TP2correction

January 26, 2021

1 TP2 - Résolution de systèmes linéaires - Correction

[1]:
```
import numpy as np
from math import *
import matplotlib.pyplot as plt
import time
```

1.1 Exercice 1 - Conditionnement

Définition de la matrice M_α

[2]:
```
def Malpha(alpha):
    return np.array([[1 + alpha, 1, 2], [3, 1, alpha], [1, 2 * alpha, 1]])
```

1.1.1 Question a (i)

[3]:
```
M = Malpha(2.01)
x = np.array([[1,1,1]]).T
b = M.dot(x)
deltab = 0.01 * np.random.rand(3, 1)
y = np.linalg.solve(M, b + deltab)
print(y)
```

```
[[1.15919522]
 [1.02473597]
 [0.75113653]]
```

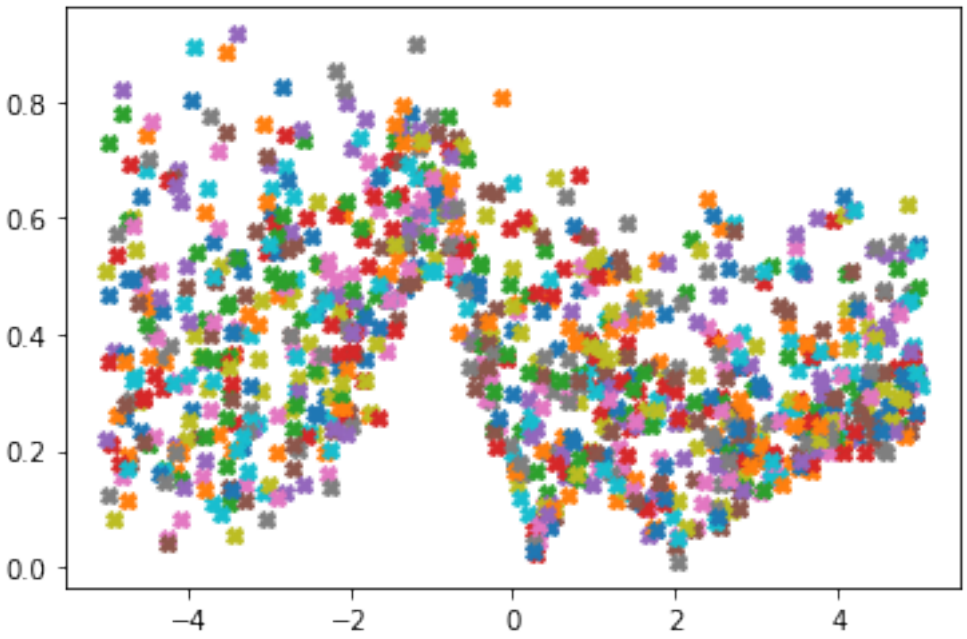
1.1.2 Question a (ii)

[4]:
```
print('erreur relative sur la perturbation : ',np.linalg.norm(y - x) / np.
 ↪linalg.norm(x))
print('borne théorique : ',np.linalg.cond(M) * np.linalg.norm(deltab) / np.
 ↪linalg.norm(b))
```

```
erreur relative sur la perturbation :  0.17116075964660038
borne théorique :  0.7035869239444641
```

1.1.3 Question b

[5]:
```
n = 1000
plt.figure(1)
for i in range(n):
    alpha = np.random.rand() * 10 - 5
    M = Malpha(alpha)
    x = np.random.rand(3, 1)
    b = M.dot(x)
    deltab = 0.01 * np.random.rand(3, 1)
    y = np.linalg.solve(M, b + deltab)
    rapport = np.linalg.norm(y - x) / np.linalg.norm(x) / (np.linalg.cond(M) *␣
 ↪np.linalg.norm(deltab) / np.linalg.norm(b))
    plt.plot(alpha,rapport,'X')
plt.show()
```



