École Polytechnique Tunisie

Compte rendu TP 2 3 Optimisation

Optimisation sous contraintes

Élèves :
HAMMEMI Nassim
ALLEGUI Mohamed Yassin

Enseignant :
N. Chouaieb M. Moakher

21 mai 2021

TP 2 : Optimisation sous contraintes

May 19, 2021

1 Partie 1 : Discrétisation et résolution directe

1.1 partie théorique

- 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, . . . , n , où x i = ih avec h = 1/(n + 1) . Puis, on approche la solution u(x) par une fonction uh (x) , continue, affine par morceaux et donnée par:

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

avec :

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

on a:

On a,

$$\begin{aligned} J(u_h) &= \int_0^1 \frac{1}{2} (\sum_{i=1}^{n} u_i \Phi_i'(x))^2 dx - \int_0^1 f(x) \sum_{i=1}^{n} u_i \Phi_i(x) dx \\ &= \int_0^1 \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} u_i u_j \Phi_i'(x) \Phi_j'(x) dx - \int_0^1 f(x) \sum_{i=1}^{n} u_i \Phi_i(x) dx \\ &= \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} u_i u_j \int_0^1 \Phi_i'(x) \Phi_j'(x) dx - \sum_{i=1}^{n} u_i \int_0^1 f(x) \Phi_i(x) dx \\ &= \frac{1}{2} \langle Au, u \rangle - \langle b, u \rangle \end{aligned}$$

avec :

$$A = \begin{bmatrix} 2 & -1 & 0 & \dots & 0 \\ -1 & 2 & -1 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & -1 & 2 & -1 \\ 0 & \dots & 0 & -1 & 2 \end{bmatrix}, \quad u = \begin{pmatrix} u_1 \\ \vdots \\ \vdots \\ \vdots \\ u_n \end{pmatrix} \quad et \quad b = \begin{pmatrix} f(x_1) \\ \vdots \\ \vdots \\ \vdots \\ f(x_n) \end{pmatrix}$$

En effet,

$$A_{ij} = \int_0^1 \Phi_i'(x)\Phi_j'(x)dx$$

1er cas : i = j

$$\begin{aligned} A_{ii} &= \int_0^1 \Phi_i'(x)^2 dx \\ &= \int_{x_{i-1}}^{x_i} \frac{1}{h^2} dx + \int_{x_i}^{x_{i+1}} \frac{1}{h^2} dx \\ &= \frac{x_i - x_{i-1}}{h^2} + \frac{x_{i+1} - x_i}{h^2} \\ &= \frac{2}{h} \end{aligned}$$

2e cas : j = i+1 ou i = j+1

$$\begin{aligned} A_{ij} &= \int_0^1 \Phi_i'(x)\Phi_{i+1}'(x)dx \\ &= \int_{x_{i-1}}^{x_i} \frac{1}{h} * 0 \; dx + \int_{x_i}^{x_{i+1}} \frac{-1}{h} * \frac{1}{h} \; dx \\ &= \frac{-1}{h} \end{aligned}$$

3e cas : i et j quelconques

$$\begin{aligned} A_{ij} &= \int_{x_{i-1}}^{x_i} \frac{1}{h} * 0 \; dx + \int_{x_i}^{x_{i+1}} \frac{-1}{h} * 0 \; dx + \int_{x_{j-1}}^{x_j} \frac{1}{h} * 0 \; dx + \int_{x_j}^{x_{j+1}} \frac{-1}{h} * 0 \; dx \\ &= 0 \end{aligned}$$

On utilise la méthode du Trapèze pour le calcul de b,

$$\begin{aligned} b &= \int_0^1 f(x)\Phi_i(x)dx \\ &= \int_{x_{i-1}}^{x_i} f(x)\phi_i(x) \; dx + \int_{x_i}^{x_{i+1}} f(x)\phi_i(x) \; dx \\ &= \frac{x_i - x_{i-1}}{2}(f(x_i)\phi_i(x_i) + f(x_{i-1})\phi_i(x_{i-1})) \\ &\quad + \frac{x_{i+1} - x_i}{2}(f(x_i)\phi_i(x_i) + f(x_{i+1})\phi_i(x_{i+1})) \\ &= \frac{x_{i+1} - x_{i-1}}{2} f(x_i)\phi_i(x_i) \\ &= hf(x_i) \end{aligned}$$

Ainsi, on montrer que le probleme peut se mettre sous la forme suivante :

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

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

avec:

$$J_n(u) = \frac{1}{2}\langle Au, u\rangle - \langle b, u\rangle$$

- Le probléme est un probléme quadratique et Comme A est *Symtrique* , de plus leur déterminant est strictement positif,donc elle est symétrique définie positive .

-Donc le problème (3) admet une unique solution.

1.2 Implémentation des fonctionnelles Jn et ∇Jn

```
In [23]: #Implémenter les fonctionnelles à travers les deux fonctions Pyth
         # J(U, x, fx) et DJ(U, x, fx)

         #''' Definir les fonctions f , g , J et DJ '''


         def j(u,x,fx):

                 n=u.size

             h=x[1]-x[0]

             A=np.zeros((n,n))
             for i in range(n):
                 for j in range(n):
```

```
            if i==j:
                A[i,i]=2/h
                if j-1 >=0:
                    A[i,j-1]=-1/h
                if j+1 <=n-1:
                    A[i,j+1]=-1/h



    b=h*fx
    r=0.5*np.vdot(np.dot(A,u),u)-np.vdot(b,u)

    return(r)


def deltaJ(u,x,fx):
    n=u.size

    h=x[1]-x[0]

    A=np.zeros((n,n))
    for i in range(n):
        for j in range(n):
            if i==j:
                A[i,i]=2/h
                if j-1 >=0:
                    A[i,j-1]=-1/h
                if j+1 <=n-1:
                    A[i,j+1]=-1/h



    b=h*fx

    return np.dot(A,u)-b
```

