MVA - Convex Optimization
Homework 3

Borachhun YOU

We have LASSO problem:

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

with variable $w \in \mathbf{R}^d$, and $X \in \mathbf{R}^{n\times d}$, $y \in \mathbf{R}^n$, $\lambda > 0$

Question 1

Derive the dual problem of LASSO and format it as a general quadratic problem:

We first rewrite LASSO into:

$$\begin{aligned} &\min \ \frac{1}{2}\|z\|_2^2 + \lambda\|w\|_1 \\ &\text{s.t. } z = Xw - y \end{aligned}$$

Lagrangian:

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

Dual function:

$$\begin{aligned} g(v) &= \inf_{w,z} L(w, z, v) \\ &= \inf_z \left(\frac{1}{2}z^Tz + v^Tz\right) + \inf_w \left(\lambda\|w\|_1 - v^TXw\right) + v^Ty \end{aligned}$$

Consider:

$$h : z \longmapsto \frac{1}{2}z^Tz + v^Tz$$

$$\nabla h = z + v = 0 \Rightarrow z = -v$$

Therefore,

$$\inf_z \left(\frac{1}{2}z^T z + v^T z\right) = -\frac{1}{2}v^T v$$

Also,

$$\begin{aligned}
\inf_w \left(\lambda\|w\|_1 - v^T X w\right) &= \inf_w \left(-\left(\frac{v^T X}{\lambda}w - \|w\|_1\right)\right) \\
&= -\sup_w \left(\frac{v^T X}{\lambda}w - \|w\|_1\right) \\
&= -\|\cdot\|_1^* \left(\frac{X^T v}{\lambda}\right) \\
&= \begin{cases} 0 & \text{if } \left\|\frac{X^T v}{\lambda}\right\|_\infty \leqslant 1 \\ -\infty & \text{otherwise} \end{cases}
\end{aligned}$$

where $\|\cdot\|_1^*$ is the conjugate of $\|\cdot\|_1$.

Thus,

$$g(v) = \begin{cases} -\frac{1}{2}v^T v + v^T y & \text{if } \left\|\frac{X^T v}{\lambda}\right\|_\infty \leqslant 1 \\ -\infty & \text{otherwise} \end{cases}$$

Dual problem:

$$\begin{cases} \max -\frac{1}{2}v^T v + v^T y \\ \text{s.t. } \left\|\frac{X^T v}{\lambda}\right\|_\infty \leqslant 1 \end{cases}$$

$$\iff$$

$$\begin{cases} \min\ \frac{1}{2}v^T v - v^T y \\ \text{s.t. } \left\|\frac{X^T v}{\lambda}\right\|_\infty \leqslant 1 \end{cases}$$

$$\iff$$

$$\begin{cases} \min\ v^T \left(\frac{1}{2}I_n\right) v - y^T v \\ \text{s.t. } \left\|\frac{X^T v}{\lambda}\right\|_\infty \leqslant 1 \end{cases}$$

Since:

$$\left\|\frac{X^T v}{\lambda}\right\|_\infty = \max_i \left|\left(\frac{X^T v}{\lambda}\right)_i\right| \leqslant 1 \Rightarrow \forall i, \left|\left(\frac{X^T v}{\lambda}\right)_i\right| \leqslant 1 \text{ or } \pm \left(X^T v\right)_i \leqslant \lambda$$

we can rewrite it to:

$$\begin{pmatrix} X^T \\ -X^T \end{pmatrix} v \preceq \lambda \mathbf{1}_{2d}$$

Therefore, the dual problem can be written as a quadratic problem:

$$\begin{aligned} &\min v^T Q v + p^T v \\ &\text{s.t. } Av \preceq b \end{aligned}$$

where:

$$\begin{aligned} Q &= \frac{1}{2} I_n \\ p &= -y \\ A &= \begin{pmatrix} X^T \\ -X^T \end{pmatrix} \\ b &= \lambda \mathbf{1}_{2d} \end{aligned}$$

Question 2

Implement the barrier method to solve the quadratic problem:

For the centering step, we use Newton method to solve the following problem:

$$\min c(v) = t(v^T Q v + p^T v) - \sum_{i=1}^{2d} \log(-A_i v + b_i)$$

where A_i is the ith row of A.