1.2 Exercice 2 - Mise en place d'un problème aux différences finies

[6]:
```
def a(x):
    #fonction a, programmée de sorte à rendre un vecteur de même taille que␣
 ↪l'entrée x
    return 0*x + 1.
def f(x):
    #fonction f, programmée de sorte à rendre un vecteur de même taille que␣
 ↪l'entrée x
```

```
    return 0*x + 1.
def construction(n,a,f):
    # n entier, a et f le nom des fonctions concernees
    h = 1/(n+1)
    vec = a((0.5+np.arange(n+1))*h)
    A = (np.diag(vec[1:-1],1)+np.diag(vec[1:-1],-1)-np.diag(vec[:-1]+vec[1:]))/
 ↪h**2
    F = f(np.arange(1,n+1)*h)
    return A,F
```

Cas simple où a et f sont constantes, égales à 1. On teste la méthode, connaissant la solution exacte.

[7]:
```
n = 100
h = 1/(n+1)
A,F=construction(n,a,f)
y=np.linalg.solve(A,F)
x=np.arange(1,n+1)*h
print('Erreur numérique : ',np.linalg.norm(y-x*(x-1)/2))
```

```
Erreur numérique :  7.774651937620554e-15
```

1.3 Exercice 3 : Chargement d'une corde d'élasticité inhomogène

[8]:
```
def a(x):
    #fonction a, programmée de sorte à rendre un vecteur de même taille que␣
 ↪l'entrée x
    return 1+ 0.8 * np.sin(10*x) * np.cos(5*x)
def f(x):
    #fonction f, programmée de sorte à rendre un vecteur de même taille que␣
 ↪l'entrée x
    xi1 = 0.2
    xi2 = 0.7
    return 0.1 + np.exp(-1000*(x-xi1)**2) + np.exp(-1000*(x-xi2)**2)
```

1.3.1 Question a

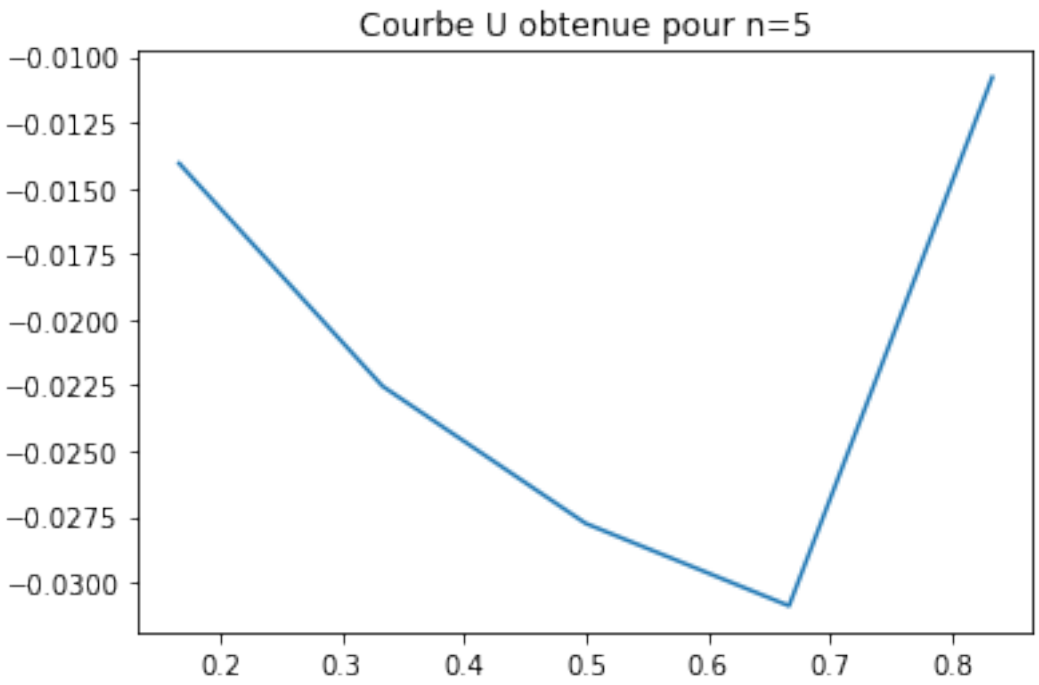
[9]:
```
n = 10
A,F = construction(n,a,f)
```

