Projet 1, Exercice 3

Zouiche Omar

31 Janvier 2021

1 HLMA606 Projet Semaine 1

1.1 Exercice 3 : Condition d'optimalité :

Soit

$$f : \begin{cases} \mathbb{R}^n \to \mathbb{R} \\ x \mapsto \langle x, y \rangle e^{-\|x\|^2} \end{cases}$$

Avec :

$$\|x\| = \sqrt{\sum_{i=1}^{n} x_i^2}$$

et le produit sclaire :

$$\langle x, y \rangle = \sum_{i=1}^{n} x_i y_i$$

On doit utiliser le code gradproj.py pour : - Calculer les dérivées partielles de f et trouver son gradient - Montre que : Si x est un point critique de f alors x est colinéaire à y, puis trouver les points critiques de f

les librairies que nous allons utiliser sont :

[21]:
```python
import math
import matplotlib.pyplot as plt
import numpy as np
import scipy as sp
import random
from scipy.optimize import minimize_scalar
```

On définit les fonctions qu'on va utiliser, Sachant que : - La dérivée partielle de f par rapport à la i-ème variable est :

$$\frac{\partial f}{\partial x_i}(x) = e^{-\|x\|^2}(y_i - 2x\langle x, y \rangle)$$

- La dimension de l'espace de départ est $n = 10$ - On prend $y = (1, \ldots ,1) \in \mathbb{R}^n$

Remarque : Trouver les dérivées partielles de f nous permet d'approcher le problème avec partie **analytique**.

[22]:
```python
#-------------------------------------------------------#
            # Définitions de fonctions
#-------------------------------------------------------#
```

```python
# Déf 1 : le produit scalaire  :
def scxy(x,y):
    scxy=0
    for i in range(0,len(x)):
        scxy = scxy + x[i]*y[i]
    return scxy

def func(x,y):
    f = scxy(x,y)*np.exp(-(np.linalg.norm(x))**2)
    return f

def funcp (x,y):
    fp=np.zeros(ndim)
    expx = np.exp(-(np.linalg.norm(x))**2)
    for i in range(0, len(x)):
        fp[i] = (y[i]*expx) - 2*x[i]*expx*scxy(x,y)
    return fp
#-----------------------------------------------------#
```