The first and second derivative of the function:

$$\nabla c(v) = t(2Qv + p) - \sum_{i=1}^{2d} \frac{-A_i^T}{-A_i v + b_i}$$

$$\nabla^2 c(v) = 2tQ + \sum_{i=1}^{2d} \frac{A_i^T A_i}{(-A_i v + b_i)^2}$$

[1]:

```
import numpy as np

def backtracking_line_search(Q, p, A, b, t, v, derive1, delta_v, alpha, beta):

    # Centering problem function
    c = lambda v_: np.nan if np.any(np.dot(-A, v_) + b <= 0) else t*(np.dot(np.
↪dot(v_.T, Q), v_) + np.dot(p.T, v_)) - np.sum(np.log(np.dot(-A, v_) + b))

    st_sz = 1
    while True:
        if np.isnan(c(v + (st_sz*delta_v))) or c(v + (st_sz*delta_v)) < c(v) +␣
↪(alpha * st_sz * np.dot(derive1.T, delta_v)):
            break
        st_sz *= beta
    return st_sz
```

[2]:

```
def centering_step(Q, p, A, b, t, v0, eps):
    v = np.array(v0)
    v_seq = [np.array(v0)]

    while True:

        # First and second derivative of the centering problem function
        derive1 = t*(2*np.dot(Q, v) + p) - np.sum(-A.T / (np.dot(-A, v) + b),␣
↪axis=1)
        derive2 = 2*t*Q + np.sum(np.diag(np.dot(A, A.T)) / (np.dot(-A, v) +␣
↪b)**2)

        # Newton step and decrement
        delta_v = np.dot(-np.linalg.inv(derive2), derive1)
        lambda_2 = np.dot(np.dot(derive1.T, np.linalg.inv(derive2)), derive1)

        # Stopping criterion
        if lambda_2 / 2 <= eps:
            break

        step_sz = backtracking_line_search(Q, p, A, b, t, v, derive1, delta_v,␣
↪alpha=0.1, beta=0.5)
        v += (step_sz * delta_v)
        v_seq.append(np.array(v))

    return v_seq
```

[3]:

```
def barr_method(Q, p, A, b, v0, eps):
    v_seq = [np.array(v0)]
    v = np.array(v0)
```

```
    t_seq = [t0]
    t = t0

    newton_iter = 0

    while True:
        v_l = centering_step(Q, p, A, b, t, v, eps=1e-5)
        newton_iter += len(v_l)-1
        v = v_l[-1]
        v_seq.append(np.array(v))
        if A.shape[0] / t < eps:
            break
        t *= mu
        t_seq.append(t)

    return v_seq, t_seq, newton_iter
```

Question 3

Test the functions on randomly generated matrix X and observations y with $\lambda = 10$ and $\mu = 2, 15, 50, 100, \ldots$:

[4]:
```
from sklearn.datasets import make_regression

n = 100
d = 7

X, y = make_regression(n_samples=n, n_features=d, random_state=42)
lmda = 10

Q = 0.5 * np.identity(n)
p = -y
A = np.concatenate((X.T, -X.T), axis=0)
b = lmda * np.ones(2*d)
v0 = np.zeros(n)
eps = 1e-5
t0 = 2
mu_list = [2, 15, 50, 100, 500, 1000]
```