1.3 Résolution du notre problème pour f(x) = 1

In [24]:
```
# 1èr Test pour f(x) =1

        def f1(x):
            return  1

        def g1(x):

                a=1.5 - 20*(x - 0.6)**2

                if a<0:
                    return  0
                else:
                    return a

        f=np.vectorize(f1)
        g=np.vectorize(g1)

        n = 100
        x = np . linspace (0 ,1 , n + 2 )

        xv = x [ 1 : - 1 ]
        fv , gv = f ( xv ) , g ( xv )

        Jf = lambda u : j(u,xv,fv)

        DJf = lambda u : deltaJ(u,xv,fv)

        const = ( { 'type': 'ineq' , 'fun' : lambda u : u - gv ,
        'jac' : lambda u : np.eye (np.size(u))})

        u = np.zeros (n)

        res = minimize ( Jf , u , method = 'SLSQP' , jac = DJf , constrai
        tol = 1e-8 , options = { 'xtol': 1e-8 , 'disp': True , 'maxiter':
```

```
s1=res.x
s1
```

```
/home/nassim/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:3
```

```
Optimization terminated successfully.    (Exit mode 0)
            Current function value: 2.4832390710768317
            Iterations: 113
            Function evaluations: 315
            Gradient evaluations: 113
```

```
Out[24]: array([0.02437865, 0.04865925, 0.07284199, 0.09692671, 0.12091339
                0.14480227, 0.16859319, 0.19228596, 0.21588064, 0.23937737
                0.26277605, 0.2860767 , 0.30927922, 0.33238385, 0.35539043
                0.37829902, 0.40110972, 0.42382236, 0.4464369 , 0.46895345
                0.49137198, 0.51369238, 0.53591481, 0.55803925, 0.58006544
                0.60199356, 0.62382364, 0.64555549, 0.66718947, 0.68872543
                0.71016339, 0.73150318, 0.75274506, 0.77388912, 0.79493503
                0.81588265, 0.83673228, 0.85748398, 0.87813764, 0.89869331
                0.91915078, 0.93951004, 0.95977132, 0.97993471, 1.
                1.00147003, 1.00284272, 1.00411732, 1.00529377, 1.00637203
                1.00735225, 1.00823457, 1.00901898, 1.0097053 , 1.01029365
                1.01078379, 1.01117627, 1.01147068, 1.01166689, 1.01176544
                1.01176565, 1.01166775, 1.01147186, 1.01117749, 1.01078514
                1.01029511, 1.00970589, 1.00901902, 1.00823416, 1.00735161
                1.00637103, 1.00529298, 1.00411759, 1.00284374, 1.00147143
                1.        , 0.96117634, 0.92225449, 0.88323476, 0.84411687
                0.80490078, 0.76558685, 0.72617492, 0.68666501, 0.64705704
                0.607351  , 0.56754721, 0.52764547, 0.48764551, 0.44754741
                0.40735137, 0.36705729, 0.32666514, 0.28617481, 0.24558663
                0.20490045, 0.16411625, 0.12323425, 0.0822542 , 0.04117603
```

1.4 Résolution du notre problème pour $f(x) = \pi^2 \sin(\pi x)$

```
In [63]: # 2ème Test )

        def f1(x):
            return  np.pi**2*np.sin(np.pi*x)

        def g1(x):

                a=1.5 - 20*(x - 0.6)**2

                if a<0:
                    return  0
                else:
                    return a

        f=np.vectorize(f1)
        g=np.vectorize(g1)

        n = 100
        x = np . linspace (0 ,1 , n + 2 )

        xv = x [ 1 : - 1 ]
        fv , gv = f ( xv ) , g ( xv )

        Jf = lambda u : j(u,xv,fv)

        DJf = lambda u : deltaJ(u,xv,fv)

        const = ( { 'type': 'ineq' , 'fun' : lambda u : u - gv ,
        'jac' : lambda u : np.eye (np.size(u))})

        u = np.zeros (n)

        res = minimize ( Jf , u , method = 'SLSQP' , jac = DJf , constrai
        tol = 1e-8 , options = { 'xtol': 1e-8 , 'disp': True , 'maxiter':
```

```
s2=res.x
s2
```

```
/home/nassim/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:3
```

```
Optimization terminated successfully.    (Exit mode 0)
            Current function value: -2.2286371779034995
            Iterations: 107
            Function evaluations: 308
            Gradient evaluations: 107
```

```
Out[63]: array([0.03502798, 0.0700259 , 0.10496374, 0.13981145, 0.17453914
                0.20911696, 0.24351526, 0.27770453, 0.31165556, 0.34533926
                0.37872683, 0.41178979, 0.44449997, 0.4768295 , 0.50875089
                0.54023707, 0.57126138, 0.60179763, 0.63182004, 0.6613034
                0.69022289, 0.71855443, 0.74627435, 0.77335961, 0.79978786
                0.82553728, 0.85058678, 0.87491594, 0.89850502, 0.92133498
                0.94338754, 0.96464516, 0.98509107, 1.00470929, 1.02348467
                1.04140276, 1.05845007, 1.07461393, 1.08988249, 1.10424476
                1.11769065, 1.13021097, 1.14179741, 1.15244241, 1.16213989
                1.17088377, 1.17866972, 1.18549388, 1.19135347, 1.19624662
                1.20017239, 1.20313077, 1.20512267, 1.20614999, 1.20621555
                1.20532307, 1.2034772 , 1.20068355, 1.19694858, 1.19227971
                1.18668526, 1.18017441, 1.17275728, 1.16444488, 1.155249
                1.14518232, 1.13425837, 1.12249157, 1.10989704, 1.09649084
                1.08228973, 1.06731121, 1.0515736 , 1.0350959 , 1.01789785
                1.        , 0.96556315, 0.93046952, 0.89474113, 0.8584011
                0.8214731 , 0.78398133, 0.74595048, 0.70740582, 0.66837304
                0.62887838, 0.58894853, 0.54861051, 0.50789185, 0.4668204
                0.42542432, 0.38373211, 0.34177262, 0.29957486, 0.2571681
                0.21458181, 0.17184569, 0.12898948, 0.08604312, 0.04303661]
```