Suite du code graproj.py modifié et adapté à notre problématique

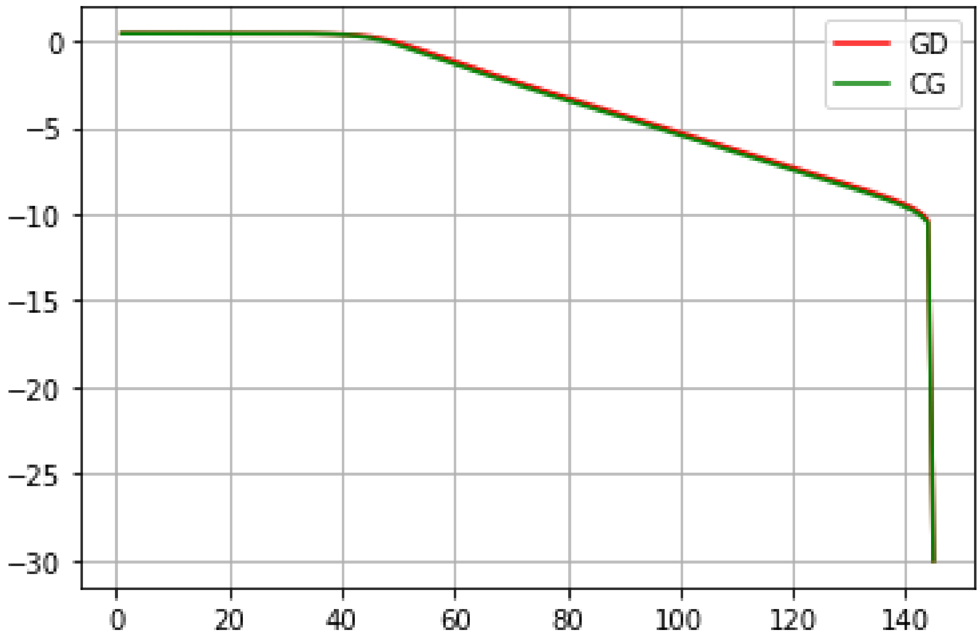
[25]:
```python
#-----------------------------------------------------#
        # Initialisations des variables
# cf. Données de l'énoncé :
#-----------------------------------------------------#
nbgrad=145
epsdf=0.0001
ndim=10
idf=0
y = np.ones(n)
#-----------------------------------------------------#

#-----------------------------------------------------#
for igc in [0,1]: #boucle generale
    ro0=0.0002
    ro=ro0

    it=[]
    history=[]
    historyg=[]

    for ii in range(0, nbgrad):
        it=it+[ii+1]
        history=history+[0]
        historyg=historyg+[0]
#-----------------------------------------------------#
```

```
        #Initialisation des vecteurs
#------------------------------------------------------#

    xmax=[]
    xmin=[]
    x=[]
    for i in range(0, ndim):
        xmax=xmax+[5]
        xmin=xmin+[-5]
        #x=x+[random.random()*0.03] #pour tester si cette méthode marche pour␣
 ↪tout x on peut prendre x aléatoire comme ceci
        x=x+ [0.20]



    dfdx=np.zeros(ndim)
    d=dfdx
    for itera in range(0, nbgrad):
        dfdx0=dfdx

        if idf==1:
            for i in range(0, ndim):
                x[i]=x[i]+epsdf
                fp=func(x, y)
                x[i]=x[i]-2*epsdf
                fm=func(x, y)
                x[i]=x[i]+epsdf
                dfdx[i]=(fp-fm)/(2*epsdf)
        elif idf==0: #Cas : On connait le gradient
            dfdx=funcp(x, y)


#gg comme la somme des carres de dfdx
        gg=0
        for j in range(0, ndim):
            gg=gg+dfdx[j]**2

#steepest descent
        if igc==0:
            for j in range(0, ndim):
                d[j]=dfdx[j]
        if igc==1:
#Polack-Ribiere  Conjugate Gradient
            xnum=0
            for j in range(0, ndim):
                xnum=xnum+dfdx[j]*(dfdx[j]-dfdx0[j])
            xden=0
```

```
            for j in range(0, ndim):
                xden=xden+dfdx[j]**2
            beta=0
            if(xden>1.e-30):
                beta=max(0,xnum/xden)

            for j in range(0, ndim):
                d[j]=dfdx[j]+beta*d[j]


#New xn+1= xn+rho*d  with d either -Gradf ou from CG
        for i in range(0, ndim):
            x[i]=x[i]-ro*d[i]
            x[i]=max(min(x[i], xmax[i]), xmin[i])

        f=func(x, y)
        history[itera]=f
        historyg[itera]=gg

####################################################
        if (itera >2 and history[itera-1] > f):
             ro=min(ro*1.25, 100*ro0)
        else:
             ro=max(ro*0.6, 0.01*ro0)
#####################################################

    print(x)

    h1=history[nbgrad-1]
    hg1=historyg[1]

    for itera in range(0, nbgrad):
        history[itera]=max(abs(history[itera] - h1),1.0e-30)
        historyg[itera]=historyg[itera] /hg1
    #plt.plot(it,history)
    print("igc=",igc)
    if igc==0:
        plt.plot(it, np.log10(history), color='red', label='GD')
    if igc==1:
        plt.plot(it, np.log10(history), color='green',label='CG')

plt.legend()
plt.grid(True)
plt.show()
```

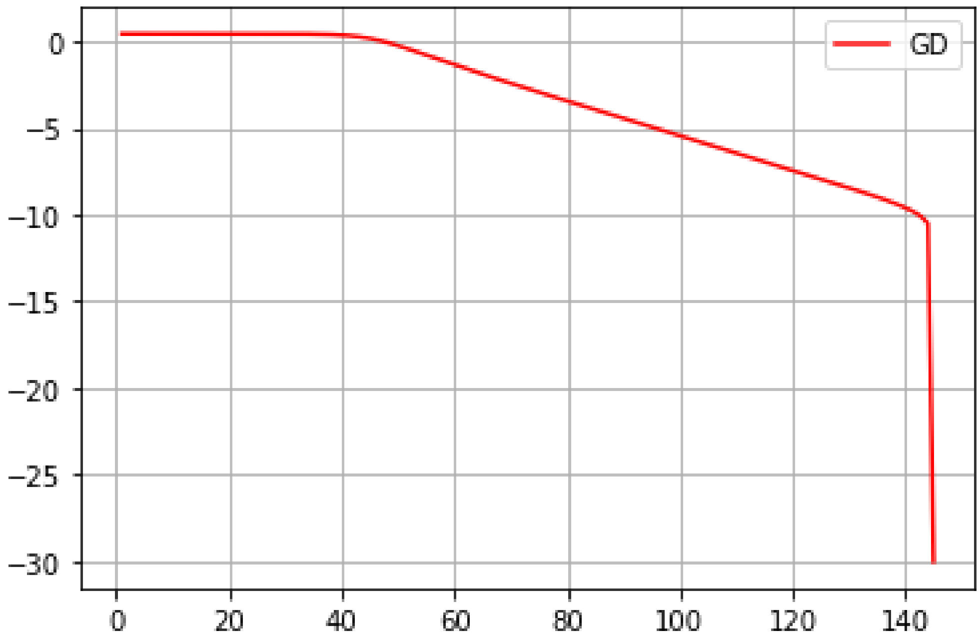
```
[-0.22360430171928355, -0.22360430171928355, -0.22360430171928355,
-0.22360430171928355, -0.22360430171928355, -0.22360430171928355,
```

```
-0.22360430171928355, -0.22360430171928355, -0.22360430171928355,
-0.22360430171928355]
igc= 0
[-0.22360471871887008, -0.22360471871887008, -0.22360471871887008,
-0.22360471871887008, -0.22360471871887008, -0.22360471871887008,
-0.22360471871887008, -0.22360471871887008, -0.22360471871887008,
-0.22360471871887008]
igc= 1
```