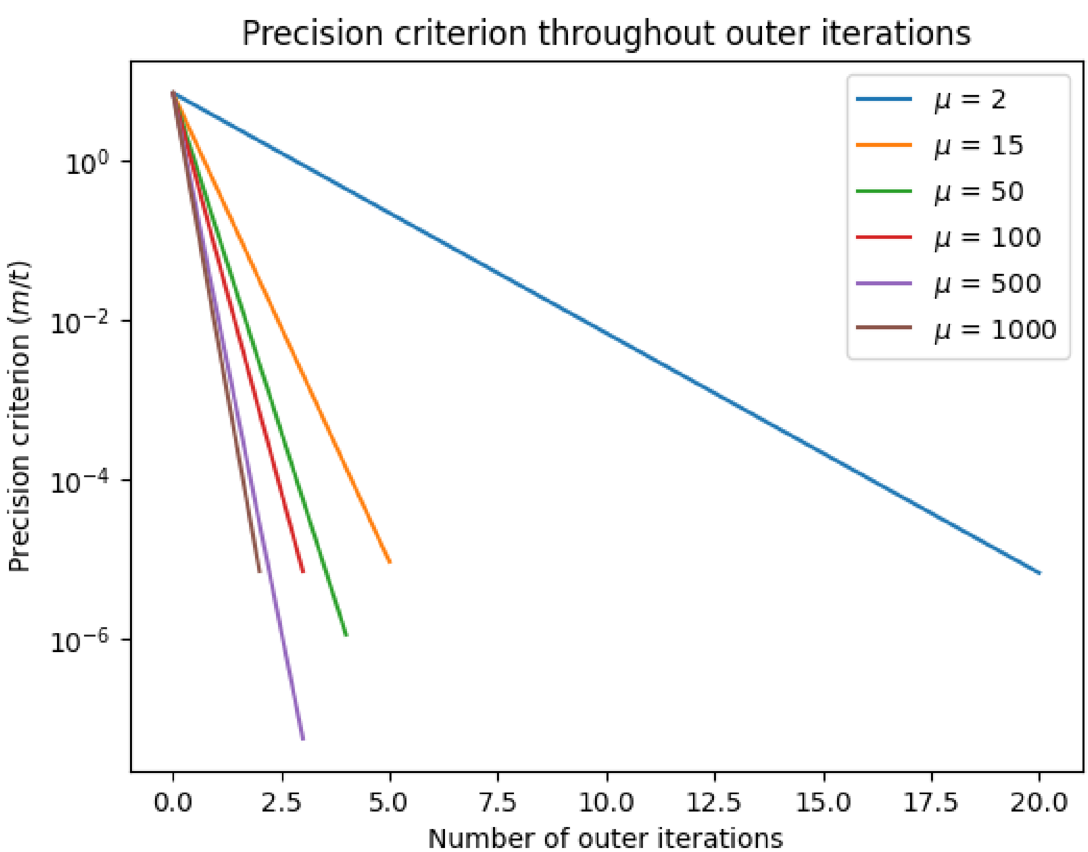
[5]:
```
dual_obj = lambda Q,p,v_: np.dot(np.dot(v_.T, Q), v_) + np.dot(p.T, v_)

opt_v_list = []
newton_iter_list = []
precision_lists = []
gap_lists = []

for mu in mu_list:
```

```
    v_seq, t_seq, newton_iter = barr_method(Q, p, A, b, v0, eps)

    newton_iter_list.append(newton_iter)
    opt_v_list.append(np.array(v_seq[-1]))
    precision_lists.append([A.shape[0] / t for t in t_seq])
    gap_lists.append([dual_obj(Q,p,v) - dual_obj(Q,p,v_seq[-1]) for v in v_seq])
```

Plot of precision criterion in semilog scale:

[6]:
```
import matplotlib.pyplot as plt

for precision_list in precision_lists:
    plt.plot(precision_list)
plt.semilogy()
plt.xlabel('Number of outer iterations')
plt.ylabel('Precision criterion $(m/t)$')
plt.title('Precision criterion throughout outer iterations')
plt.legend(['$\mu$ = {}'.format(mu) for mu in mu_list])
```

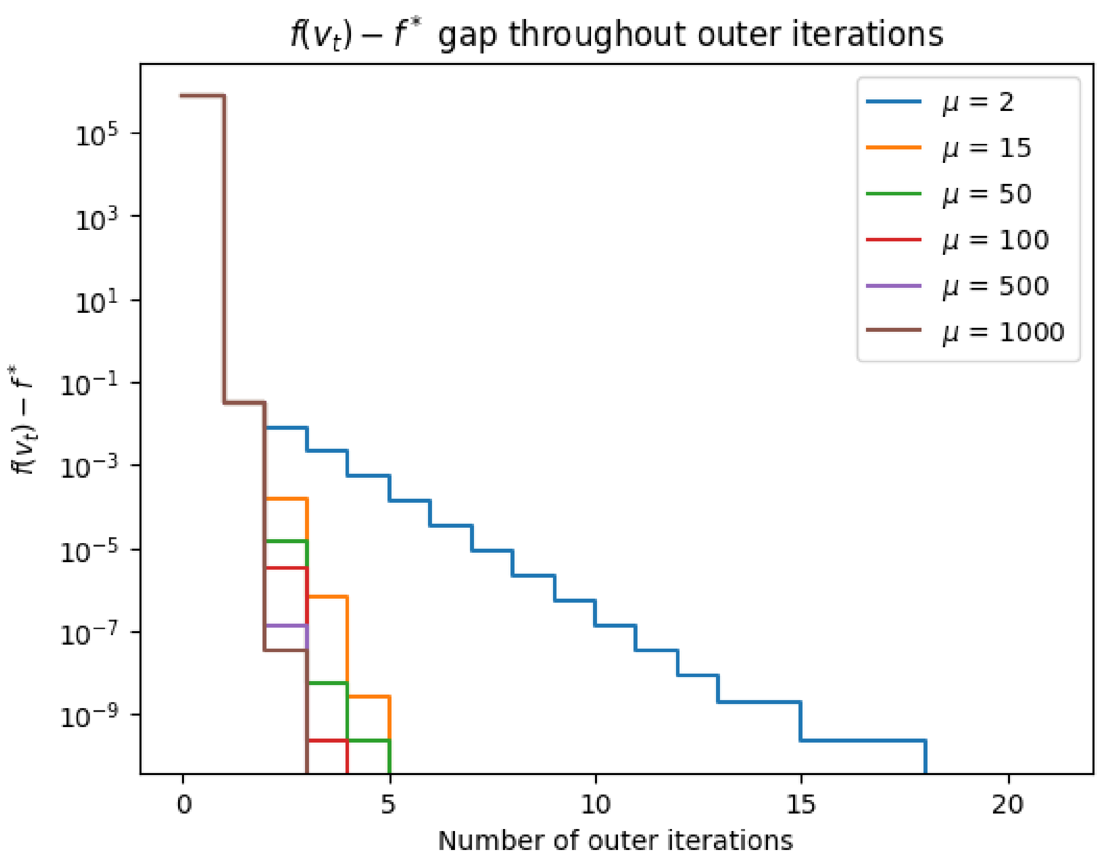
[6]: <matplotlib.legend.Legend at 0x7d9d997fb2b0>