1.5 Vérification des conditions pour la solution (1)

```
In [26]: print("U(0) = ", round(s1[0]))
         print("U(1) = ", round(s1[99]))
```

```
U(0) =  0.0
U(1) =  0.0
```

```
In [27]: s1>gv
```

```
Out[27]: array([ True,  True,  True,  True,  True,  True,  True,  True,  T
                 True,  True,  True,  True,  True,  True,  True,  True,  T
                 True,  True,  True,  True,  True,  True,  True,  True,  T
                 True,  True,  True,  True,  True,  True,  True,  True,  T
                 True,  True,  True,  True,  True,  True,  True,  True,  T
                 True,  True,  True,  True,  True,  True,  True,  True,  T
                 True,  True,  True,  True,  True,  True,  True,  True,  T
                 True,  True,  True,  True,  True,  True,  True,  True,  T
                 True,  True,  True,  True,  True,  True,  True,  True,  T
                 True,  True,  True,  True,  True,  True,  True,  True,  T
                 True])
```

*

on remarque donc que U(x) > g(x), donc la solution numérique vérifie la condition 4.b

•

```
In [62]: # calcule de U"(x)
         n=100
         # préparation des tableaux qui vont recevoir les valeurs

         xnew = np.zeros(n-1)
         yp = np.zeros(n-1)

         # calcul des abscisses et des valeurs de la dérivée 1
         for i in range(n-1):
             xnew[i] = (x[i] + x[i+1]) / 2
             yp[i] = (s1[i+1] - s1[i]) / (x[i+1] - x[i])

         # préparation des tableaux qui vont recevoir les valeurs
```

```
xnew1 = np.zeros(n-2)
yp1 = np.zeros(n-2)

# calcul des abscisses et des valeurs de la dérivée 2
for i in range(n-2):
    xnew1[i] = (xnew[i] + xnew[i+1]) / 2
    yp1[i] = (yp[i+1] - yp[i]) / (xnew[i+1] - xnew[i])

# vérificatio que u" (x) ≥ f (x)

yp1 >= fv[:98]
```

```
Out[62]: array([False,  True, False,  True, False,  True, False,  True, Fal
                True, False,  True, False,  True, False,  True, False,  Tr
               False,  True, False,  True, False,  True, False,  True, Fal
                True, False,  True, False,  True, False,  True, False,  Tr
               False,  True, False,  True, False,  True, False,  True, Fal
                True, False,  True, False,  True, False,  True, False,  Tr
               False,  True, False,  True, False,  True, False,  True, Fal
                True, False,  True, False,  True, False,  True, False,  Tr
               False,  True, False,  True, False,  True, False,  True, Fal
                True, False,  True, False,  True, False,  True, False,  Tr
               False,  True, False,  True, False,  True, False,  True])
```

la solution numérique ne vérife pas la condition 4.a

-

```
In [56]: (yp1 - fv[0:98])* (s1[0:98] - gv[0:98]) ==0
```

```
/home/nassim/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:1
  """Entry point for launching an IPython kernel.
```

```
Out[56]: array([False, False, False, False, False, False, False, False, Fal
                False, False, False, False, False, False, False, False, Fal
                False, False, False, False, False, False, False, False, Fal
                False, False, False, False, False, False, False, False, Fal
```

```
       False, False, False, False, False, False, False, False, Fa
       False, False, False, False, False, False, False, False, Fa
       False, False, False, False, False, False, False, False, Fa
       False, False, False, False, False, False, False, False, Fa
       False, False, False, False, False, False, False, False, Fa
       False, False, False, False, False, False, False, False, Fa
       False, False, False, False, False, False, False, False])
```

donc la solution numérique ne vérifie pas la condition 4.c

1.6 Augmentation de la nombre n :

```
In [70]: # pour f(x) =1

         def f1(x):
             return  1

         def g1(x):

                 a=1.5 - 20*(x - 0.6)**2

                 if a<0:
                     return  0
                 else:
                     return a

         f=np.vectorize(f1)
         g=np.vectorize(g1)

         n = 200
         x = np . linspace (0 ,1 , n + 2 )

         xv = x [ 1 : - 1 ]
         fv , gv = f ( xv ) , g ( xv )

         Jf = lambda u : j(u,xv,fv)
```

```
DJf = lambda u : deltaJ(u,xv,fv)

const = ( { 'type': 'ineq' , 'fun' : lambda u : u - gv ,
'jac' : lambda u : np.eye (np.size(u))})

u = np.zeros (n)

res = minimize ( Jf , u , method = 'SLSQP' , jac = DJf , constrai
tol = 1e-8 , options = { 'xtol': 1e-8 , 'disp': True , 'maxiter':

s1=res.x

print("1ère  Test\n ")

print("U(0) = ", round(s1[0]))
print("U(1) = ", round(s1[199]))

print("2ème  Test\n")
print(s1>gv)

#////////////////////

# calcule de U"(x)
n=200
# préparation des tableaux qui vont recevoir les valeurs

xnew = np.zeros(n-1)
yp = np.zeros(n-1)

# calcul des abscisses et des valeurs de la dérivée 1
for i in range(n-1):
    xnew[i] = (x[i] + x[i+1]) / 2
    yp[i] = (s1[i+1] - s1[i]) / (x[i+1] - x[i])

# préparation des tableaux qui vont recevoir les valeurs
```

```
        xnew1 = np.zeros(n-2)
        yp1 = np.zeros(n-2)

        # calcul des abscisses et des valeurs de la dérivée 2
        for i in range(n-2):
            xnew1[i] = (xnew[i] + xnew[i+1]) / 2
            yp1[i] = (yp[i+1] - yp[i]) / (xnew[i+1] - xnew[i])

        print("3ième Test\n")

        print(yp1 >= fv[:198])


        print("4ième Test \n")

        print((yp1-fv[:198])*(s1[:198] - gv[:198])==0)
```

```
/home/nassim/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:34


Optimization terminated successfully.    (Exit mode 0)
            Current function value: 2.518014678732643
            Iterations: 218
            Function evaluations: 727
            Gradient evaluations: 218
1ère  Test

U(0) =  0.0
U(1) =  0.0
2ème  Test

[ True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
```

```
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True]
3ième Test

[False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False]
4ième Test

[False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
```

```
False False False False False False False False False False False False
False False False False False False False False False False False False
False False False False False False False False False False False False
False False False False False False False False False False False False
False False False False False False False False False False False False
False False False False False False False False False False False False
False False False False False False False False False False False False
False False False False False False False False False False False False
False False False False False False False False False False False False
False False False False False False False False False False False False
False False False False False False False False False False False False
False False False False False False False False False False False False
False False False False False False False False False False False False
False False False False False False False False False False False False]
```

