# Homework 3

Megi Dervishi

## Exercise 1

We can re-write the LASSO problem as follows by introducing a new variable:

$$\min_{v,w} \ \frac{1}{2}\|Xw-y\|_2^2 + \lambda\|w\|_1$$
$$\text{s.t} \quad v = Xw - y$$

The Lagrangian is:

$$\mathcal{L}((v,w),\mu) = \frac{1}{2}\|v\|_2^2 + \lambda\|w\|_1 + \mu^T(v - Xw + y)$$
$$= \frac{1}{2}\|v\|_2^2 + \mu^T v + \lambda\|w\|_1 - (X^T\mu)^T w + y^T\mu$$

And the dual function is:

$$g(\mu) = \inf_{v,w} \mathcal{L}((v,w),\mu) = y^T\mu + \inf_v \Big(\underbrace{\tfrac{1}{2}\|v\|_2^2 + \mu^T v}_{\text{a) convex and differentiable.}}\Big) + \inf_w \Big(\underbrace{\lambda\|w\|_1 - (X^T\mu)^T w}_{\text{b) use conjugate of } L_1 \text{ from Hw2.}}\Big)$$

a) $\nabla_v \mathcal{L}(v,w,\mu) = 0 \Rightarrow v = -\mu$

Hence $\min_v \frac{1}{2}\|v\|_2^2 + \mu^T v = \frac{1}{2}\|\mu\|_2^2 - \mu^T\mu$

$$= -\frac{1}{2}\|\mu\|_2^2$$

b) $\inf_w \lambda\|w\|_1 - (X^T\mu)^T w = \lambda \sup_w \left( \left(\frac{1}{\lambda}X^T\mu\right)^T w - \|w\|_1 \right)$

$$= \lambda \left\| \frac{1}{\lambda} X^T\mu \right\|_1^* = \left\| \frac{1}{\lambda} X^T\mu \right\|_1^* \qquad \text{since } \lambda > 0$$

$$= \begin{cases} 0 & \text{if } \|\frac{1}{\lambda}X^T\mu\|_\infty \leq 1 \\ +\infty & \text{otherwise.} \end{cases} \qquad \text{from} \quad \|\cdot\|_1^*(u) = \begin{cases} 0 & \text{if } \|u\|_\infty \leq 1 \\ +\infty & \text{otherwise} \end{cases}$$

So we have that: $g(\mu) = y^T\mu - \frac{1}{2}\|\mu\|_2^2 + \left\|\frac{1}{\lambda}X^T\mu\right\|_1^*$

And we have:

$$\max_v \ y^T\mu - \frac{1}{2}\|\mu\|_2^2 \quad \text{which can be reformulated as:}$$
$$\text{s.t} \quad \left\|\frac{1}{\lambda}X^T\mu\right\|_\infty \leq 1$$

$$\left\|\frac{1}{\lambda}X^T\mu\right\|_\infty \leq 1 \Leftrightarrow \max_i \left|\frac{1}{\lambda}X^T\mu\right|_i \leq 1 \Leftrightarrow \forall_i \ -\lambda \leq X^T\mu \leq \lambda \Leftrightarrow \begin{pmatrix} X^T\mu \\ -X^T\mu \end{pmatrix} \leq \lambda \mathbb{1}_{2d}$$

Hence we have for $p = -y$, $Q = \frac{1}{2}Id$, $A = \begin{pmatrix} X^T \\ -X^T \end{pmatrix}$, $b = \lambda \mathbb{1}_{2d}$

$$\min_v \ \mu^T Q\mu + p^T\mu$$
$$\text{s.t} \quad A\mu \leq b.$$

# Exercise 2

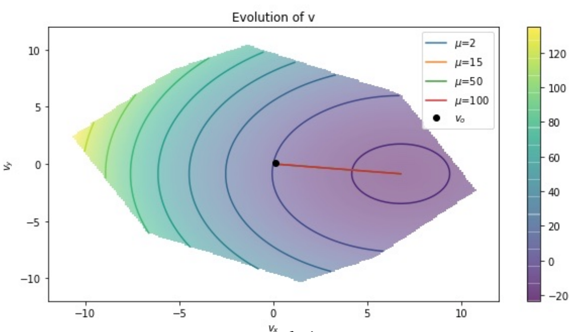
```python
import numpy as np
import matplotlib.pyplot as plt


#Question 2
b = 10

def g(Q,p,A,t,v):
  return t*(v.T @Q@v + p.T@v)- np.sum(np.log(b-A@v))

def backtrack_linesearch(Q,p,A,t,v,grad,newton_step):
  step_t = 1
  alpha = 0.05
  beta = 0.95
  def condition(step_t):
    c = g(Q,p,A,t,v+step_t * newton_step) > g(Q,p,A,t,v)+alpha*step_t*grad.T@newton_step
    return c
  while condition(step_t) or (any(b-A@(v+step_t*newton_step)<=0)):
    step_t = beta*step_t
  return step_t


#Centering step
def centering_step(Q,p,A,b,t,v0,eps):
  v = v0
  prev_v = v*np.inf
  v_seq = [v0]
  while np.linalg.norm(prev_v - v) > eps:
    h = 1/(b - A@v)
    grad = t * (2*Q@v + p) + A.T@h
    hessian = 2 * t * Q + A.T @ np.diag(h.reshape(-1))**2 @ A
    newton_step = -np.linalg.pinv(hessian) @ grad
    step_t = backtrack_linesearch(Q,p,A,t,v,grad,newton_step)
    prev_v = v
    v = v + step_t * newton_step
    v_seq.append(v)
  return v_seq

def f(Q,p,v):
  return v.T@Q@v + p.T@v
```

```python
#Barrier method
def barr_method(Q,p,A,b,v0,eps,u):
  t = 1
  c = A.shape[0] #number of constraints
  v = v0
  v_seq = [v0]
  precision_seq = [c/t]
  fv = [f(Q,p,v)[0,0]]
  while (c/t > eps):
    v = centering_step(Q,p,A,b,t,v0,eps)[-1]
    v_seq.append(v)
    t = t*u
    precision_seq.append(c/t)
    fv.append(f(Q,p,v)[0,0])
  return v_seq, precision_seq, abs(fv-fv[-1])
```