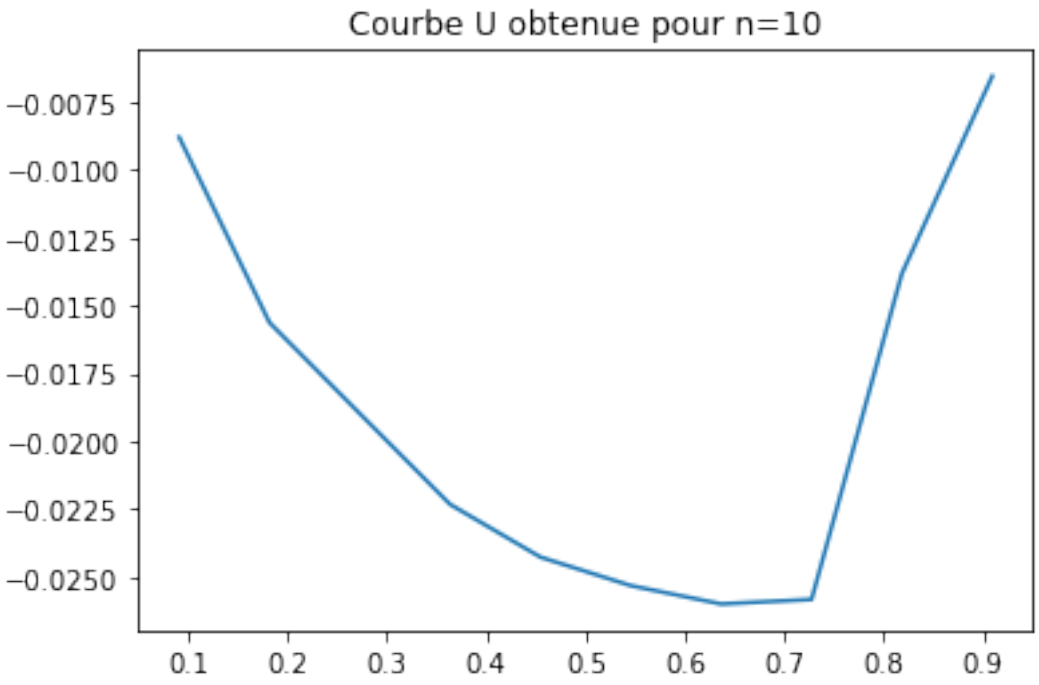
[10]:
```
if np.linalg.norm(A-A.T)==0.0 and max(np.linalg.eigvals(A))<0:
    print('La matrice A est bien symétrique et définie négative')
else:
    print('Erreur : la matrice A n est pas symétrique, définie négative')
```

