1 Question 1

The problem to solve is:

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

where $w \in \mathbb{R}^d$, $X = \begin{pmatrix} x_1^T \\ \vdots \\ x_n^T \end{pmatrix} \in \mathbb{R}^{n\times d}$, $y = (y_1, \dots, y_n)^T \in \mathbb{R}^n$, and $\lambda > 0$ is the regularization parameter.

1.1 Deriving the dual problem of LASSO and reformulating it as a quadratic problem

The dual problem can be expressed as:

$$\text{minimize } v^T Q v + p^T v \quad \text{subject to } Av \preceq b \tag{QP}$$

where $v \in \mathbb{R}^n$ and $Q \succeq 0$.
To derive the dual problem, we start by introducing an equality constraint to the original formulation:

$$\begin{gathered} \min_{z,w} \frac{1}{2}\|z\|_2^2 + \lambda\|w\|_1 \\ \text{subject to } z = Xw - y \end{gathered} \tag{1}$$

The Lagrangian and the dual function are given as follows:

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

$$g(\mu) = \inf_{w,z} L(w, z, \mu) = \inf_z \left(\frac{1}{2}\|z\|_2^2 + \mu^T z\right) + \inf_w \left(\lambda\|w\|_1 - \mu^T Xw\right) + \mu^T y \tag{3}$$

The infimum with respect to z is calculated using the first-order condition $\nabla f(z) = 0$, as the function is convex in z:

$$\nabla f(z) = z + \mu = 0 \quad \implies z = -\mu \tag{4}$$

Next, to compute the infimum with respect to w, we use the result of Exercise 2.1 from a previous homework. The conjugate function of $f(x) = \|x\|_1$ is:

$$f^*(y) = \begin{cases} 0 & \text{if } \|y\|_\infty \le 1 \\ \infty & \text{otherwise} \end{cases}$$

Rewriting the infimum as a supremum, we have:

$$\inf_w(\lambda\|w\|_1 - \mu^T Xw) = -\sup_w(\mu^T Xw - \lambda\|w\|_1) = \lambda f^*\left(\frac{X^T\mu}{\lambda}\right) \tag{6}$$

Thus, the dual function becomes:

$$g(\mu) = -\frac{1}{2}\mu^T\mu + \lambda f^*\left(\frac{X^T\mu}{\lambda}\right) + \mu^T y \tag{7}$$

Finally, the dual problem is formulated as:

$$\begin{gathered} \max_\mu \mu^T y - \frac{1}{2}\mu^T\mu \\ \text{subject to } \|X^T\mu\|_\infty \le \lambda \end{gathered} \tag{8}$$

This is equivalent to the following quadratic programming (QP) problem:

$$\min_{\mu} \frac{1}{2}\mu^T\mu - \mu^T y \quad \text{subject to } \|X^T y\|_\infty \leq \lambda \tag{9}$$

The terms in this formulation are as follows:

- $Q = \frac{1}{2} I_{d\times n}$
- $p = y$
- $b \in \mathbb{R}^{2d}$, $b_i = \lambda \forall i$
- $A \in \mathbb{R}^{2d\times n} = (X, -X)^T$

It is important to note that the dimensionality $2d$ arises due to the infinity norm, which enforces two constraints per coordinate of $X^T v$: $(X^T)_i v \leq \lambda$ and $(X^T)_i v \geq -\lambda$.

2 Question 2

To solve the optimization problem, we define the function f, its gradient ∇f, and its Hessian $H(f)$ as follows:

2.1 Definition of f

The function f is defined as:

$$f(v, Q, p, b, A, t) = t\left(v^\top Q v + p^\top v\right) - \sum_{i=1}^{2n} \log\left(b_i - (A^\top v)_i\right),$$

where:

- $A \in \mathbb{R}^{n\times 2d}$
- $Q \in \mathbb{R}^{n\times n}$
- $v, p \in \mathbb{R}^n$
- $b \in \mathbb{R}^{2n}$
- $t \in \mathbb{R}$

2.2 Gradient of f

The gradient ∇f is given by:

$$\nabla f(v, Q, p, b, A, t) = t\,(2Qv + p) - A\left(\frac{1}{b - A^\top v}\right),$$

where $\frac{1}{b-A^\top v}$ is a vector of dimension $2n$ with each element:

$$\left[\frac{1}{b - A^\top v}\right]_i = \frac{1}{b_i - (A^\top v)_i}.$$

2.3 Hessian of f

The Hessian $H(f)$ is expressed as:

$$H(f)(v, Q, b, A, t) = t \cdot 2Q + A \cdot \text{diag}\left(\frac{1}{(b - A^\top v)^2}\right) \cdot A^\top,$$

where $\text{diag}\left(\frac{1}{(b-A^\top v)^2}\right)$ is a diagonal matrix with diagonal elements:

$$\text{diag}\left(\frac{1}{(b - A^\top v)^2}\right)_{ii} = \frac{1}{(b_i - (A^\top v)_i)^2}.$$

