Projet 3

Zouiche Omar

13 Février 2021

1ère partie : Exercice 9 : Minimisation sur un ensemble On considère $f(x,y) = -(log(x) + log(y) + log(1-x) + log(1-y))$ pour $(x,y) \in \Omega \subset \mathbb{R}$. 1) Déterminer et représenter Ω. 2) Déterminer le gradient et le hessien de f. 3) Pourquoi f admet un minimum local stric ? Déterminer le. On définit g sur Ω par $g(x,y) = e^{-f(x,y)}$. Justifier l'existence d'un maximum pour g et déterminer sa valeur.

1) La fonction f est définie si :

- $x \in]0,1[$
- $y \in]0,1[$ Car les fonctions logarithme sont définies de tel dans f, donc on peut prendre $\Omega =]0,1[^2$

2) On a :

$$\nabla f = (\frac{1}{1-x} - \frac{1}{x}, \frac{1}{1-y} - \frac{1}{y})$$

Et :

$$H(f) = \begin{pmatrix} \frac{1}{x^2} + \frac{1}{(1-x)^2} & 0 \\ 0 & \frac{1}{y^2} + \frac{1}{(1-y)^2} \end{pmatrix}$$

Avec :

- $\$ \frac{\partial f}{\partial x = \frac{1}{1-x} - \frac{1}{x} \$\$ \frac{\partial f}{\partial y} = \frac{1}{1-y} - \frac{1}{y} \$}$
- $\$ \frac{\partial^2 f}{\partial x^2 = \frac{1}{x^2} + \frac{1}{(1-x)^2} \$\$ \frac{\partial^2 f}{\partial x \partial y} = 0\$}$
- $\$ \frac{\partial^2 f}{\partial y \partial x = 0\$\$ \frac{\partial^2 f}{\partial y^2} = \frac{1}{y^2} + \frac{1}{(1-y)^2} \$}$

3) Les valeurs propres de $H(f)$ sont positives, donc f admet un minimum local strict et on a :

$$\nabla f = 0 \iff \frac{1}{1-x} - \frac{1}{x} = 0 \wedge \frac{1}{1-y} - \frac{1}{y} = 0 \iff x = 1/2 \wedge y = 1/2$$

- g est décroissante, donc le maximum de g est le minimum de f, on a :

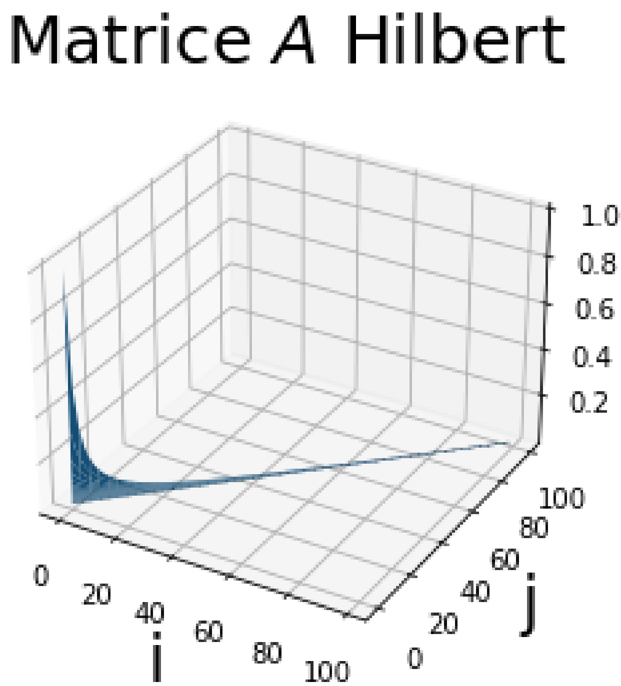
$$g(\frac{1}{2}, \frac{1}{2}) = \frac{1}{16}$$

****2ème partie :**** Soit $A \in M_{nn}(\mathbb{R})$ la matrice d'Hilbert, Nous allons résoudre $Ax = b$ avec gradproj.py et le code matrice_gradpy.py