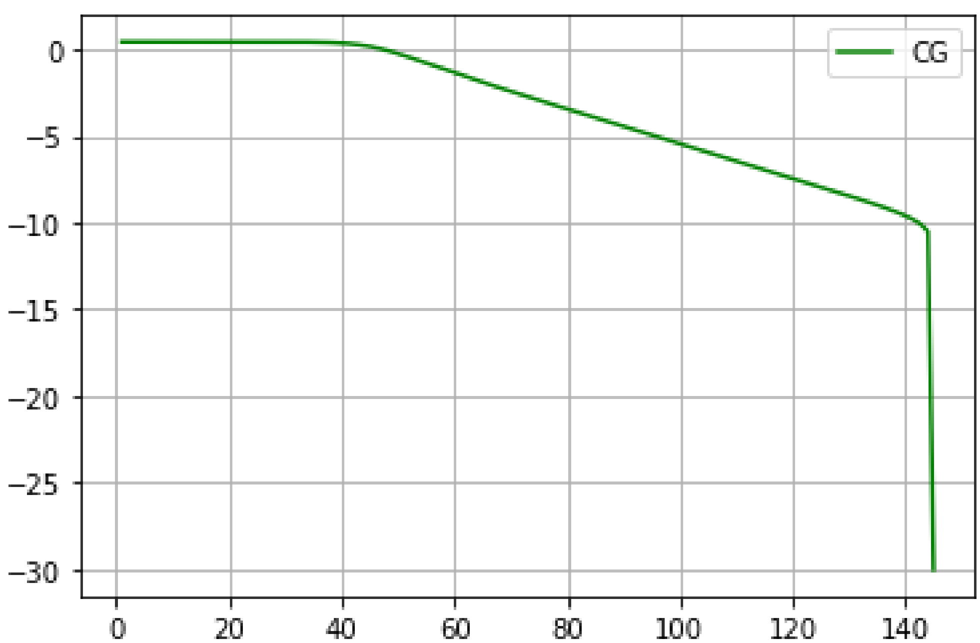
Les résultats des méthodes **Descente de gradient** est donnée par :

[36]:
```
plt.plot(it, np.log10(history), color='red', label='GD')
plt.legend()
plt.grid(True)
plt.show()
```



Les résultats de la méthode **Gradient conjugué** sont donnés par :

[34]:
```
plt.plot(it, np.log10(history), color='green',label='CG')
plt.legend()
plt.grid(True)
plt.show()
```



Ces résultats utilise le **gradient exact** qu'on a trouvé **analytiquement** et qu'on a définit au dessus.

