HW3_DECHARRIN

November 17, 2023

20/11/2024 de Charrin Théotime - MVA

1 Convex Optimization - Homework 3

1.1 Question 1

1.1.1 Rappels HW2

On se souvient que le dual de

$$\min_x \|Ax - b\|_2^2 + \|x\|_1 \quad \text{(P)}$$

a pour dual (en utilisant la norme duale $\|.\|_1^*$)

$$\max_\nu -\frac{1}{4}\|\nu\|_2^2 + \nu^T b \quad \text{s.c.} \begin{cases} \|A^T\nu\|_\infty \leq 1 \\ \nu \geq 0 \end{cases} \qquad \text{(D)}$$

On va réutiliser le même raisonnement ici.

1.1.2 Dans notre cas

On réécrit notre problème :

$$\begin{aligned} &\min_w \frac{1}{2}\|Xw - y\|_2^2 + \lambda\|w\|_1 \quad \text{(LASSO)} \\ \Leftrightarrow &\min_{z,w} \frac{1}{2}\|z\|_2^2 + \lambda\|w\|_1 \qquad \text{s.c. } z - Xw + y = 0 \end{aligned}$$

On a le lagrangien :

$$\begin{aligned} \mathcal{L}(z, w, v) &= \frac{1}{2}\|z\|_2^2 + \lambda\|w\|_1 + v^T(z - Xw + y) \\ &= \frac{1}{2}\|z\|_2^2 + v^T z + \lambda\|w\|_1 - v^T Xw + v^T y \end{aligned}$$

On résoud $\nabla_z \mathcal{L} = 0 \;\Leftrightarrow\; z = -v$

Comme pour **HW2**, résoudre

$$\begin{aligned} \inf_w \; \lambda\|w\|_1 - v^T Xw \Leftrightarrow \inf_w \; \|w\|_1 - \left(\frac{X^T v}{\lambda}\right)^T w \\ = \begin{cases} 0 \text{ si } \|\frac{X^T v}{\lambda}\|_\infty \leq 1 \\ +\infty \text{ sinon} \end{cases} \end{aligned}$$

On a donc

$$\begin{aligned}\inf_{z,w}\mathcal{L}(z,v,w) &= \underbrace{\frac{1}{2}\|-v\|_2^2+v^T(-v)}_{-\frac{1}{2}\|v\|_2^2}+\underbrace{\lambda\|w\|_1-v^TXw}_{\text{s.c. } \|\frac{X^Tv}{\lambda}\|_\infty\leq 1}+v^Ty\\ &= -\frac{1}{2}\|v\|_2^2+v^Ty \qquad \text{s.c. } \|\frac{X^Tv}{\lambda}\|_\infty\leq 1\end{aligned}$$

Le problème dual revient à maximiser cet inf, d'où

$$\max_v g(v)=\max_v -\frac{1}{2}\|v\|_2^2+v^Ty \Leftrightarrow \min_v \frac{1}{2}\|v\|_2^2-v^Ty \;\; \text{s.c. } \|\frac{X^Tv}{\lambda}\|_\infty\leq 1 \qquad \text{(DUAL)}$$

On peut dire que :

$$\begin{aligned}\|\frac{X^Tv}{\lambda}\|_\infty\leq 1 &\Leftrightarrow -1\leq\frac{[X^Tv]_i}{\lambda}\leq 1 \qquad \forall i\in[[1,n]]\\ &\Leftrightarrow \frac{[X^Tv]_i}{\lambda}\leq 1 \quad \text{et} \quad -\frac{[X^Tv]_i}{\lambda}\leq 1\\ &\Leftrightarrow \begin{pmatrix}\frac{X^T}{\lambda}\\ -\frac{X^T}{\lambda}\end{pmatrix}\preccurlyeq \mathbf{1}_{2d}\\ &\Leftrightarrow Av\preccurlyeq\lambda\mathbf{1}_{2d} \qquad \text{avec } A=\begin{pmatrix}X^T\\ -X^T\end{pmatrix}\end{aligned}$$