[8]:
```
import numpy as np
import matplotlib.pyplot as plt
from math import exp
```

```python
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.ticker import LinearLocator
from scipy.linalg import hilbert


#Initialisation
igc=0 #pas de méthode du gradient conjugué
nbiter=100
res=np.zeros((nbiter))
rho_opt=np.zeros((nbiter))
N= 100
x=np.zeros((N))
b= np.zeros((N))
y=np.zeros((N))
a = np.zeros((N,N))
gk=np.zeros((N))
xx=np.zeros((N))
agk=np.zeros((N))

gkgk=0
agkgk=0
adk=np.zeros((N))
adkdk=0
gk1gk1=0

ortho=np.zeros((nbiter))
plt.figure()

a = hilbert(N)

#Boucle1 et graphe1 : Matrice A

for i in range(1,N):
    x[i]=i
    for j in range(1,N):
        y[j]=j

ax = plt.gca(projection='3d')
surf = ax.plot_surface(x,y,a, linewidth=0)
plt.title('Matrice $A$ Hilbert', fontsize = 25)
plt.xlabel('i', fontsize = 25)
plt.ylabel('j', fontsize = 25)


#Boucle2 et graphe2 : Vecteur b
```

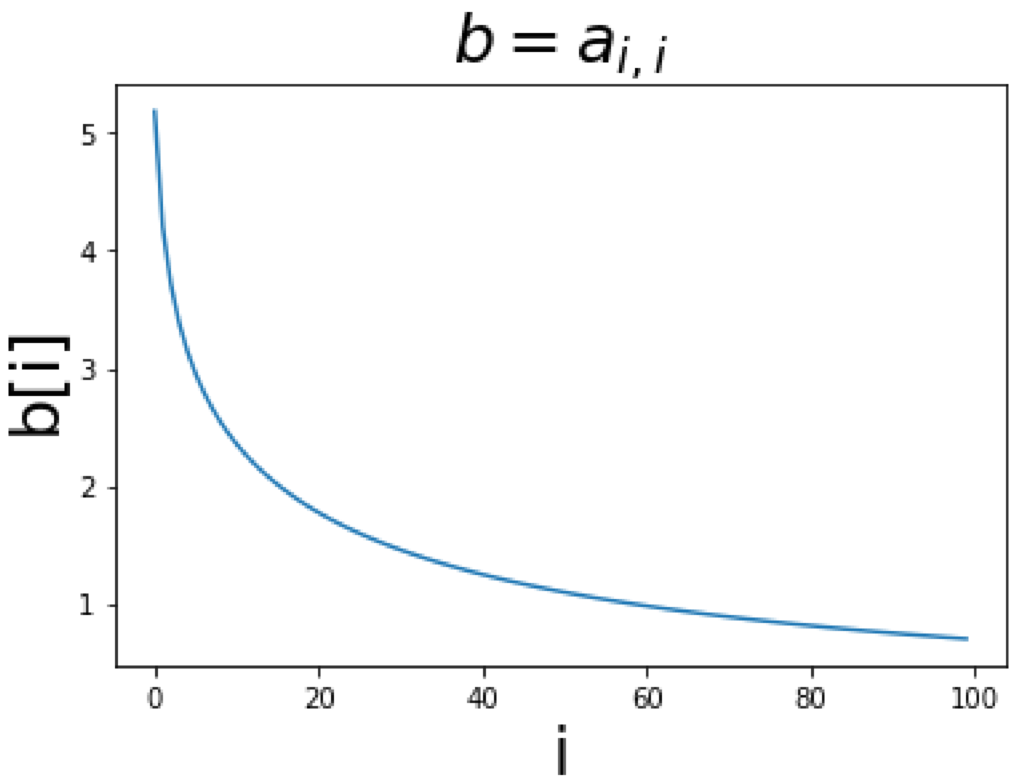
[8]: Text(0.5, 0, 'j')



[11]:
```
#Boucle2 et graphe2 : Vecteur b

b = a.dot(np.ones(N))
plt.figure()
plt.plot(b)
plt.title('$b = a_{i,i}$', fontsize = 25)
plt.xlabel('i', fontsize = 25)
plt.ylabel('b[i]',fontsize = 25)

#-------------------- MAIN PROGRAM -------------------

for k in range(0,nbiter): #------ BOUCLE GENERALE --------
    ortho[k]=0
    for i in range(1,N):
        gk[i]=0
        for j in range(1,N):
            gk[i]=gk[i]+a[i][j]*xx[j]
        gk[i]=gk[i]-b[i]
        for s in range (2,N):
            ortho[s]=abs((np.vdot(gk[s],gk[s-1])))
```

```
    for i in range(1,N):
        agk[i]=0
        for j in range(1,N):
            agk[i]=agk[i]+a[i][j]*gk[j]

    rho_opt[k]=0
    for i in range(1,N):
        gkgk += gk[i]**2
        agkgk += agk[i]*gk[i]
        rho_opt[i]=gk[i]**2/agk[i]*gk[i]


    alphak=gkgk/agkgk   #pas optimal
    rho_opt[k]/=rho_opt[0]



    res[k]=0  #res = || xk+1 - xk||
    for i in range(1,N):
        res[k]+=abs(alphak*gk[i])
        xx[i]-=alphak*gk[i]

    res[k]/=res[0]

plt.show()
```

```
<ipython-input-11-7bc576f3e8bc>:37: RuntimeWarning: invalid value encountered in
double_scalars
  rho_opt[k]/=rho_opt[0]
```

