Paule GRANGETTE
paule.grangette@gmail.com

HW3 - LASSO
Convex Optimization 2022

11/27/22

1 Question 1

Derive the dual problem of LASSO and format it as a general Quadratic Problem

Let's consider the LASSO problem :

$$\min_w \frac{1}{2}||Xw - y||_2^2 + \lambda||w||_1 \tag{1}$$

Let's set : $z = Xw - y$. The problem is now equivalent to :

$$\min_{w,z} \frac{1}{2}z^T z + \lambda||w||_1$$
$$\text{s.t. } z = Xw - y$$

Let's compute the Lagrangian of the problem.

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

But,

$$\inf_w(||w||_1 - (-\lambda^{-1}\mu^T w)) = \begin{cases} 0, & \text{if } ||-\lambda^{-1}\mu^T X||_* \leq 1 \\ -\infty, & \text{otherwise} \end{cases}$$

And let's consider : $\hat{\mathcal{L}}(z) = \frac{1}{2}z^T z - \mu^T z$. By deriving the equation :

$$\frac{\partial \hat{\mathcal{L}}}{\partial z}(z) = z - \mu \longrightarrow \frac{\partial \hat{\mathcal{L}}}{\partial z}(z_0) = 0 \iff z_0 = \mu$$

By inserting this value in the expression not derived, the minimum is :

$$\hat{\mathcal{L}}(z_0) = \frac{1}{2}\mu^T\mu - \mu^T\mu = -\frac{1}{2}\mu^T\mu$$

Finally, the function that minimizes the Laplacian is :

$$g(\mu) = \inf_{z,w}\mathcal{L}(w,z,\mu) = \begin{cases} -\mu^T y - \frac{1}{2}\mu^T\mu, & \text{if} ||\mu^T X||_* \leq \lambda \\ -\infty, \text{otherwise.} \end{cases}$$

Therefore, the dual problem of LASSO is :

$$\begin{array}{ccc} \max_\mu -\mu^T y - \frac{1}{2}\mu^T\mu & \iff & \min_v v^T Q v + p^T v \\ \text{s.t. } ||\mu^T X||_* \leq \lambda & & \text{s.t. } Av \leq b \end{array}$$

where $Q = \frac{1}{2}\mathrm{I}_n$, $p = y$, $b = \lambda\mathrm{I}_{2d}$, $A = \begin{pmatrix} X^T \\ -X^T \end{pmatrix}$

2 Question 2

Implement the barrier method to solve QP.

Let's consider : $f_{barr}(v) = tf(v) + \phi$, where $f(v) = v^T Q v + p^T v$ and $\phi = -\sum log(b - Av)$.

The implementation of the barrier method has been done using the algorithm given in lecture and in Boyd's book. For the centering step, I applied the Newton's method.

The Newton step is get for all $x \in \rtimes \gtrdot f$, by

$$\Delta x_{nt} = -\nabla^2 f_{barr}(x)^{-1} \nabla f_{barr}(x),$$

and Newton decrement with :

$$\lambda(x) = (\nabla f_{barr}(x)^T \nabla^2 f_{barr}(x)^{-1} \nabla f_{barr}(x))^{1/2} = (\Delta x_{nt}^T \nabla^2 f_{barr}(x) \Delta x_{nt})^{1/2}$$

Here is the value of the gradient and the hessian of f_{barr} :

$$\nabla f_{barr}(v) = t(Q^T + Q)v + p + \sum_{i=1}^{2d} \frac{A_i^T}{b_i - \sum_{j=1}^{n} A_{i,j} v_j},$$

$$\nabla^2 f_{barr}(v) = 2tQ + \sum_{i=1}^{2d} \frac{A_i^T A_i}{(b_i - \sum_{j=1}^{n} A_{i,j} v_j)^2},$$

with A_i the i-th line of the matrix A.

$\lambda^2/2$ is used as a stopping criterion since it approximates the quantity $f - p^*$:

$$f(x) - \inf_y \hat{f}(y) = f(x) - \hat{f}(x + \Delta x_{nt}) = \frac{1}{2}\lambda(x)^2$$

λ occurs also in the constant of the backtracking line search since :

$$-\lambda(x)^2 = \nabla f(x)^T \Delta x_{nt} = \frac{d}{dt} f(x + \Delta x_{nt} t)|_{t=0}$$

Furthermore, in the backtracking line search, we use two constants $\alpha \in (0, \frac{1}{2})$ and $\beta \in (0, 1)$ that need to be determined. By default, I used $\alpha = 0.01$ and $\beta = 0.5$

3 Question 3

What would be an appropriate choice for μ ?

Here are the results are got for $n = 100$ and $d = 400$ with a noise of standard deviation of 1.4 in the matrix X, an epsilon of $1e - 05$ and the first vector $v_0 = 0_{n,1}$ as $v_0 \in \mathbf{dom} f$.