HW 3 Convex Optimisation

Question 2

```
## Shape des variables

# A --> R^n*2d
# Q --> R^n**2
# v, p --> R^n
# b --> R^2n

###

import numpy as np
import cvxpy as cp
import warnings
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm


def value_g(Q, p, A, b, t, v):
    value = t * (v.T[np.newaxis] @ Q @ v + p.T[np.newaxis] @ v) -
np.sum(np.log(b - A @ v))
    return value[0]

def value_dual(Q, p, v):
    return v.T[np.newaxis] @ Q @ v + p.T[np.newaxis] @ v

def gradiant_g(Q, p, A, b, t, v):
    gradient = 2 * t * (Q @ v) + t * p + np.sum(A * np.reciprocal(b -
A @ v)[:, np.newaxis], axis=0)
    return gradient

def hessian_g(Q, A, b, t, v):
    sum_term = np.power(b - A @ v, 2)
    hessian_sum = A[0][np.newaxis].T @ A[0][np.newaxis] / sum_term[0]
    for i in range(1, 2 * d):
        hessian_sum += A[i][np.newaxis].T @ A[i][np.newaxis] /
sum_term[i]
    hessian = 2 * t * Q + hessian_sum
    return hessian

def line_search(v, dv, alpha=0.5, beta=0.9):
    t = 1
    while t > 1e-6:
        objective_v = value_g(Q, p, A, b, t, v)
        objective_dv = value_g(Q, p, A, b, t, v + t * dv)
```

```
        gradient_v = gradiant_g(Q, p, A, b, t, v)

        criterion = objective_v + alpha * t * (gradient_v.T @ dv)
        stopping_criterium = (objective_dv < criterion)

        if stopping_criterium:
            break

        if np.any(b - A @ (v + t * dv) <= 0):
            return t

        t *= beta

    return t

def centering_step(Q, p, A, b, t, v0, eps, max_iter=500):
    nb_iter = 0
    stopping_criterium = False
    v = v0.copy()
    v_n = [v0]
    i = 0

    while not stopping_criterium and i < max_iter:
        i += 1
        grad_g = gradiant_g(Q, p, A, b, t, v)
        hess_g = hessian_g(Q, A, b, t, v)
        delta_v = np.linalg.pinv(hess_g) @ grad_g
        step = line_search(v, delta_v)
        v = v - step * delta_v
        v_n.append(v)
        lambda2 = grad_g.T @ delta_v
        stopping_criterium = (lambda2 / 2 <= eps)
        nb_iter += 1

    return v_n, nb_iter

def barr_method(Q, p, A, b, v0, eps, mu=2, t=1, max_iter=500):
    nb_iter = 0
    stopping_criterium = False
    v = v0.copy()
    v_n = []
    m = A.shape[0]
    i = 0

    while not stopping_criterium and i < max_iter:
        i += 1
        v_n.append(v)
        v_all, nb_iter_centerStep = centering_step(Q, p, A, b, t,
v_n[-1], eps)
        v = v_all[-1]
```

```
        stopping_criterium = (m / t < eps)
        t *= mu
        nb_iter += nb_iter_centerStep

    return v_n, nb_iter
```

Question 3

```
np.random.seed(784)

def initialize_problem(n=4, d=3, lmbda=10):
    v0 = np.zeros(n)
    eps = 0.01
    X = np.random.rand(n, d)
    y = np.random.rand(n)
    Q = np.eye(n) / 2
    p = -y.copy()
    A = np.concatenate([X.T, -X.T])
    b = lmbda * np.ones(2 * d)
    return X, y, Q, p, A, b, v0, eps
```

In the next cell, the calculation of the sequences for v is done for different values of μ : 2, 15, 50, 100, 250, 500 and 1000.

Then for each of the obtained sequences, the associated dual value is calculated.