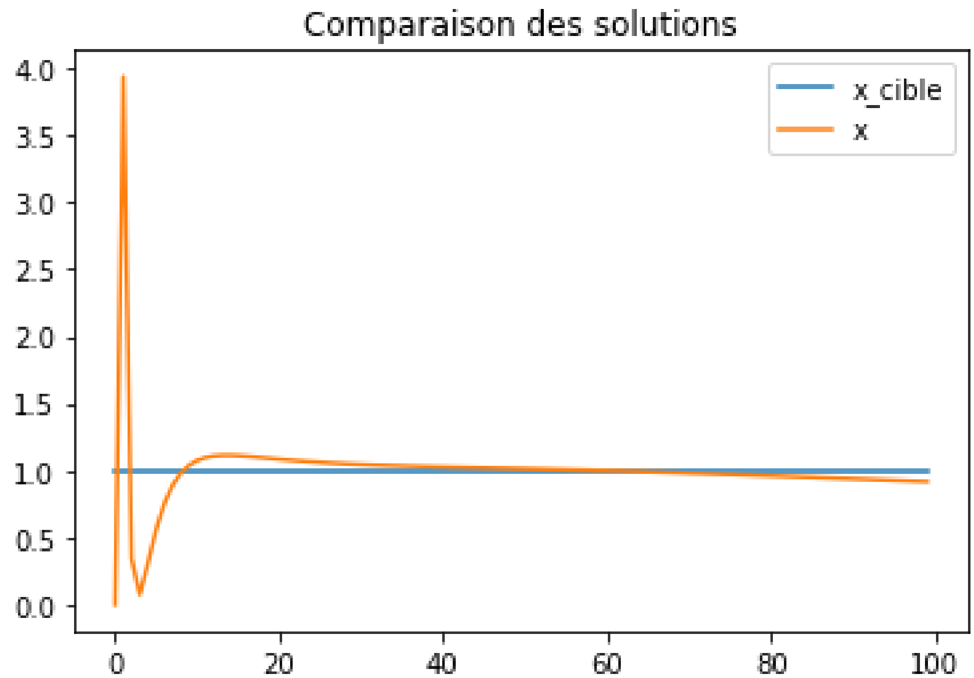


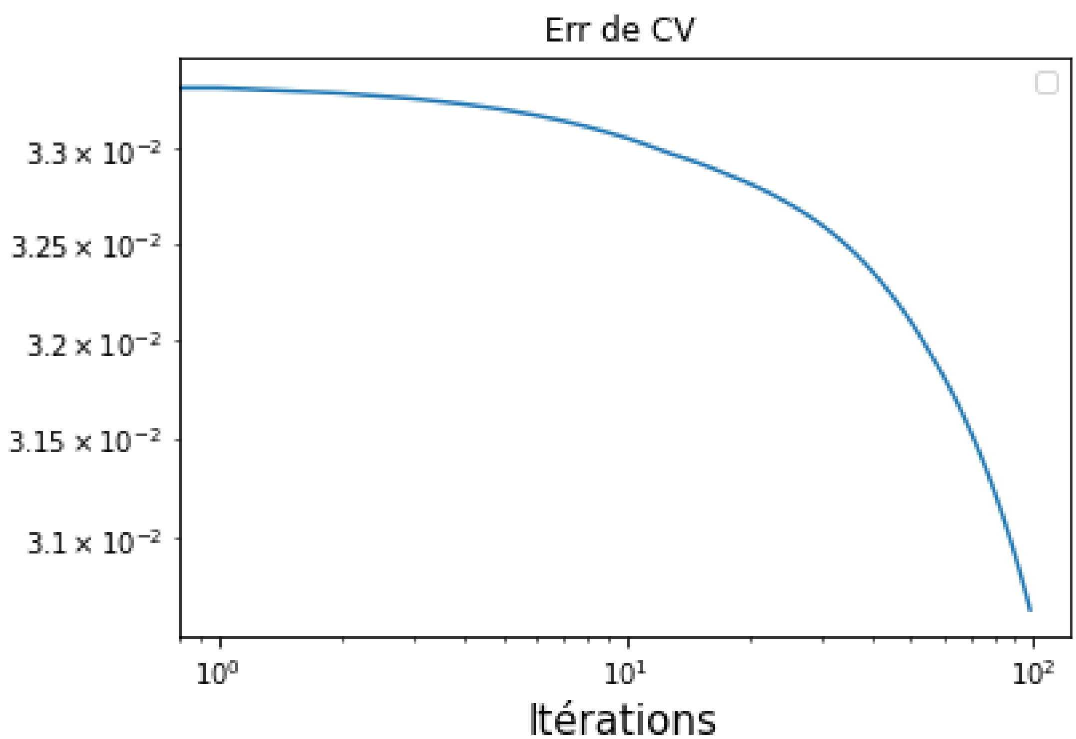
Ce graphique représente b cible. A étant mal conditionnée, nous allons maintenant voir la différence entre x et x cible.