on faisant augmenter le nombre n de 100 vers 200 on ramarque que quelque condition sont vérifées :

__

__ U(0) = U(1) =0

__

__ U(x) > g(x)

**

2 Partie 2 : Méthode du gradient projeté

2.1 Ecrire un programme projK.py qui prend en argument un point u et g n = (g(x i)) i=1,. . . ,n et qui renvoie $P_k(u)$

```
In [110]: def projection(v,g) :
              Pk=np.maximum(v, g )
              return Pk
```

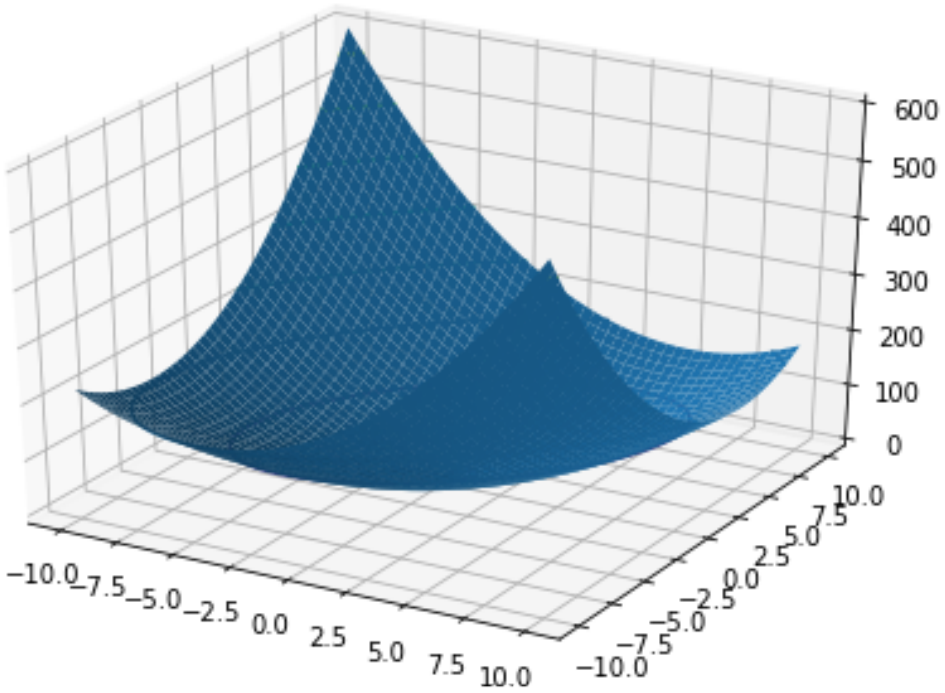
2.2 Implémentation

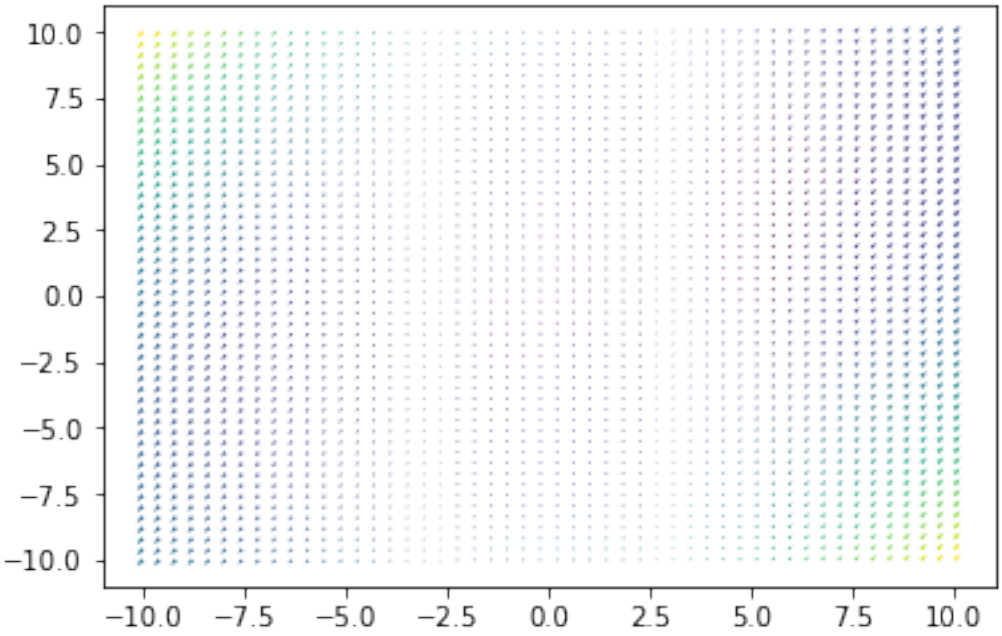
```
In [18]: #Méthode du gradient projeté à pas fixe

         def gradient_projete_fixe (J , DJ , gn , u0 , rho , Tol , iterMax , store ) :

             k=0
             r=Tol
             u=u0
             l=[]
             while (k < iterMax and r >= Tol ):
                 w=-DJ(u)
                 Pk=np.maximum(u+rho*w, gn )
                 v=u
                 u=Pk
                 r=np.linalg.norm(v-u)
                 k+=1
                 l.append(u)
             if (store==1):
                 return l
             else:
                 return l[-1]
```