En notant $\frac{1}{2}\|v\|_2^2=v^T\frac{1}{2}I_nv=v^TQv$ et $y=-p, \quad b=\lambda\mathbf{1}_{2d}$ on a :

$$\min_v v^TQv+v^Tp \qquad \text{s.c.} \begin{cases}Q=\frac{1}{2}I_n\\ Av\preccurlyeq b\end{cases}$$

1.2 Question 2

Pour la méthode des points intérieurs, on transforme notre fonction objective $g_0(v)=v^TQv+v^Tp$ en la fonction $g_t(v)=tg_0(v)+\phi$ avec $\phi=-\sum_1^{2d}-\log(b_i-[Av]_i)$.

On calcule le gradient et la Hessienne de g_t (on note $(A_i)_{1\leq i\leq 2d}\in\mathbb{R}^n$ la i-ème ligne de A) :

$$\nabla_v g_t(v)=t(2Qv+p)+\sum_1^{2d}\frac{A_i^T}{b_i-[Av]_i}\nabla^2 g_t(v)=2tQ+\sum_1^{2d}\frac{A_iA_i^T}{(b_i-[Av]_i)^2}$$

On a le pseudo-algorithme suivant : > Choisir un v_0 faisable, $t_0>0,\ \mu>1,\epsilon>0$ > - Tant que $\frac{2d}{t}>\epsilon$: » - Faire une étape de centrage, *i.e.* trouver $\min_v g_t(v)$ par la méthode de Newton : »> - Tant que $\lambda^2=\nabla g_t(v)\nabla^2 g_t(v)^{-1}\nabla g_t(v)<\epsilon$: »» - Choisir un pas $\Delta v=-\nabla^2 g_t(v)^{-1}\nabla g_t(v)$ »» - Trouver la longueur du pas ξ tel que $g_t(v+\xi\nabla v)<g_t(v)+\alpha\xi\nabla g_t(v)^T\nabla v$ (par méthode de line backtracking de paramètres α et β) »» - Updater $v=v+\xi\nabla v$ » - Updater $t=\mu t$

1.3 Question 3

Pour évaluer l'impact de μ sur l'évalutation de w, on utilise la complementary slackness des conditions KKT (stricte convexité du lagrangien, stricte faisabilité): - on pose v^* solution optimale du dual (et donc du lagrangien $\mathcal{L}(z,w,v)$) - alors on vérifie $v^{*T}\sum_1^{2d}(z-Xw+y)=0$ à z,w optimaux.

On a vu plus haut que $\nabla_z \mathcal{L}(z, w, v^*) = 0 \Rightarrow z = -v^*$

Pour vérifier la complementary slackness il faut donc $-v^* - Xw + y = 0 \Leftrightarrow w = X^{-1}(y - v^*)$.