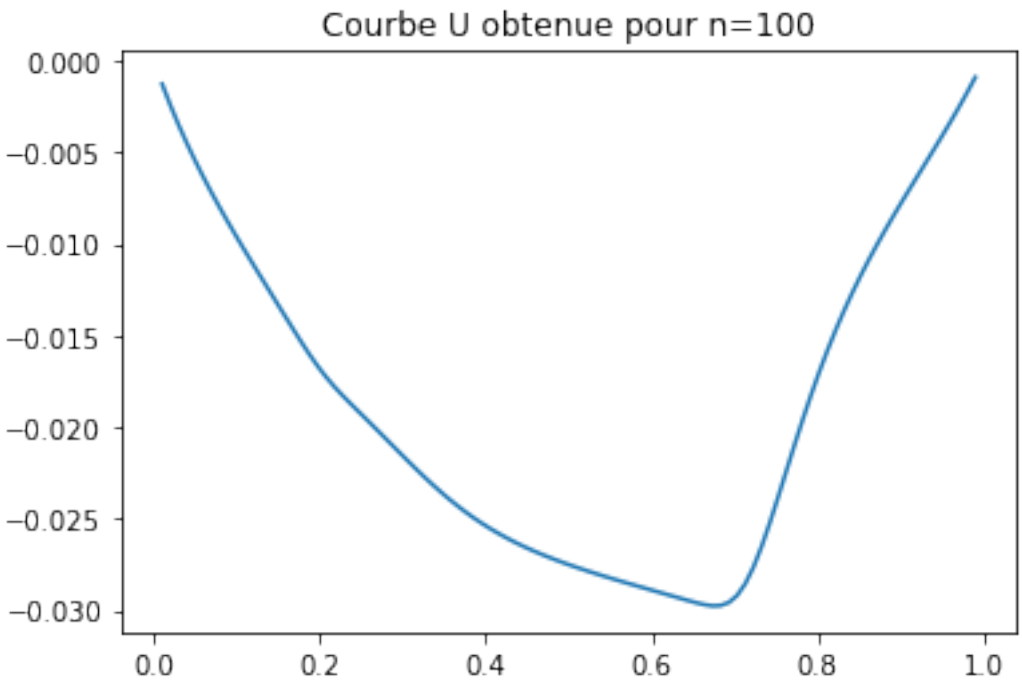
```
La matrice A est bien symétrique et définie négative
```