2.3 Pour n = 2 , tracer sur une même figure les courbes de niveaux de J 2 ainsi que le champ de vecteurs ∇J_2 sur le pavé $[-10, 10] \times [-10; 10]$, calculer les itérations $u(k)$ = (u 1 , u 2) 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 $u(0) = (8, 4)$, ρ = 0.1 et Tol = 10 −

```
In [28]: import numpy as np
         import matplotlib.pyplot as plt
         from mpl_toolkits.mplot3d import Axes3D

         v1 = np.linspace(-10,10,50)
         v2 = np.linspace(-10,10,50)
```

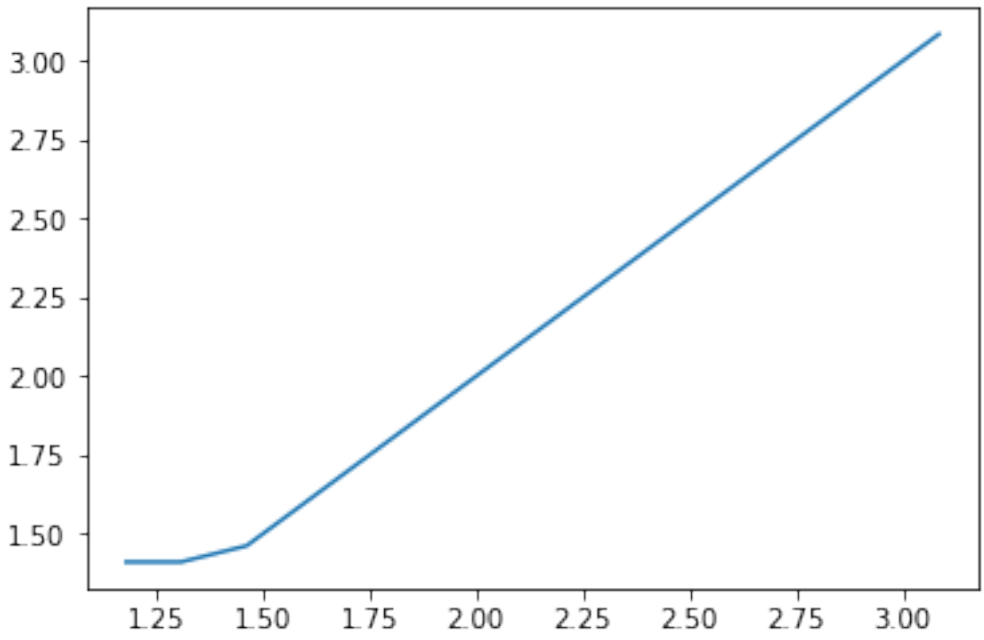
```
v,u = np.meshgrid(v1,v2)

z = v*(2*v-u) + u*(2*u-v) -u -v


#Pour le courbe en 3D:
fig = plt.figure()
ax = Axes3D(fig)
ax.plot_surface(v,u,z)

#Pour le contour
plt.contour(v,u,z)
plt.show()

plt.quiver(v,u,v2,v1,Z,scale=1500,units='width')

plt.show()
```





In [36]:
```
n = 2


x = np . linspace (0 ,1 , n + 2 )
xv = x [ 1 : - 1 ]
fv , gv = f ( xv ) , g ( xv )
Jf = lambda u : j(u,xv,fv)

DJf = lambda u : deltaJ(u,xv,fv)
u0=np.array(n*[4])

l1= gradient_projete_fixe(Jf, DJf,gv,u0,0.1,10^-5,100,1)

xx=[i[0] for i in l1]
yy=[i[1] for i in l1]

plt.plot(xx,yy)
```

Out[36]:
```
[<matplotlib.lines.Line2D at 0x7f5353707a90>]
```



2.4 Pour n = 5, 20, 50, 100 , affcher à l'aide de la fonction print le nombre d'itérations ainsi que le temps de calcul pour chaque n . Tracer sur une même figure les solutions approchées U_n , ainsi que le graphe de la fonction g . On prendra ρ = 0.1 , Tol = 10 −5