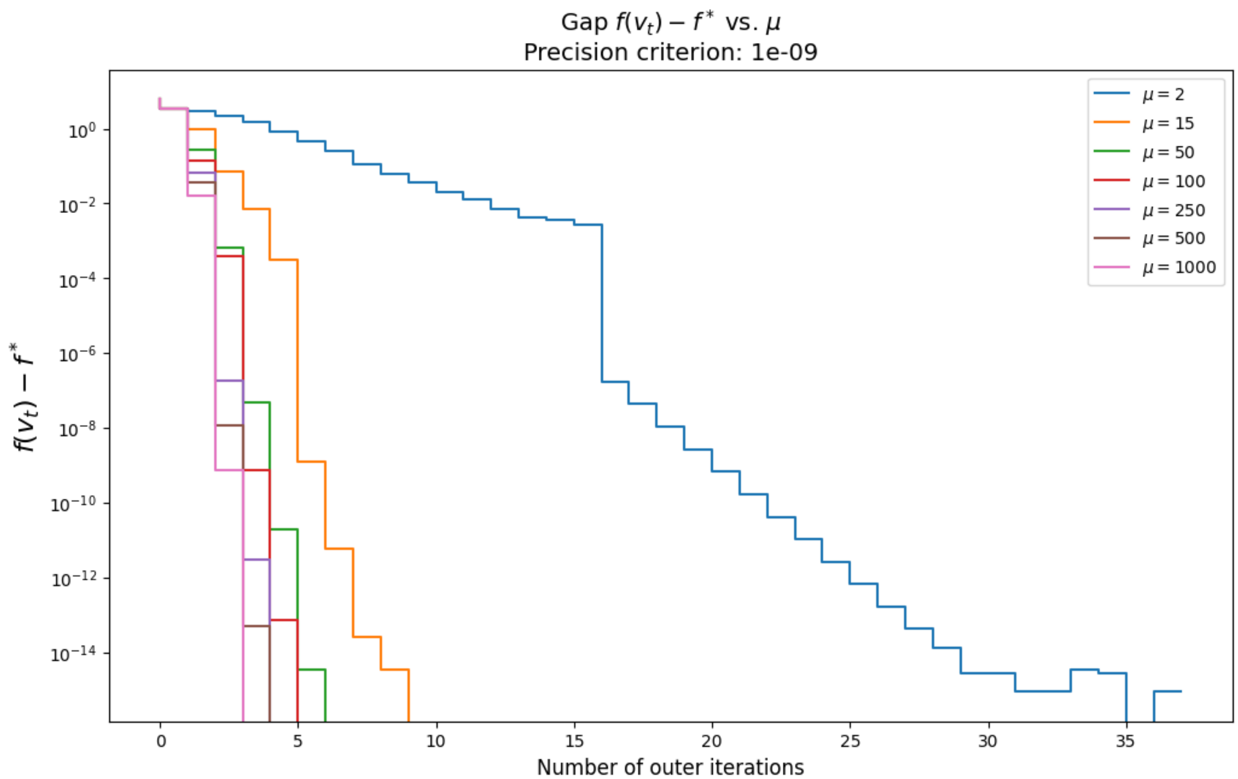
```
np.random.seed(2022)

# Initialisation des paramètres
n, d = 40, 50
X, y, Q, p, A, b, v0, eps = initialize_problem(n, d)
eps = 1e-9

mu_values = [2, 15, 50, 100, 250, 500, 1000]
results = []

# Calcul des séquences pour différentes valeurs de mu
for mu in tqdm(mu_values):
    v_sequence, iterations = barr_method(Q, p, A, b, v0, eps=eps,
mu=mu, t=1)
    results.append(v_sequence)

# Calcul des valeurs du dual pour toutes les séquences
dual_values = [[value_dual(Q, p, v) for v in results[i]] for i in
range(len(results))]

  0%|
| 0/7 [00:00<?, ?it/s]C:\Users\DAO.EZSPACE\AppData\Local\Temp\
```

```
ipykernel_20604\83268961.py:2: RuntimeWarning: invalid value
encountered in log
  value = t * (v.T[np.newaxis] @ Q @ v + p.T[np.newaxis] @ v) -
np.sum(np.log(b - A @ v))
100%|██████████████████████████████████████████████████| 7/7 [00:52<00:00,  7.50s/it]
```

Once all sequences are obtained for the different μ values tested, let's represent the gap $f(v) - f^{¿}$ versus the number of iterations.

```python
plt.figure(figsize=(12, 7))

iteration_counts, objective_gaps = [], []

for mu in mu_values:
    v_list, iterations = barr_method(Q, p, A, b, v0, eps=eps, mu=mu,
t=1)
    iteration_counts.append(iterations)
    gap_list = []

    best_value = min(value_dual(Q, p, v) for v in v_list)

    for v in v_list:
        gap = value_dual(Q, p, v) - best_value
        gap_list.append(gap[0])

    plt.step(range(len(gap_list)), gap_list)

plt.semilogy()
plt.xlabel('Number of outer iterations', fontsize=12)
plt.ylabel('$f(v_t) - f^*$', fontsize=16)
plt.title(f'Gap $f(v_t) - f^*$ vs. $\\mu$\nPrecision criterion:
{eps}', fontsize=14)
plt.legend([f'$\\mu = {x}$' for x in mu_values], loc='best')
plt.show()
```

```
C:\Users\DAO.EZSPACE\AppData\Local\Temp\ipykernel_20604\83268961.py:2:
RuntimeWarning: invalid value encountered in log
  value = t * (v.T[np.newaxis] @ Q @ v + p.T[np.newaxis] @ v) -
np.sum(np.log(b - A @ v))
```



We conclude that **μ = 1000** is the fastest and the best value.

Thanks for reading Thomas Gravier