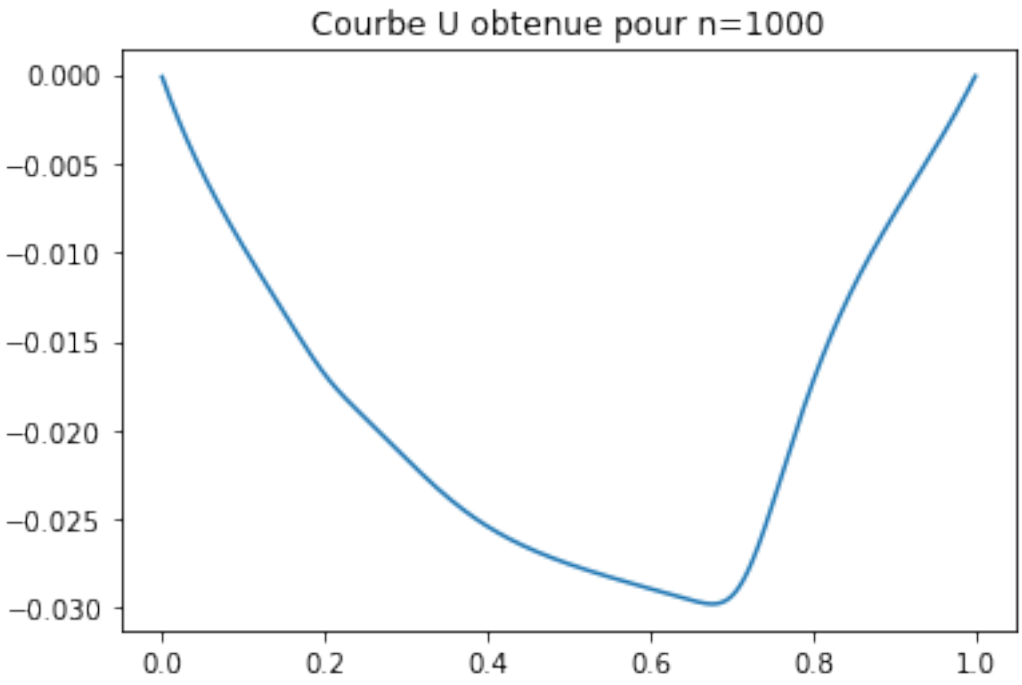
1.3.2 Question b

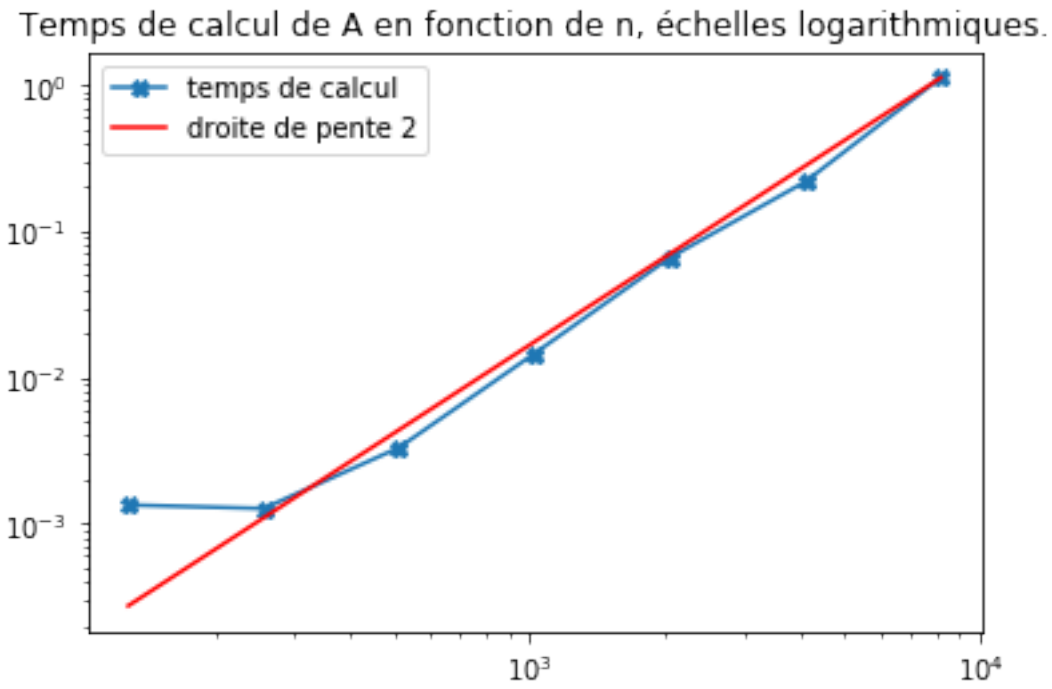
[11]:
```
tabn = [5,10,100,1000]
for n in tabn:
    h = 1/(n+1)
    A,F=construction(n,a,f)
    u=np.linalg.solve(A,F)
    x=np.arange(1,n+1)*h
    plt.figure()
    plt.plot(x,u)
    plt.title('Courbe U obtenue pour n='+str(n))
    plt.show()
tabn = 2**np.arange(7,14)
tabtempsA=[]
tabtempsU=[]
for n in tabn:
    h = 1/(n+1)
    temps0=time.time()
    A,F=construction(n,a,f)
    temps1=time.time()
    u=np.linalg.solve(A,F)
    temps2=time.time()
    tabtempsA.append(temps1-temps0)
    tabtempsU.append(temps2-temps1)
plt.loglog(tabn,tabtempsA,'X-',label = "temps de calcul")
plt.loglog(tabn,tabtempsA[-1]*tabn**2/tabn[-1]**2,'r-', label = "droite de␣
 ↪pente 2")
plt.title('Temps de calcul de A en fonction de n, échelles logarithmiques.')
plt.legend(loc = "best")
plt.show()
plt.loglog(tabn,tabtempsU,'X-', label = "temps de calcul")
plt.loglog(tabn,tabtempsU[-1]*tabn**2/tabn[-1]**2,'r-',label = "droite de pente␣
 ↪2")
plt.loglog(tabn,tabtempsU[-1]*tabn**3/tabn[-1]**3,'g-', label = "droite de␣
 ↪pente 3")
plt.title('Temps de calcul de U en fonction de n, échelles logarithmiques.')
plt.legend(loc = "best")
plt.show()
```