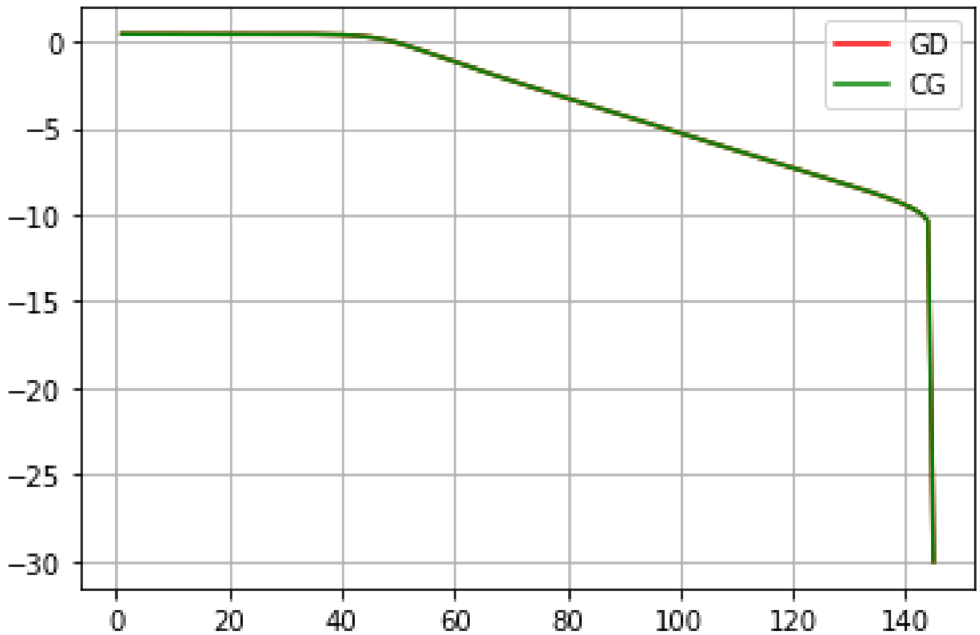
Approche numérique

[24]:
```
#-----------------------------------------------------#
        # Initialisations des variables
# cf. Données de l'énoncé :
#-----------------------------------------------------#
nbgrad=145
epsdf=0.0001
ndim=10
idf=0
y = np.ones(n)
#-----------------------------------------------------#

for igc in [0,1]: #boucle generale
    ro0=0.0002
    ro=ro0

    it=[]
    history=[]
    historyg=[]

    for ii in range(0, nbgrad):
        it=it+[ii+1]
        history=history+[0]
        historyg=historyg+[0]

    xmax=[]
    xmin=[]
    x=[]
    for i in range(0, ndim):
        xmax=xmax+[5]
        xmin=xmin+[-5]

        x=x+ [0.20]

    dfdx=np.zeros(ndim)
    d=dfdx

    for itera in range(0, nbgrad): #iter pour le prof
        dfdx0=dfdx

        if idf==1:
            for i in range(0, ndim):
```

```
                x[i]=x[i]+epsdf
                fp=func(x, y)
                x[i]=x[i]-2*epsdf
                fm=func(x, y)
                x[i]=x[i]+epsdf
                dfdx[i]=(fp-fm)/(2*epsdf)
        elif idf==0:
            dfdx=funcp(x, y)


        gg=0
        for j in range(0, ndim):
            gg=gg+dfdx[j]**2

#steepest descent
        if igc==0:
            for j in range(0, ndim):
                d[j]=dfdx[j]
        if igc==1:
#Polack-Ribiere  Conjugate Gradient
            xnum=0
            for j in range(0, ndim):
                xnum=xnum+dfdx[j]*(dfdx[j]-dfdx0[j])
            xden=0
            for j in range(0, ndim):
                xden=xden+dfdx[j]**2
            beta=0
            if(xden>1.e-30):
                beta=max(0,xnum/xden)

            for j in range(0, ndim):
                d[j]=dfdx[j]+beta*d[j]


#New xn+1= xn+rho*d  with d either -Gradf ou from CG
        for i in range(0, ndim):
            x[i]=x[i]-ro*d[i]
            x[i]=max(min(x[i], xmax[i]), xmin[i])

        f=func(x, y)
        history[itera]=f
        historyg[itera]=gg

#          # "petite astuce d'optimisation"
        if (itera >2 and history[itera-1] > f):
             ro=min(ro*1.25, 100*ro0)
        else:
```

```
            ro=max(ro*0.6, 0.01*ro0)
#-----------------------------------------------------#

    print(x)

    h1=history[nbgrad-1]
    hg1=historyg[1]

    for itera in range(0, nbgrad):
        history[itera]=max(abs(history[itera] - h1),1.0e-30)
        historyg[itera]=historyg[itera] /hg1
    #plt.plot(it,history)
    print("igc=",igc)
    if igc==0:
        plt.plot(it, np.log10(history), color='red', label='GD')
    if igc==1:
        plt.plot(it, np.log10(history), color='green',label='CG')

plt.legend()
plt.grid(True)
plt.show()
```

```
[-0.2236043017935167, -0.22360430179351481, -0.22360430179356858,
-0.22360430179355006, -0.22360430179353658, -0.22360430179359908,
-0.22360430179359916, -0.22360430179353435, -0.22360430179359214,
-0.22360430179353707]
igc= 0
[-0.2236043017935167, -0.22360430179351481, -0.22360430179356858,
-0.22360430179355006, -0.22360430179353658, -0.22360430179359908,
-0.22360430179359916, -0.22360430179353435, -0.22360430179359214,
-0.22360430179353707]
igc= 1
```



La méthode numérique donne bien les valeurs minimales de f, on peut aussi voir le maximum avec les fonctions func et funcp.