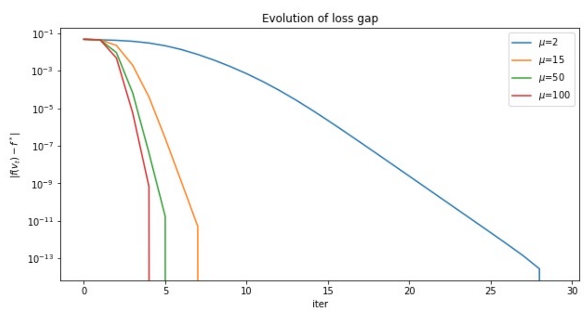
# Exercise 3.

```python
#Question 3
lambd = 10
X =  0.4*np.random.randn(2, 100)
y =  4*np.random.randn(2).reshape(-1,1)
Q =  np.eye(2)/2
p =  -y
A = np.concatenate((X.T, - X.T), axis=0)
v0 = np.zeros(2).reshape(-1,1)+0.01
eps = 1e-8

# colorplot
dstep = 0.1
pmin, pmax = -1.2*lambd, 1.2*lambd
cory, corx = np.mgrid[pmin:pmax+dstep:dstep, pmin:pmax+dstep:dstep]
potential = np.zeros(cory.shape)
for i in range(cory.shape[0]):
  for j in range(cory.shape[1]):
    cur_point = np.array([corx[i, j], cory[i, j]])
    #print(cur_point.shape, Q.shape, p.shape)
    if (A @ cur_point <= b).all():
      potential[i,j] = cur_point.T @ Q @ cur_point + p.T @ cur_point
    else:
      potential[i,j] = np.inf

print(potential.shape)

v_seq_u = dict((u,[]) for u in [2,15,50,100])
precision_u = dict((u,[]) for u in [2,15,50,100])
gap_u = dict((u,[]) for u in [2,15,50,100])

for u in [2,15,50,100]:
    v_seq,precision,gap=barr_method(Q,p,A,b,v0,eps,u)
    v_seq_u[u] = np.array(v_seq).squeeze()
    precision_u[u] = precision
    gap_u[u] = gap
```

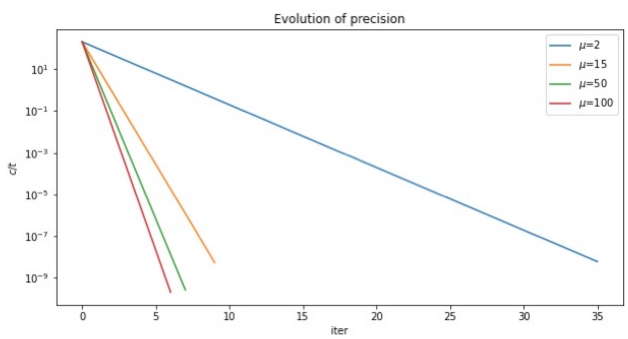
```python
#plots
plt.figure(figsize =(10,5))
plt.contour(corx, cory, potential)
plt.pcolor(corx, cory, potential, alpha=0.5, snap=True)
colorbar = plt.colorbar()
colorbar.set_alpha(0.5)
for u in v_seq_u.keys():
  plt.plot(v_seq_u[u][:,0],v_seq_u[u][:,1],label=f'$\mu$={u}')
plt.plot(0.1,0.1,'o',color='black', label='$v_o$')
plt.title('Evolution of v')
plt.xlabel('$v_x$')
plt.ylabel('$v_y$')
plt.legend()
plt.show()

plt.figure(figsize =(10,5))
for u in precision_u.keys():
  plt.semilogy(precision_u[u],label=f'$\mu$={u}')
plt.title('Evolution of precision')
plt.xlabel('iter')
plt.ylabel('$c/t$')
plt.legend()
plt.show()

plt.figure(figsize =(10,5))
for u in precision_u.keys():
  plt.semilogy(loss_u[u],label=f'$\mu$={u}')
plt.title('Evolution of loss gap')
plt.xlabel('iter')
plt.ylabel('$|f(v_t)-f^*|$')
plt.legend()
plt.show()
```




(a)




(b)




(c)

We can see from Plot (a) that all solutions converge, but having $\mu=50$ or $\mu=100$ takes less backtracking (outer) iterations to reach the solution than with $\mu=2$. So an appropriate $\mu$ could be $\mu=50$.