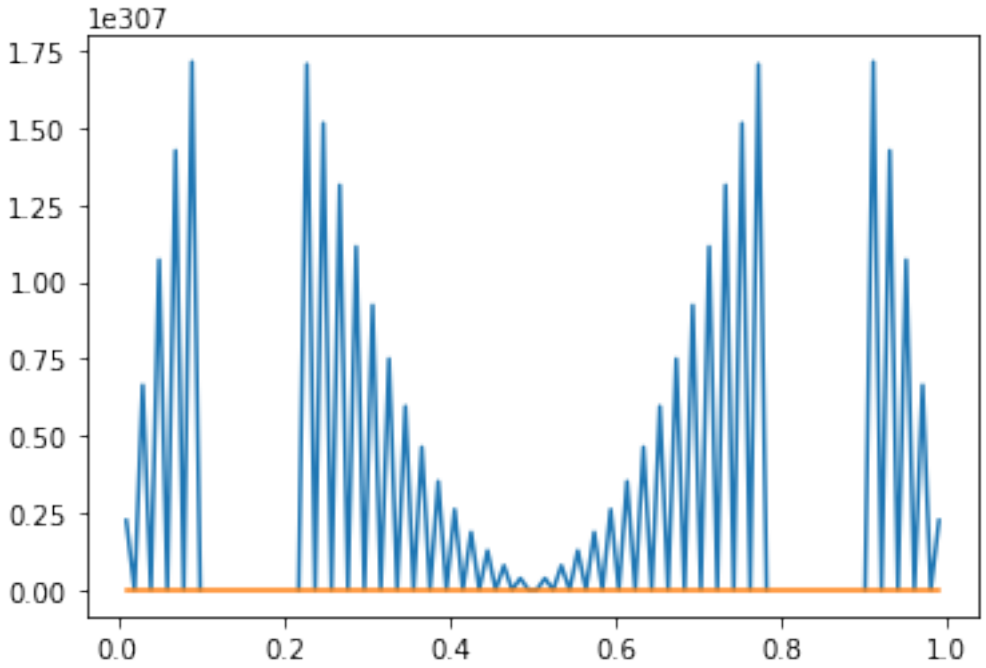
```
In [41]: import time
         from datetime import timedelta

         l=[5,20,50,100]
         for i in l:
             start_time = time.monotonic()

             n = i

             x = np . linspace (0 ,1 , n + 2 )

             xv = x [ 1 : - 1 ]
             fv , gv = f(xv) , g(xv)
             Jf = lambda u : j(u,xv,fv)

             DJf = lambda u : deltaJ(u,xv,fv)
             u0=np.array(n*[4])

             l1= gradient_projete_fixe(Jf, DJf,gv,u0,0.1,10^-5,200000,1)
```

```
            end_time = time.monotonic()

            print("le temps de calcul pour n = "+str(i)+"\n",timedelta(seconds=end_time - start

            print("le nombre d'itération pour n = "+str(i)+"\n",len(l1))
```

```
/home/nassim/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:11: RuntimeWarning: inv
  # This is added back by InteractiveShellApp.init_path()
```

```
le temps de calcul pour n = 5
 0:00:01.199728
le nombre d'itération pour n = 5
 18355
le temps de calcul pour n = 20
 0:00:00.082090
le nombre d'itération pour n = 20
 511
le temps de calcul pour n = 50
 0:00:00.099261
le nombre d'itération pour n = 50
 309
le temps de calcul pour n = 100
 0:00:00.322961
le nombre d'itération pour n = 100
 239
```

In [65]:
```
#Représentation de la solution u et la fonction g dans la mm graphe

        UU=l1[-2]
        xv =np.linspace (0 ,1 , n + 2 )[1:-1]

        plt.plot(xv,UU)
        plt.plot(xv,gv)
        plt.show()
```



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

In [66]:
```
#pour ρ = 0.5

import time
from datetime import timedelta

l=[5,20,50,100]
for i in l:
    start_time = time.monotonic()

    n = i

    x = np . linspace (0 ,1 , n + 2 )

    xv = x [ 1 : - 1 ]
    fv , gv = f(xv) , g(xv)
    Jf = lambda u : j(u,xv,fv)

    DJf = lambda u : deltaJ(u,xv,fv)
    u0=np.array(n*[4])

    l1= gradient_projete_fixe(Jf, DJf,gv,u0,0.5,10^-5,200000,1)
```

```
            end_time = time.monotonic()

            print("le temps de calcul pour n = "+str(i)+"\n",timedelta(seconds=end_time - start

            print("le nombre d'itération pour n = "+str(i)+"\n",len(l1))
```

```
le temps de calcul pour n = 5
 0:00:00.029258
le nombre d'itération pour n = 5
 432
le temps de calcul pour n = 20
 0:00:00.058722
le nombre d'itération pour n = 20
 239
le temps de calcul pour n = 50
 0:00:00.062975
le nombre d'itération pour n = 50
 184
le temps de calcul pour n = 100
 0:00:00.191133
le nombre d'itération pour n = 100
 157
```

In [89]:
```
#pour ρ = 1

import time
from datetime import timedelta

l=[5,20,50,100]
for i in l:
    start_time = time.monotonic()

    n = i

    x = np . linspace (0 ,1 , n + 2 )

    xv = x [ 1 : - 1 ]
    fv , gv = f(xv) , g(xv)
    Jf = lambda u : j(u,xv,fv)

    DJf = lambda u : deltaJ(u,xv,fv)
    u0=np.array(n*[4])

    l1= gradient_projete_fixe(Jf, DJf,gv,u0,1,10^-5,200000,1)
```

```
            end_time = time.monotonic()

            print("le temps de calcul pour n = "+str(i)+"\n",timedelta(seconds=end_time - start

            print("le nombre d'itération pour n = "+str(i)+"\n",len(l1))
```

```
le temps de calcul pour n = 5
 0:00:00.006494
le nombre d'itération pour n = 5
 79
le temps de calcul pour n = 20
 0:00:00.010029
le nombre d'itération pour n = 20
 69
le temps de calcul pour n = 50
 0:00:00.031326
le nombre d'itération pour n = 50
 64
le temps de calcul pour n = 100
 0:00:00.107235
le nombre d'itération pour n = 100
 61
```

```
/home/nassim/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:11: RuntimeWarning: ove
  # This is added back by InteractiveShellApp.init_path()
```