[13]:
```
plt.figure()
plt.plot(x,np.ones(N), label='x_cible')
plt.plot(x,xx, label='x')
plt.title('Comparaison des solutions')
plt.legend()


plt.figure()
plt.plot(res[1:nbiter])
plt.title('Err de CV ')
plt.xscale("log")
plt.yscale('log')
plt.xlabel('Itérations', fontsize = 15)
plt.legend()
```

```
No handles with labels found to put in legend.
```

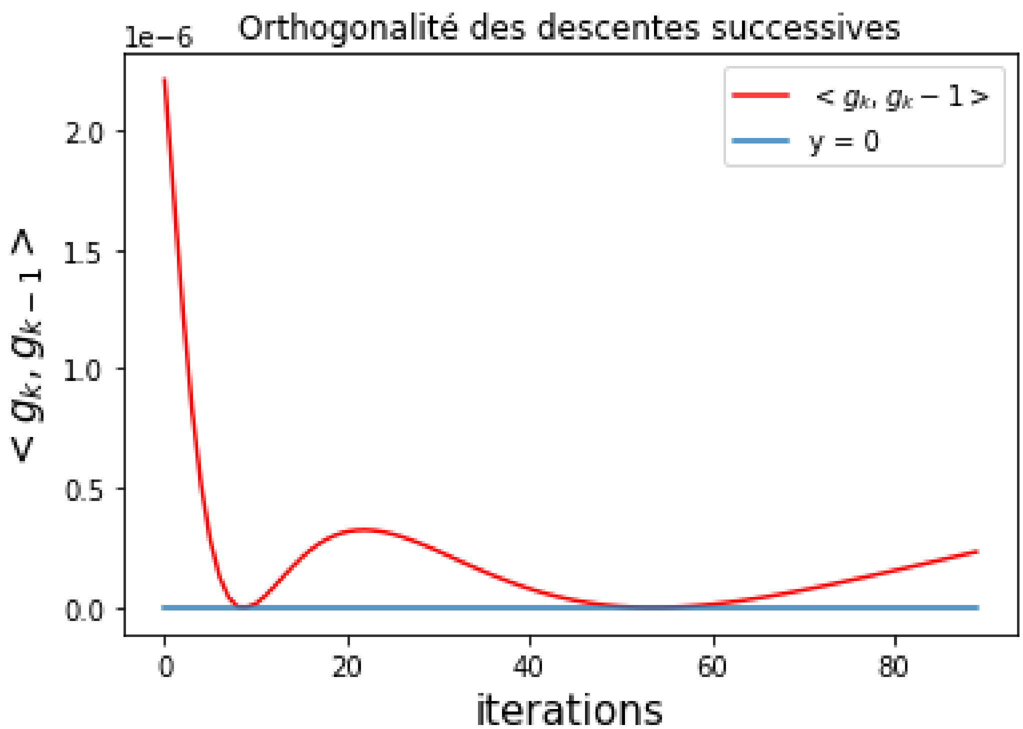
[13]: `<matplotlib.legend.Legend at 0x20e889e2a00>`





On observe clairement une différence entre x et x cible, la convergence des solutions n'est pas bonne. Quant à l'orthogonalité des descentes successives qu'on va calculer avec le produit scalaire $< g_k, g_{k-1} >$.

[14]:
```
plt.plot((ortho[10:nbiter]), color='r', label='$<g_k, g_k-1>$')
plt.plot(np.zeros(nbiter-10), label = 'y = 0')
plt.xlabel('iterations', fontsize=15)
plt.title('Orthogonalité des descentes successives')
plt.ylabel('$<g_k, g_{k-1}>$', fontsize = 15)
plt.legend()

plt.show()
```



Le produit scalaire $< g_k, g_{k-1} >$ est nul à partir de 50ème itération mais on n'a pas forcément une convergence vers 0.