[1]:
```
%pip install cvxpy
%pip install umap-learn
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: cvxpy in
/home/theodechrn/.local/lib/python3.10/site-packages (1.4.1)
Requirement already satisfied: osqp>=0.6.2 in
/home/theodechrn/.local/lib/python3.10/site-packages (from cvxpy) (0.6.3)
Requirement already satisfied: ecos>=2 in
/home/theodechrn/.local/lib/python3.10/site-packages (from cvxpy) (2.0.12)
Requirement already satisfied: clarabel>=0.5.0 in
/home/theodechrn/.local/lib/python3.10/site-packages (from cvxpy) (0.6.0)
Requirement already satisfied: scs>=3.0 in
/home/theodechrn/.local/lib/python3.10/site-packages (from cvxpy) (3.2.4)
Requirement already satisfied: numpy>=1.15 in
/home/theodechrn/.local/lib/python3.10/site-packages (from cvxpy) (1.26.0)
Requirement already satisfied: scipy>=1.1.0 in
/home/theodechrn/.local/lib/python3.10/site-packages (from cvxpy) (1.11.3)
Requirement already satisfied: pybind11 in
/home/theodechrn/.local/lib/python3.10/site-packages (from cvxpy) (2.11.1)
Requirement already satisfied: qdldl in
/home/theodechrn/.local/lib/python3.10/site-packages (from osqp>=0.6.2->cvxpy)
(0.1.7.post0)
Note: you may need to restart the kernel to use updated packages.
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: umap-learn in
/home/theodechrn/.local/lib/python3.10/site-packages (0.5.4)
Requirement already satisfied: numpy>=1.17 in
/home/theodechrn/.local/lib/python3.10/site-packages (from umap-learn) (1.26.0)
Requirement already satisfied: scipy>=1.3.1 in
/home/theodechrn/.local/lib/python3.10/site-packages (from umap-learn) (1.11.3)
Requirement already satisfied: scikit-learn>=0.22 in
/home/theodechrn/.local/lib/python3.10/site-packages (from umap-learn) (1.3.1)
Requirement already satisfied: numba>=0.51.2 in
/home/theodechrn/.local/lib/python3.10/site-packages (from umap-learn) (0.58.1)
Requirement already satisfied: pynndescent>=0.5 in
/home/theodechrn/.local/lib/python3.10/site-packages (from umap-learn) (0.5.10)
Requirement already satisfied: tqdm in
/home/theodechrn/.local/lib/python3.10/site-packages (from umap-learn) (4.65.0)
Requirement already satisfied: tbb>=2019.0 in
/home/theodechrn/.local/lib/python3.10/site-packages (from umap-learn)
(2021.11.0)
Requirement already satisfied: llvmlite<0.42,>=0.41.0dev0 in
/home/theodechrn/.local/lib/python3.10/site-packages (from numba>=0.51.2->umap-
```

```
learn) (0.41.1)
Requirement already satisfied: joblib>=0.11 in
/home/theodechrn/.local/lib/python3.10/site-packages (from
pynndescent>=0.5->umap-learn) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/home/theodechrn/.local/lib/python3.10/site-packages (from scikit-
learn>=0.22->umap-learn) (3.2.0)
Note: you may need to restart the kernel to use updated packages.
```

[2]:

```
import numpy as np
import matplotlib.pyplot as plt
import cvxpy as cp
```

[3]:

```
#On l'utilisera fin de question 3, mais ça fait plus de sens d'initialiser tout␣
↪en même temps
def primal(w, X, y, Lambda=10):
    return .5 * np.linalg.norm(X @ w - y, 2)**2 + Lambda * np.linalg.norm(w, 1)

##La fonction g(v) du problème dual
def dual (v,p):
    n=v.shape[0]
    Q=.5*np.eye(n)
    return v.T @ Q @ v + p.T@v

## La fonction f(x)=tf_0(x)+phi(x)
def logbarrier(v,Q,p,A,b,t):
    if np.any(b - A.dot(v) <= 0):
        raise Exception('v non faisable : log négatif ', b-A@v)
    #return t * dual(v,p) - np.sum(np.log(b - A @ v))
    return t*(v.T@Q@v + p.T.dot(v)) - np.sum(np.log(b-A.dot(v)))
##Dérivée d'ordre 1 de f, NÉGATIVE???
def g(v,Q,p,A,b,t):
    denominator=1/(b-A@v)
    grad=t*(2*Q@v+p)+A.T@denominator
    return grad
    #return t*((Q.T+Q).dot(v)+p) + np.sum((1/(b-A.dot(v)).T)*A.T, axis=1).
↪reshape(-1,1)
##Hessienne de f
def Hessian(v,Q,p,A,b,t):
    denominator = 1 / (b - A @ v)
    H = 2 * t * Q + A.T @ np.diag(denominator)**2 @ A
    return H
    #temp = b-A.dot(v)
    #return 2*t*Q + np.sum([1/(temp[i])**2 * A[i,].reshape(-1,1).dot(A[i,].
↪reshape(1,-1)) for i in range(A.shape[0])])
```

