temps d'execution pour n=100
0.4889047000001483
```

1.1.7 2.9 Reprendre la question précédente pour $f(x) = \pi^2 \sin(\pi x)$.

[19]:
```python
tracer(50,2)
```



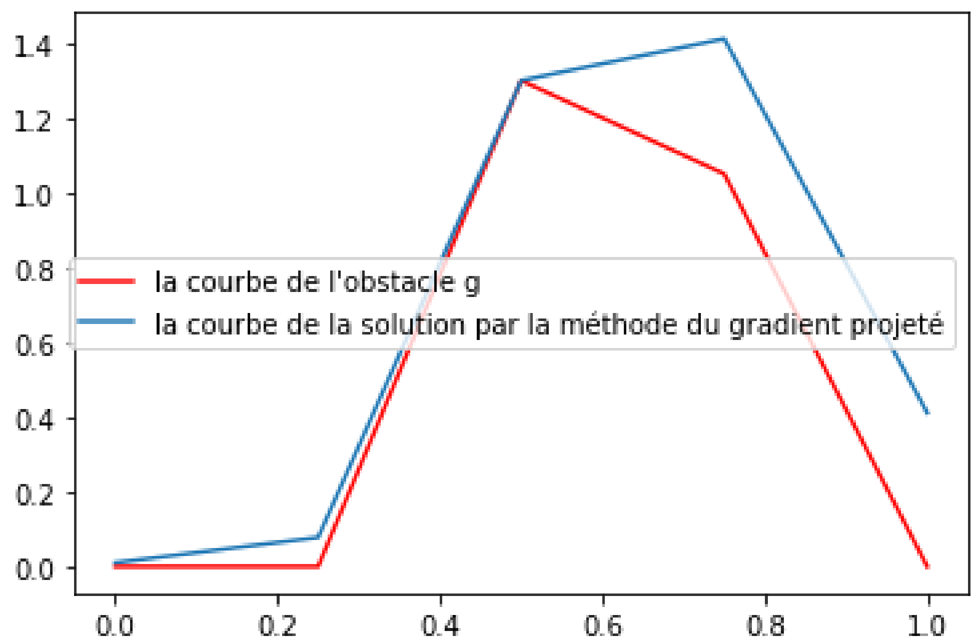


[24]:
```
tps1 = time.perf_counter()
print("Le nombre d'itération est")
print(gradproj(10000,w,0.00001,gn,5))
tps2 = time.perf_counter()
print("le temps d'execution pour n=5")
```

```
print(tps2-tps1)
```

```
Le nombre d'itération est
18
le temps d'execution pour n=5
0.0011853999999971165
```

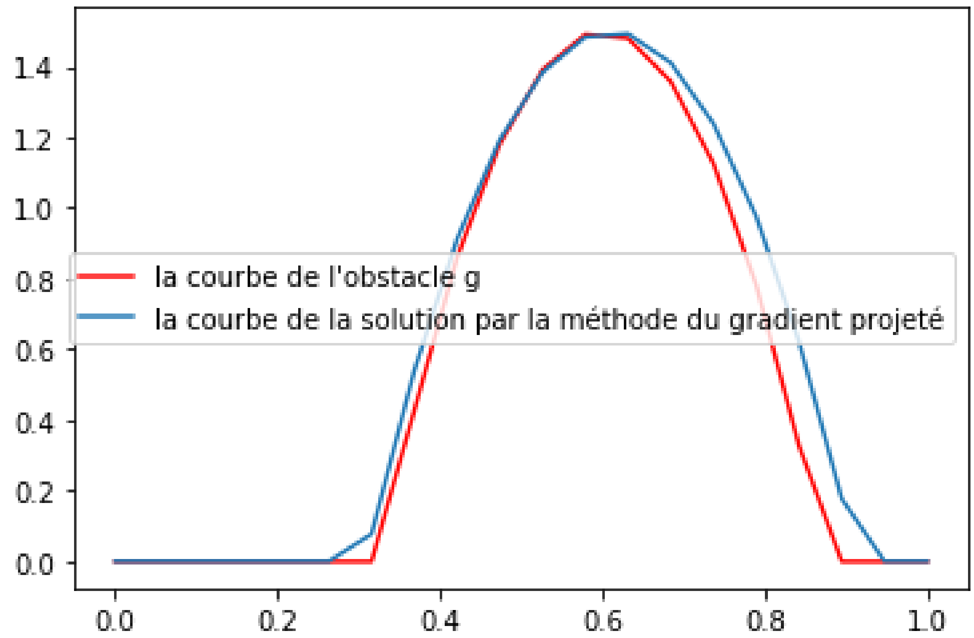
[50]:
```
graphe()
```



[30]:
```
tps3 = time.perf_counter()
print("Le nombre d'itération est")
print(gradproj(10000,w,0.00001,gn,20))
tps4 = time.perf_counter()
print("le temps d'execution pour n=20")
print(tps4-tps3)
```