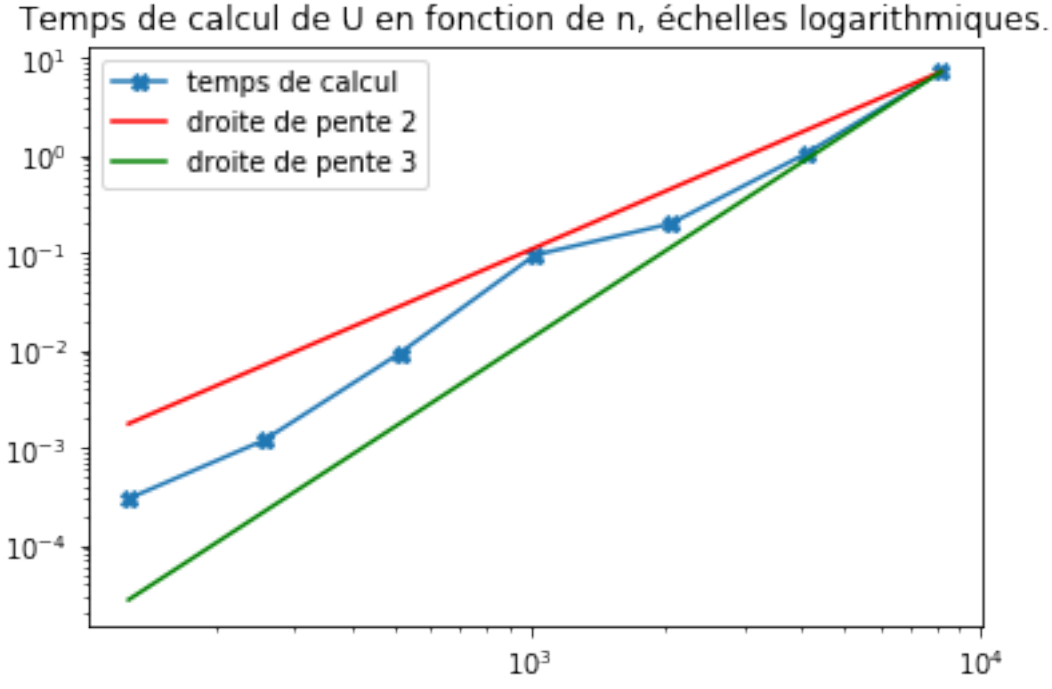

Courbe U obtenue pour n=5
−0.0100
−0.0125
−0.0150
−0.0175
−0.0200
−0.0225
−0.0250
−0.0275
−0.0300
0.2
0.3
0.4
0.5
0.6
0.7
0.8


Courbe U obtenue pour n=10
−0.0075
−0.0100
−0.0125
−0.0150
−0.0175
−0.0200
−0.0225
−0.0250
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9


Courbe U obtenue pour n=100
0.000
−0.005
−0.010
−0.015
−0.020
−0.025
−0.030
0.0
0.2
0.4
0.6
0.8
1.0


Courbe U obtenue pour n=1000
0.000
−0.005
−0.010
−0.015
−0.020
−0.025
−0.030
0.0
0.2
0.4
0.6
0.8
1.0