[4]:
```
#On définit le line backtracking pour l'algorithme de Newton avec les␣
↪paramètres alpha et beta p. 23
def backtrack(f,grad_f,dv,v,t,alpha=0.1,beta=0.7):
    rate = 1
    #On vérifie que le nouveau v sera réalisable et qu'on n'a pas atteint le␣
↪critère d'arrêt
    while not (((b-A.dot(v+rate*dv))>0).all()) or (f(v+rate*dv,t) > f(v,t) +␣
↪alpha*rate*grad_f(v,t).T.dot(dv)):
        rate = beta*rate
    return rate


#Méthode de centering step avec la backtracking line
def centering_step(Q,p,A,b,t,v0,mu,alpha=0.7,beta=0.7,eps=1e-6):
    v_seq=[v0]
    inner_steps=0

    ##Pour être plus facile à rentrer dans backtrack
    f=lambda v_current, t:logbarrier(v_current,Q,p,A,b,t)
    grad_f=lambda v_current, t:g(v_current,Q,p,A,b,t)
    H=lambda v_current, t:Hessian(v_current,Q,p,A,b,t)

    v=v0
    while True:
        dv=-np.linalg.inv(H(v,t))@grad_f(v,t)
        lambda_gap=-grad_f(v,t).T@dv
        #On calcule (lambda^2)/2 pour le critère d'arrêt
        if 0.5*lambda_gap<=eps:
            break
            #On sort de la boucle while si on a atteint la précision machine␣
↪requise
        else:
            t_nt=backtrack(f,grad_f,dv,v,t,alpha,beta)
            v=v+t_nt*dv
            v_seq.append(v)
            inner_steps+=1
    return v_seq, inner_steps
```

[5]:
```
def barr_method(Q,p,A,b,v0,mu,alpha=0.1,beta=0.7,eps=1e-6):
    v_seq=[v0]
    t=1
    total_Nst=[0]
    #On a 2*d contraintes d'inégalités
    m=A.shape[0]
    while m/t>=eps:
        #Centering Step
        v,k=centering_step(Q,p,A,b,t,v_seq[-1],mu,alpha,beta,eps)
```

```
        vstar=v[-1]
        v_seq.append(vstar)
        t*=mu
        total_Nst.append(total_Nst[-1]+k)
    return np.array(v_seq),total_Nst
```

[6]:
```
from sklearn.datasets import make_regression
def random_toydataset(n,d,Lambda=10):
    X,y,coef = make_regression(n_samples=n, n_features=d, n_informative=10,␣
 ↪coef=True, noise = 1)
    p=-y
    Q=np.eye(n)/2
    b=Lambda*np.ones(2*d)
    v0=np.zeros(n)
    A=np.vstack((X.T,-X.T))
    return X,y,Q,p,A,b,v0
```