Plot of $f(v_t) - f^*$ gap in semilog scale:

[7]:
```
for gap_list in gap_lists:
    plt.step(range(len(gap_list)), gap_list, where='post')
plt.semilogy()
plt.xlabel('Number of outer iterations')
plt.ylabel('$f(v_t) - f^*$')
plt.title('$f(v_t) - f^*$ gap throughout outer iterations')
plt.legend(['$\mu$ = {}'.format(mu) for mu in mu_list])
```

[7]:
```
<matplotlib.legend.Legend at 0x7d9d686d0370>
```

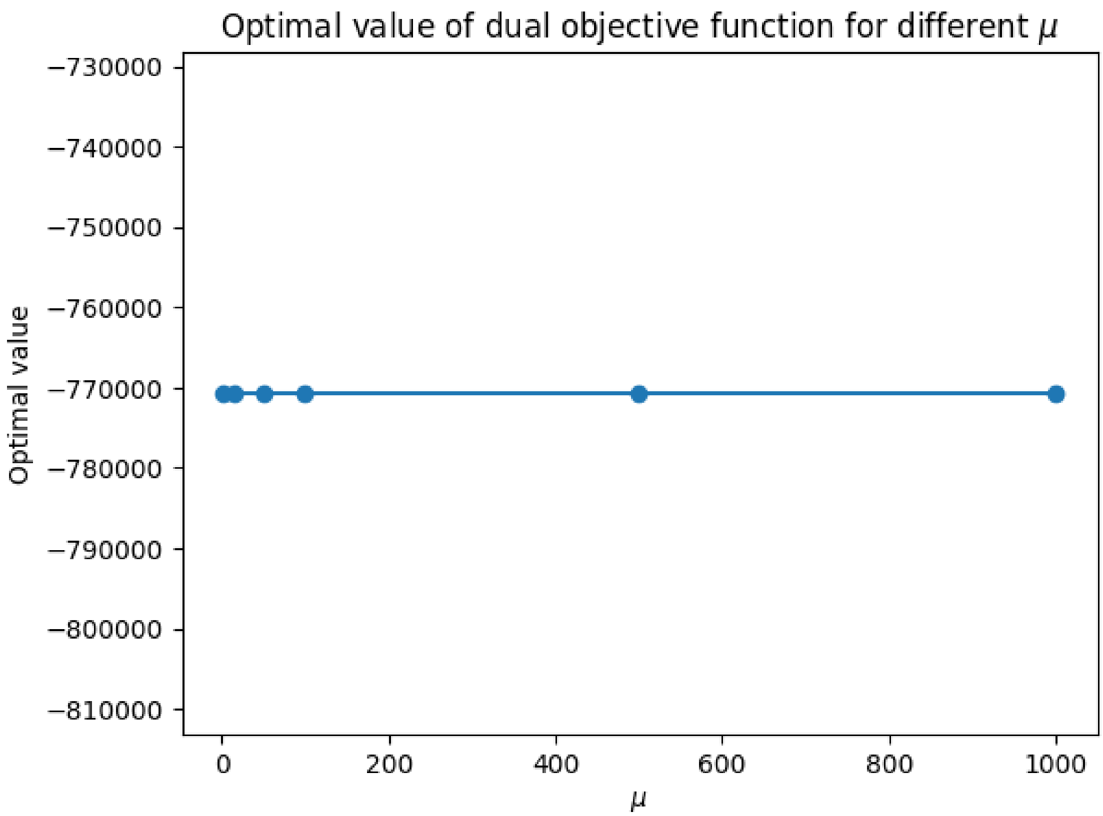


The graph above shows that small values of μ take a large number of outer iterations to converge, which corresponds to few Newton iterations. The opposite is true for large values of μ. The best choice of μ makes a trade-off between the number of outer iterations and Newton iterations, in this case it would be $\mu = 15$.