We can see that whatever the value of μ, the precision curve remains exactly the same.
Nevertheless, for $\mu = 2$, the number of Newton steps is small but the number of iterations is a bit higher than the others with 92 iterations. For $\mu > 2$, the number of Newton steps is increasing exponentially with μ but the number of iterations is a bit smaller than for $\mu = 2$ and all comparable. So, a good trade-off to choose μ would be to choose μ in the order of the magnitude of ten. Here, we could choose $\mu = 50$ for example in the following figure.

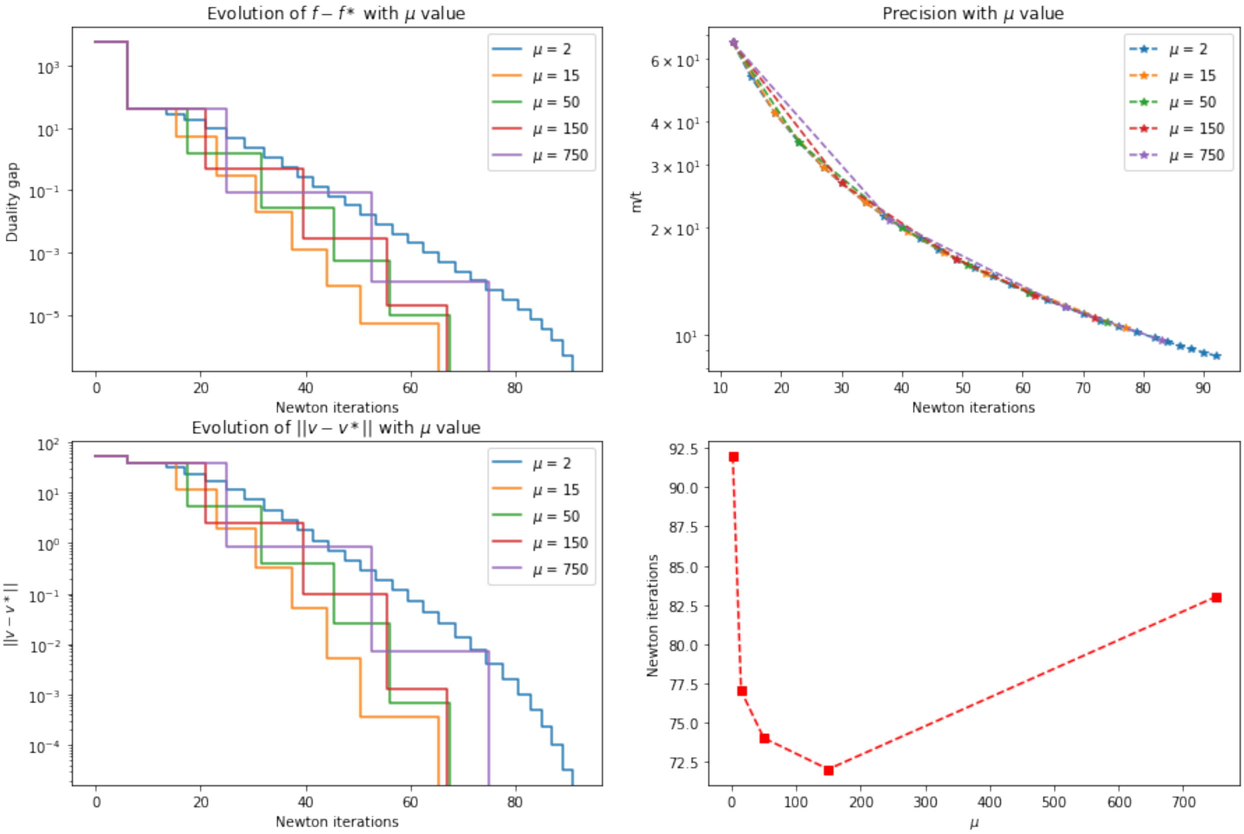

Figure 1: Performance comparison between μ values in the barrier method

Convex Optimization - Homework 3

Author : Paule Grangette

```
# Libraries
import numpy as np
import scipy.linalg as LA
import matplotlib.pyplot as plt
from sklearn.datasets import make_regression
```

Question 1

Derive the dual problem of LASSO and format it as a general Quadratic Problem.