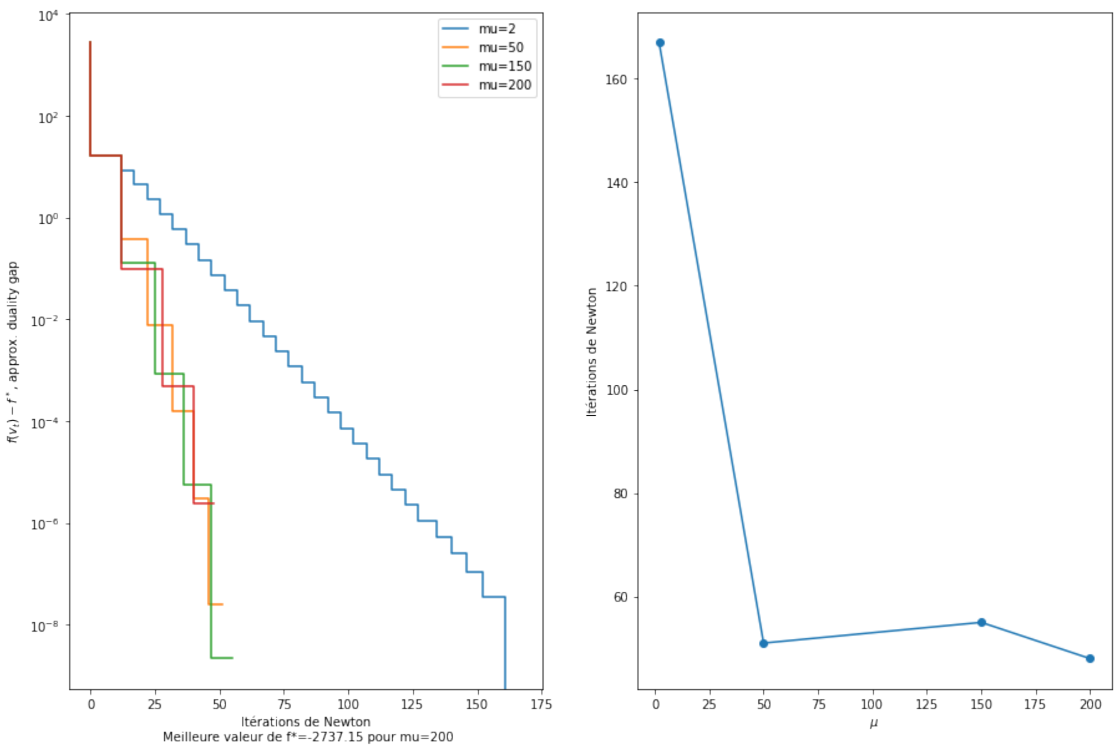
[7]:
```
#m=100 inégalités, n=50 variables
n=50
d=50
X,y,Q,p,A,b,v0=random_toydataset(n,d)
mu_list=[2,50,150,200]
v_total=[]
iters_total=[]
last_iter=[]
f_total=[]
for mu in mu_list:
    v_traj, iters_newt = barr_method(Q,p,A,b,v0,mu,alpha=.1,beta=.7,eps=1e-7)
    f=min([dual(v,p) for v in v_traj])
    v_total.append(v_traj)
    iters_total.append(iters_newt)
    last_iter.append(iters_newt[-1])
    f_total.append(f)

f_star=min(f_total)
best_mu=mu_list[np.argmin(f_total)]
fig,(ax1,ax2) =plt.subplots(1,2, figsize=(15,10))
for n, mu in enumerate(mu_list):
    values = [dual(v0,p) - f_star for v0 in v_total[n]]
    ax1.step(iters_total[n], values, label=f'{mu=}')
ax2.plot(mu_list, last_iter, marker='o')
ax2.set_xlabel('$\mu$')
ax2.set_ylabel('Itérations de Newton')
ax1.legend()
ax1.semilogy()
ax1.set_xlabel(f'Itérations de Newton\n Meilleure valeur de f*={f_star:.2f}␣
 ↪pour {mu=}')
```

```python
ax1.set_ylabel('$f(v_t)-f^*$, approx. duality gap')
plt.show()


#Valeurs théoriques avec le solver CVX
n=50
x = cp.Variable(n)
prob = cp.Problem(cp.Minimize(cp.quad_form(x, Q) + p.T @ x),
                 [A @ x <= b])
prob.solve()

# Print result.
print(f'La valeur est optimale selon CVX est {prob.value}\nNous avons trouvé {f_star}.\nDifférence : {f_star-prob.value}')
```