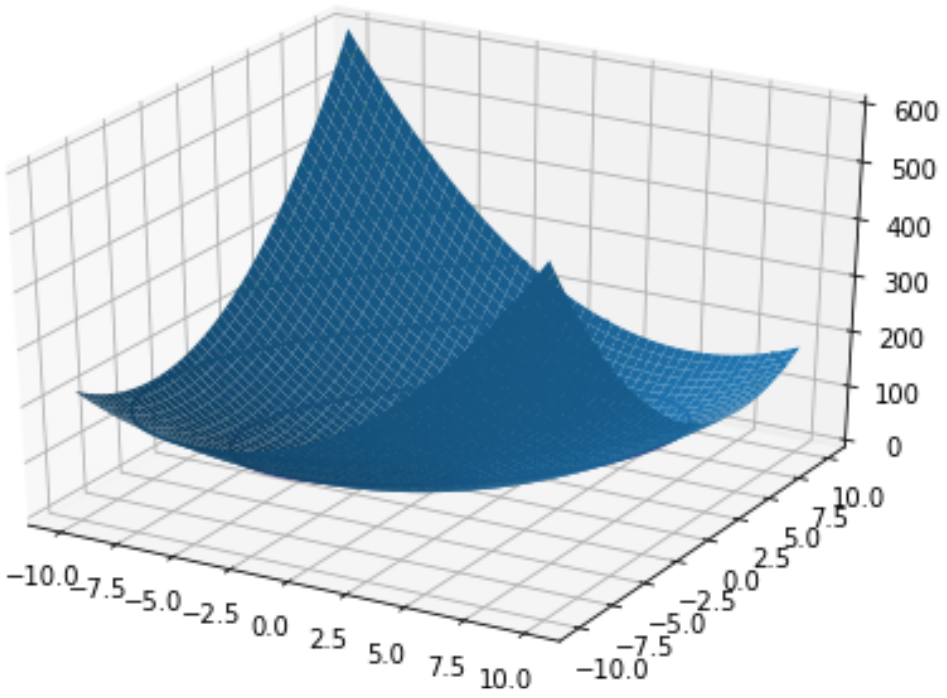
On constate que ,pour ρ =0.5 le temps de calcule ainsi que le nombre d'itération sot dimuniées aussi pou ρ =1 , ces deux dernier ont diminuées aussi .

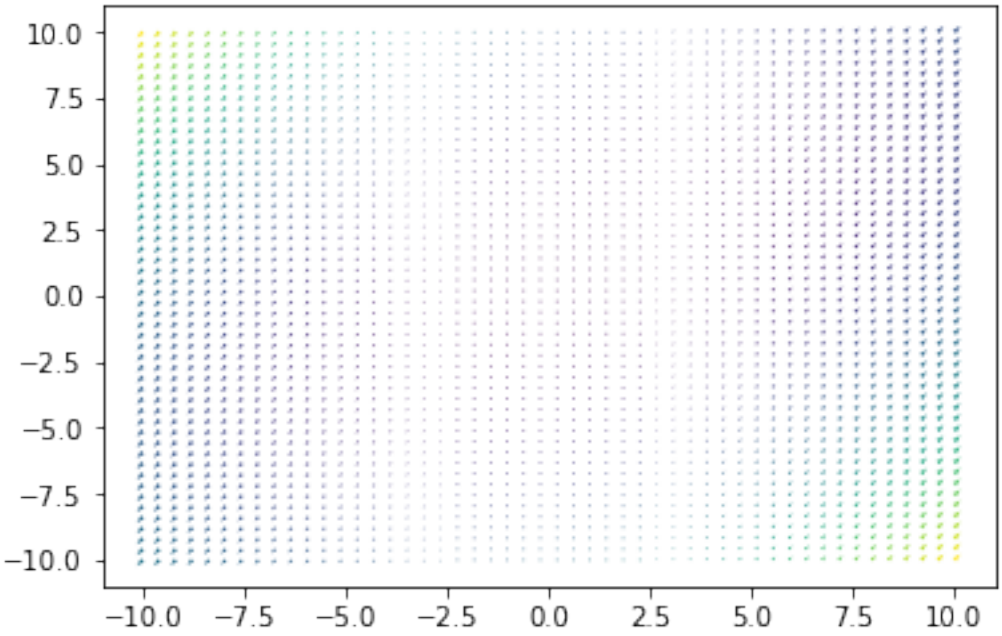
2.6 Reprendre les questions 2.5 et 2.6 avec ρ optimal :

```
In [109]: #Calcule du ρ optimale
          n=2

          h=x[1]-x[0]

          A=np.zeros((n,n))
          for i in range(n):

               for j in range(n):

                      if i==j:
                          A[i,i]=2/h
                          if j-1 >=0:
                              A[i,j-1]=-1/h
```

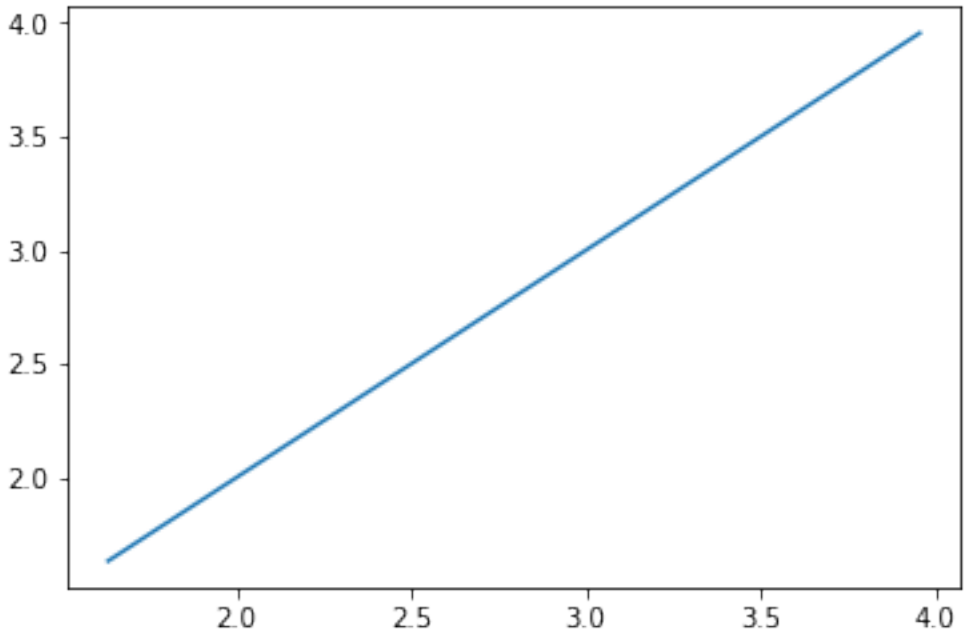
```
                if j+1 <=n-1:
                    A[i,j+1]=-1/h

l = np.linalg.eigvals(A)
l=np.sort(l)

rho=2/(l[0]+l[-1])
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

v1 = np.linspace(-10,10,50)
v2 = np.linspace(-10,10,50)

v,u = np.meshgrid(v1,v2)

z = v*(2*v-u) + u*(2*u-v) -u -v


#Pour le courbe en 3D:
fig = plt.figure()
ax = Axes3D(fig)
ax.plot_surface(v,u,z)

#Pour le contour
plt.contour(v,u,z)
plt.show()

plt.quiver(v,u,v2,v1,Z,scale=1500,units='width')

plt.show()

#######

# tracer sur la même figure que précédemment la ligne qui relie les u(k)

n = 2


x = np . linspace (0 ,1 , n + 2 )
xv = x [ 1 : - 1 ]
fv , gv = f ( xv ) , g ( xv )
Jf = lambda u : j(u,xv,fv)

DJf = lambda u : deltaJ(u,xv,fv)
u0=np.array(n*[4])

l1= gradient_projete_fixe(Jf, DJf,gv,u0,rho,10^-5,100,1)
```

```
xx=[i[0] for i in l1]
yy=[i[1] for i in l1]

plt.plot(xx,yy)
```