Question 2

Implement the barrier method to solve QP.

```
# Useful functions
def f_barr(Q, p, A, b, t, v):
  '''Computes the barrier function f_barr = t*f + $\phi$,
  where f(v) = v.T@Q@v+p.T@v and phi = - np.sum(np.log(b-A@v))'''
  if (b-A@v<=0).any() :
    raise ValueError("Inequality constraint is not satisfied. v is not feasible.")
  return t*(v.T@Q@v + p.T@v) - np.sum(np.log(b - A@v))

def grad_barr(Q, p, A, b, v, t):
  '''Computes the gradient of f_barr'''
  return t*((Q + Q.T) @ v +p) + np.sum((1/(b-A@v).T*A.T), axis=1).reshape(-1,1)

def hessian_barr(Q, p, A, b, t, v):
  '''Computes the hessian of f_barr'''
  r = lambda v: b-A @ v
  hess_phi = np.zeros((len(v), len(v)))
  for i in range(A.shape[0]):
    hess_phi += 1/(r(v)[i])**2 * A[i,:].reshape(-1,1)@A[i,:].reshape(1,-1)
  return 2*t*Q + hess_phi

def linesearch(Q, p, A, b, t, v, delta, lamb_sq, alpha, beta):
  '''Backtracking line search : the constant used is -lambda**2
  NB : alpha $\in$ (0, 1/2) and beta $\in$ (0,1)'''
  slope = 1
  while (b-A@(v+slope*delta)<=0).any() or (f_barr(Q,p,A,b,t,v+slope*delta) > f_barr(Q,p,A,
    slope*=beta
```

```
  return slope


def centering_step(Q, p, A, b, t, v0, eps):

  alpha = 0.01
  beta = 0.5
  n_eps = 0
  v0_new = v0

  while True :
    v0 = v0_new
    #Compute the Newton step and decrement
    hess = hessian_barr(Q,p,A,b,t,v0)
    gradient = grad_barr(Q,p,A,b,v0,t)
    delta_v_nt = - LA.inv(hess) @ gradient
    lambda_square = float((-1)*gradient.T @ delta_v_nt)
    #Stopping criterion
    if lambda_square/2 <= eps :
      break
    #Line search
    slope = linesearch(Q, p, A, b, t, v0, delta_v_nt, lambda_square, alpha, beta)
    #Update
    n_eps += 1
    v0_new = v0 + slope*delta_v_nt

  return v0, n_eps


def barr_method(Q, p, A, b, v0, eps, mu):
  t = 1
  m = A.shape[0]
  precision_list = [m]
  n_list = [0]
  v_list = [v0]
  while True:
    #Centering step
    v0, n_eps = centering_step(Q,p,A,b,t,v0,eps)
    #Update the list
    v_list.append(v0)
    n_list.append(n_eps+n_list[-1])
    #Stopping criterion
    if m/t < eps :
      break
    #Increase t
    t = mu*t
    precision_list.append(t)
  return v_list, n_list, precision_list
```

Question 3

```
#Let's generate X and y in the LASSO problem and define the model
```

```
lamb = 10
n_samples = 100
n_features = 400
std_noise = 1.4

X, y, w_min = make_regression(n_samples, n_features, coef=True, noise = std_noise, random_

Q = np.eye(n_samples)/2
p = (-y).reshape(-1,1)
A = np.vstack((X.T, -X.T))
b = lamb*np.ones((2*n_features, 1))
v0 = np.zeros((n_samples,1))
eps = 1e-05
mu = [2, 15, 50, 150, 750]

fig, ax = plt.subplots(2,2,figsize=(15,10))
n_it_by_mu = []
n_sup_it_by_mu = []
for m in mu :
  v_seq, n_eps_seq, precision = barr_method(Q, p, A, b, v0, eps, m)
  v_best = v_seq[-1]
  v_diff = [np.sum(np.abs(v-v_best)) for v in v_seq]
  precision = [2*n_features/n for n in n_eps_seq[1:]]
  gap_traj = [float((v.T@Q@v + p.T@v) - (v_best.T@Q@v_best + p.T@v_best)) for v in v_seq]
  n_it_by_mu.append(n_eps_seq[-1])
  ax[0,0].step(n_eps_seq, gap_traj, where='mid', label = '$\mu$ = {}'.format(m))
  ax[1,0].step(n_eps_seq, v_diff, where='mid', label = '$\mu$ = {}'.format(m))
  ax[0,1].plot(n_eps_seq[1:], precision, '*--', label='$\mu$ = {}'.format(m))
ax[0,0].set_xlabel('Newton iterations')
ax[0,0].set_ylabel('Duality gap')
ax[0,0].set_title('Evolution of $f-f*$ with $\mu$ value')
ax[0,0].semilogy()
ax[0,0].legend()
ax[0,1].set_xlabel('Newton iterations')
ax[0,1].set_ylabel('m/t')
ax[0,1].set_title('Precision with $\mu$ value')
ax[0,1].semilogy()
ax[0,1].legend()
ax[1,0].set_xlabel('Newton iterations')
ax[1,0].set_ylabel('$||v-v*||$')
ax[1,0].set_title('Evolution of $||v-v*||$ with $\mu$ value')
ax[1,0].semilogy()
ax[1,0].legend()
ax[1,1].set_xlabel('$\mu$')
ax[1,1].set_ylabel('Newton iterations')
ax[1,1].plot(mu, n_it_by_mu,"rs--")
```

```
[<matplotlib.lines.Line2D at 0x7f20c9e31dd0>]
```