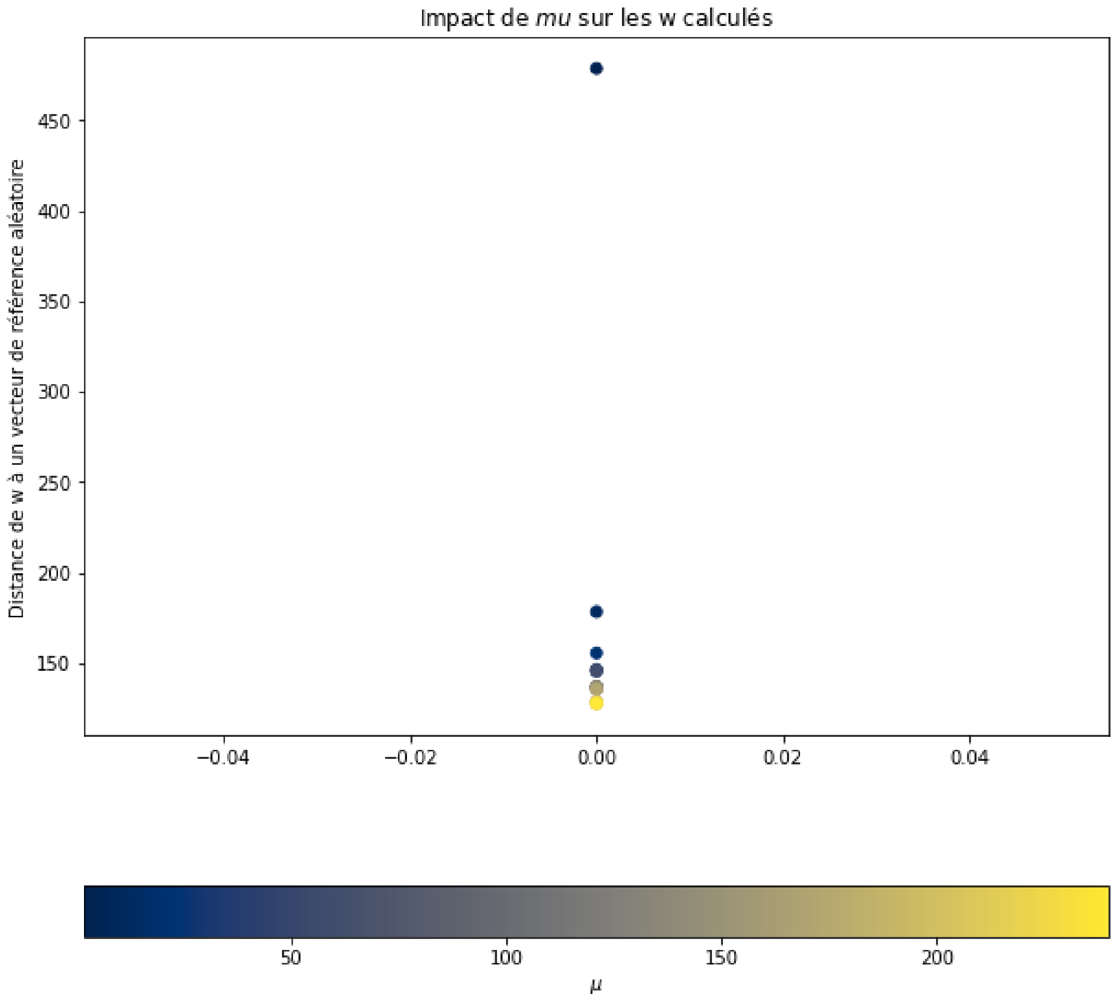

```
La valeur est optimale selon CVX est -2737.1509225863606
Nous avons trouvé -2737.1509225509735.
Différence : 3.538707460393198e-08
```

[8]:
```python
from matplotlib import colormaps
import umap
```

[11]:

```
#On a avec complementary slackness : w*=1/X * (y-v*)
distance=[]
w_list=[]
mu_list=np.linspace(2,240,25)
for n,mu in enumerate(mu_list):
    v_traj, iters_newt = barr_method(Q,p,A,b,v0,mu,alpha=.1,beta=.7,eps=1e-7)
    values = [dual(v0,p) - f_star for v0 in v_traj]
    v_star=np.argmin(values)
    w=np.linalg.pinv(X)@(y-v_star)
    w_list.append(w)
    reference=np.random.rand()*np.ones(w.shape)
    distance.append(np.linalg.norm(w-reference))

plt.figure(figsize=(10,10))
plt.scatter(np.
 ↪zeros(len(distance)),distance,marker='o',c=mu_list,cmap="cividis")
plt.colorbar(label="$\mu$", orientation="horizontal")
plt.title("Impact de $mu$ sur les w calculés")
plt.ylabel('Distance de w à un vecteur de référence aléatoire')
plt.show()
```


Impact de *mu* sur les w calculés
Distance de w à un vecteur de référence aléatoire
450
400
350
300
250
200
150
−0.04
−0.02
0.00
0.02
0.04
50
100
150
200
μ