[8]:
```
plt.plot(mu_list, [dual_obj(Q,p,opt_v) for opt_v in opt_v_list], marker='o')
plt.xlabel('$\mu$')
plt.ylabel('Optimal value')
```

```
plt.title('Optimal value of dual objective function for different $\mu$')
```

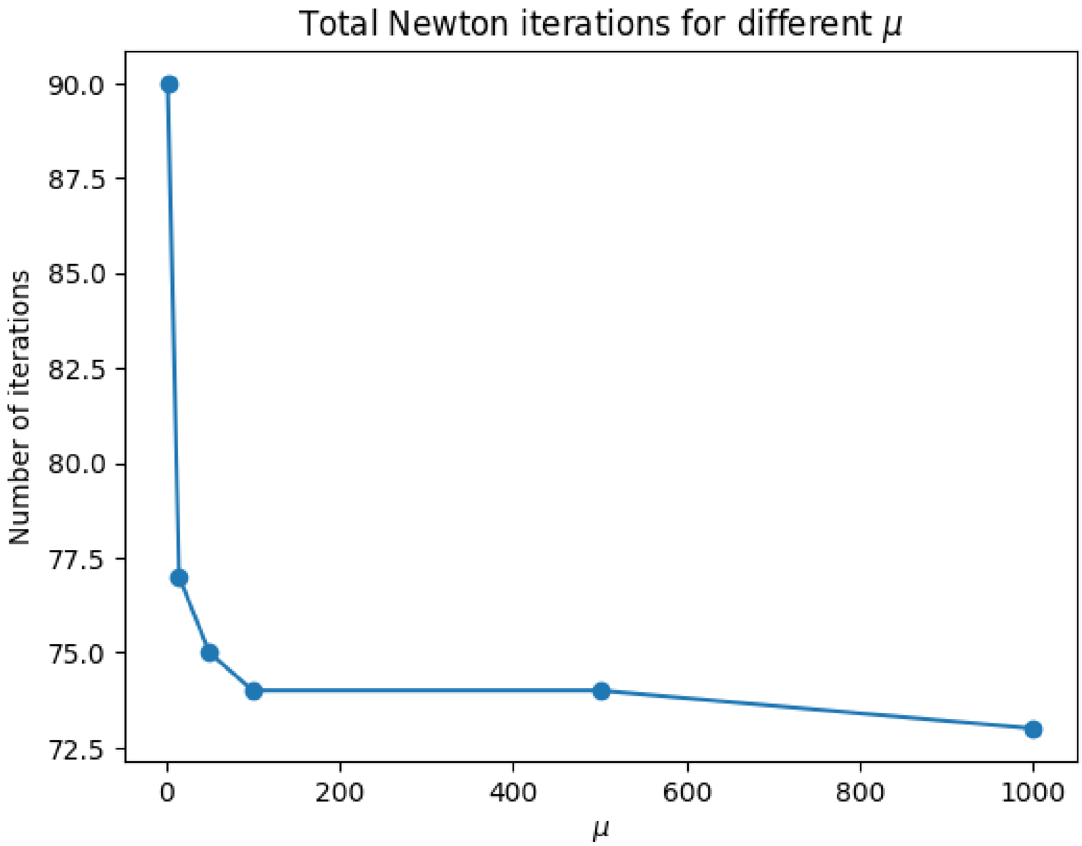
[8]: Text(0.5, 1.0, 'Optimal value of dual objective function for different $\\mu$')



From the graph above, we can see that the optimal value of the dual objective function stays approximately the same for different values of μ. Thus, μ has no impact on w.

[9]:
```
plt.plot(mu_list, newton_iter_list, marker='o')
plt.xlabel('$\mu$')
plt.ylabel('Number of iterations')
plt.title('Total Newton iterations for different $\mu$')
```

[9]: Text(0.5, 1.0, 'Total Newton iterations for different $\\mu$')



The graph above shows that the higher the value of μ, the less total Newton iterations it takes to reach convergence.