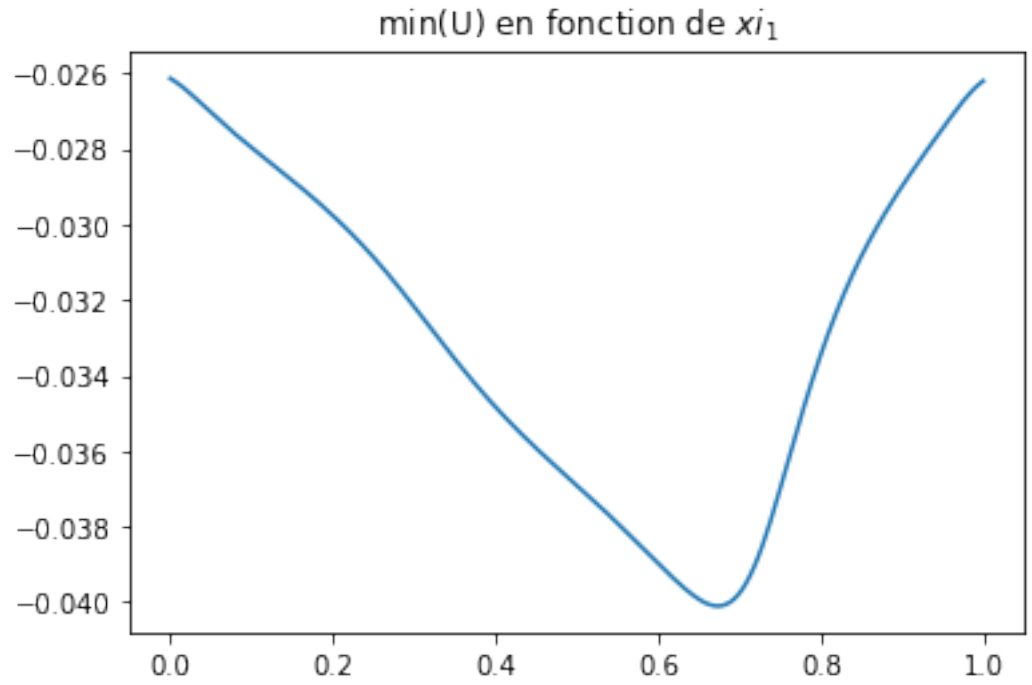

Temps de calcul de A en fonction de n, échelles logarithmiques.
temps de calcul
droite de pente 2
10^0
10^{-1}
10^{-2}
10^{-3}
10^3
10^4


Temps de calcul de U en fonction de n, échelles logarithmiques.
temps de calcul
droite de pente 2
droite de pente 3
10^1
10^0
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^3
10^4

1.3.3 Question c

[12]:
```
def f_0(x,xi1,xi2):
    #fonction f, programmée de sorte à rendre un vecteur de même taille que␣
 ↪l'entrée x
    return 0.1 + np.exp(-1000*(x-xi1)**2) + np.exp(-1000*(x-xi2)**2)

n = 1000
h = 1/(n+1)
xi1 = np.linspace(0,1,200)
xi2 = 0.7
min_u = []
for xi in xi1:
    def g(x):
        return f_0(x,xi,xi2)
    A,F = construction(n,a,g)
    u = np.linalg.solve(A,F)
    min_u.append(min(u))
plt.figure()
plt.plot(xi1,min_u)
plt.title('min(U) en fonction de $xi_1$')
plt.show()
```



1.3.4 Question d

[13]:
```
from matplotlib import cm
from mpl_toolkits.mplot3d import axes3d
```

[14]:
```
fig = plt.figure()
ax = fig.gca(projection='3d')

xi1 = np.linspace(0,1,20)
xi2 = np.linspace(0,1,20)

XI1,XI2 = np.meshgrid(xi1,xi2)
min_U = np.zeros_like(XI1)
for i in range (len(xi1)):
    for j in range (len(xi2)):
        def g(x):
            return f_0(x,xi1[i],xi2[j])
        A,F = construction(n,a,g)
        u = np.linalg.solve(A,F)
        min_U[i,j] = min(u)
surf = ax.plot_surface(XI1,XI2,min_U,cmap=cm.coolwarm)
plt.show()
```