```
Le nombre d'itération est
215
le temps d'execution pour n=20
0.012777500000026976
```

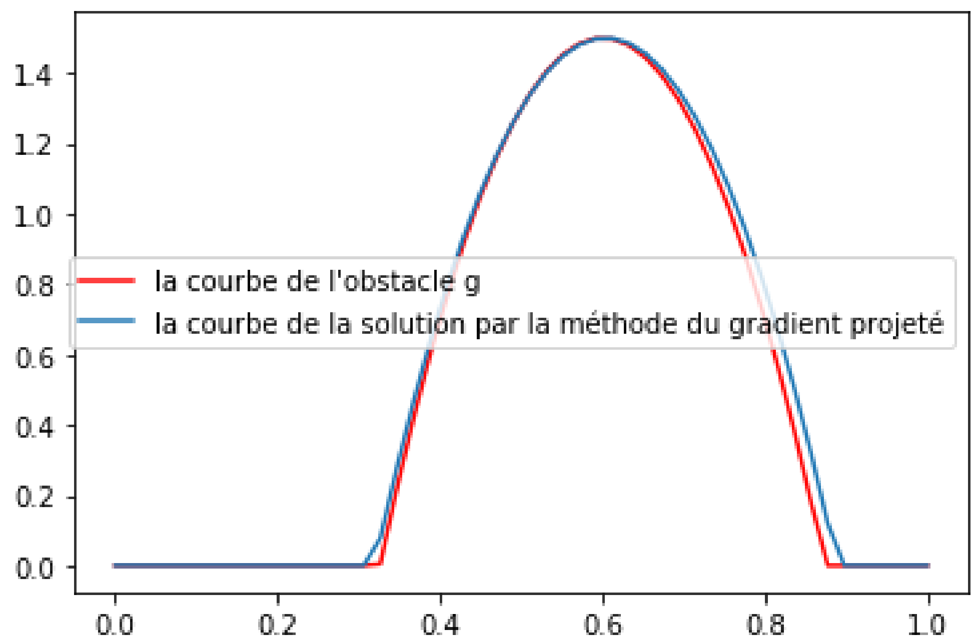
[26]:
```
graphe()
```



[37]:
```
tps5 = time.perf_counter()
print("Le nombre d'itération est")
print(gradproj(10000,w,0.00001,gn,50))
tps6 = time.perf_counter()
print("le temps d'execution pour n=50")
print(tps6-tps5)
```

```
Le nombre d'itération est
1074
le temps d'execution pour n=50
0.07908469999995305
```

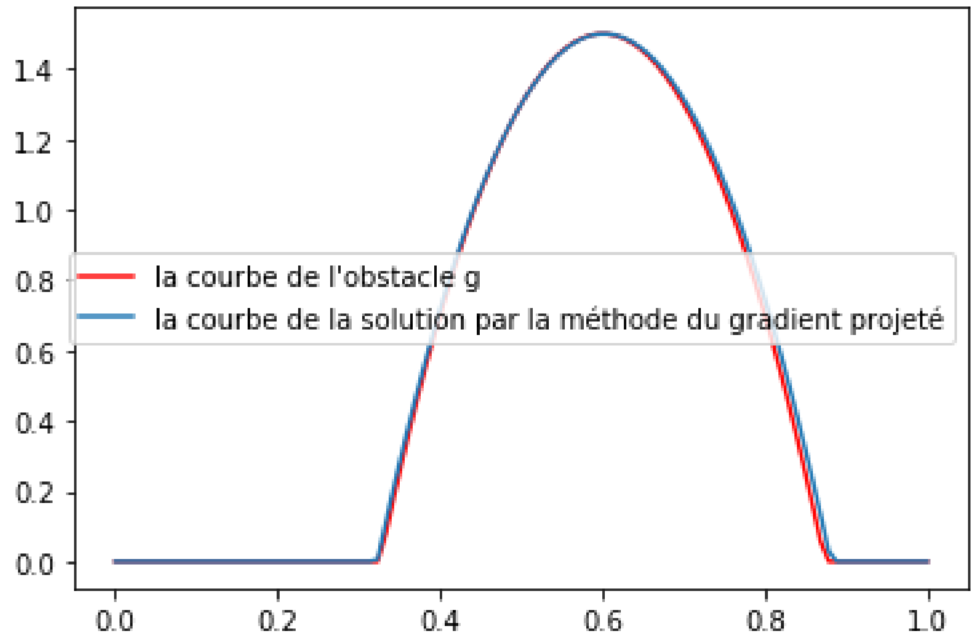
[34]:
```
graphe()
```



[43]:
```
tps7 = time.perf_counter()
print("Le nombre d'itération")
print(gradproj(10000,w,0.00001,gn,100))
tps8 = time.perf_counter()
print("le temps d'execution pour n=100")
print(tps8-tps7)
```

```
Le nombre d'itération
3815
le temps d'execution pour n=100
0.5021127999999635
```

[42]:
```
graphe()
```


1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
la courbe de l'obstacle g
la courbe de la solution par la méthode du gradient projeté