Out[109]: [<matplotlib.lines.Line2D at 0x7f535963aef0>]



In [113]:
```
#pour ρ = ρ_optimal
import numpy as np
```

```
import time
from datetime import timedelta

l=[5,20,50,100]
for i in l:
    start_time = time.monotonic()

    n = i

    x = np . linspace (0 ,1 , n + 2 )



    h=1/n

    A=np.zeros((n,n))

    for k in range(n):

       for j in range(n):

            if k==j:
                A[k,k]= 2/h
                if j-1 >=0:
                    A[k,j-1]= -1/h
                if j+1 <=n-1:
                    A[k,j+1]= -1/h


    l= np.linalg.eigvals(A)
    l=np.sort(l)

    rho=2/(l[0]+l[-1])



    xv = x [ 1 : - 1 ]
    fv , gv = f(xv) , g(xv)
    Jf = lambda u : j(u,xv,fv)

    DJf = lambda u : deltaJ(u,xv,fv)
    u0=np.array(n*[4])
    l1= gradient_projete_fixe(Jf, DJf,gv,u0,rho,10^-5,10000,1)

    end_time = time.monotonic()

    print("le temps de calcul pour n = "+str(i)+"\n",timedelta(seconds=end_time - star
```

```
            print("le nombre d'itération pour n = "+str(i)+"\n",len(l1))
```

```
le temps de calcul pour n = 5
 0:00:00.692931
le nombre d'itération pour n = 5
 10000
le temps de calcul pour n = 20
 0:00:01.341293
le nombre d'itération pour n = 20
 10000
le temps de calcul pour n = 50
 0:00:03.186838
le nombre d'itération pour n = 50
 10000
le temps de calcul pour n = 100
 0:00:11.604936
le nombre d'itération pour n = 100
 10000
```

On constate que pour n=5 par exemple :

la valeur de ρ	le temps de calcule	Nombre d'itération
0.1	0:00:01.199728	18355
0.5	0:00:00.029258	432
1.0	0:00:00.006494	79
ρ optimale	0:00:00.055997	599

2.7 Reprendre la question précédente pour f (x) = π 2 sin(πx)

```
In [112]: # 2ème Test pour f(x) =   [U+FFFD][U+FFFD]^2sin([U+FFFD][U+FFFD] x)

          def f1(x):
              return  np.pi**2*np.sin(np.pi*x)

          def g1(x):

                  a=1.5 - 20*(x - 0.6)**2

                  if a<0:
                      return  0
                  else:
                      return a

          f=np.vectorize(f1)
```

```
g=np.vectorize(g1)

n = 100
x = np . linspace (0 ,1 , n + 2 )

xv = x [ 1 : - 1 ]
fv , gv = f ( xv ) , g ( xv )

Jf = lambda u : j(u,xv,fv)

DJf = lambda u : deltaJ(u,xv,fv)


#pour ρ = ρ_optimal

import numpy as np
import time
from datetime import timedelta

l=[5,20,50,100]
for i in l:
    start_time = time.monotonic()

    n = i

    x = np . linspace (0 ,1 , n + 2 )



    h=1/n

    A=np.zeros((n,n))

    for k in range(n):

       for j in range(n):

            if k==j:
                A[k,k]= 2/h
                if j-1 >=0:
                    A[k,j-1]= -1/h
                if j+1 <=n-1:
                    A[k,j+1]= -1/h


    l= np.linalg.eigvals(A)
    l=np.sort(l)
```

```
rho=2/(l[0]+l[-1])



xv = x [ 1 : - 1 ]
fv , gv = f(xv) , g(xv)
Jf = lambda u : j(u,xv,fv)

DJf = lambda u : deltaJ(u,xv,fv)
u0=np.array(n*[4])
l1= gradient_projete_fixe(Jf, DJf,gv,u0,rho,10^-5,10000,1)

end_time = time.monotonic()

print("le temps de calcul pour n = "+str(i)+"\n",timedelta(seconds=end_time - star


print("le nombre d'itération pour n = "+str(i)+"\n",len(l1))
```

```
le temps de calcul pour n = 5
 0:00:00.705168
le nombre d'itération pour n = 5
 10000
le temps de calcul pour n = 20
 0:00:01.289774
le nombre d'itération pour n = 20
 10000
le temps de calcul pour n = 50
 0:00:03.107963
le nombre d'itération pour n = 50
 10000
le temps de calcul pour n = 100
 0:00:09.638028
le nombre d'itération pour n = 100
 10000
```