NGUIMATSIA TIOFACK Franki

Question 1 : Derivation of the Dual Problem for LASSO

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

We can reformulate the LASSO problem as :

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

The Lagrangian is then defined as :

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

with $\nu \in \mathbb{R}^n$.

This leads to the dual function :

$$g(\nu) = \inf_{w,v}\left[\frac{1}{2}\|v\|_2^2 + \lambda\|w\|_1 + \nu^T(v + y - Xw)\right]$$

Breaking it down further :

$$\begin{aligned} g(\nu) &= \inf_v\left[\frac{1}{2}\|v\|_2^2 + \nu^T v\right] + \inf_w\left[\lambda\|w\|_1 - \nu^T Xw\right] + \nu^T y \\ &= \inf_v\left[\frac{1}{2}\|v\|_2^2 + \nu^T v\right] - \lambda\sup_w\left[\left(\frac{1}{\lambda}X^T\nu\right)^T w - \|w\|_1\right] + \nu^T y \end{aligned}$$

Since $\frac{1}{2}\|v\|_2^2 + \nu^T v$ is convex and differentiable, we find that :

$$\nabla_v\left(\frac{1}{2}\|v\|_2^2 + \nu^T v\right) = v + \nu$$

Thus, the optimal v^* satisfying $v + \nu = 0$ is $v^* = -\nu$. (2)

Utilizing results from Exercise 2 of Homework 2 :

$$\sup_w\left[\left(\frac{1}{\lambda}X^T\nu\right)^T w - \|w\|_1\right] = \begin{cases} 0 & \text{if } \|\frac{1}{\lambda}X^T\nu\|_\infty \leq 1, \\ +\infty & \text{otherwise.} \end{cases}$$

Finally, the dual of the LASSO problem can be expressed as :

$$\max_\nu \; -\frac{1}{2}\|\nu\|_2^2 + \nu^T y$$

$$\text{s.t. } \|X^T\nu\|_\infty \leq \lambda$$

Rewriting the constraint, we get :

$$\|X^T\nu\|_\infty \leq \lambda \iff \begin{pmatrix} X^T \\ -X^T \end{pmatrix} \nu \leq \lambda \mathbf{1}_{2d}$$

where $\mathbf{1}_{2d} \in \mathbb{R}^{2d}$ is defined such that $\forall i \in \{1, \ldots, d\} : (\mathbf{1}_{2d})_i = 1$.

Therefore, the dual LASSO problem can be reformulated in the quadratic form as :

$$\min_\nu \frac{1}{2}\nu^T\nu - \nu^T y$$

$$\text{s.t. } \begin{pmatrix} X^T \\ -X^T \end{pmatrix} \nu \preceq \lambda \mathbf{1}_{2d}$$

By setting $Q = \frac{1}{2}I_n$, $A = \begin{pmatrix} X^T \\ -X^T \end{pmatrix}$ and $p = -y$, we obtain the following dual problem :

$$\min_v v^T Q v + p^T v$$

$$s.t.\ Av \preceq b$$

Question 2 : Basis derivation for Barrier method implementation

Let consider the following notation :

$$A = \begin{pmatrix} a_1^T \\ a_2^T \\ \vdots \\ a_{2d}^T \end{pmatrix}$$

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

where each a_i is a vector in $\mathbb{R}^n$ for i ranging from 1 to $2d$.

From this, the gradient of $f_t(v)$ with respect to v is obtained as :

$$\nabla_v f_t(v) = 2tQv + tp + \sum_{i=1}^{2d} \frac{a_i}{(b_i - a_i^T v)}$$

Furthermore, the second derivative, or the Hessian, of $f_t(v)$ with respect to v is :

$$\nabla_v^2 f_t(v) = 2tQ + \sum_{i=1}^{2d} \frac{a_i a_i^T}{(b_i - a_i^T v)^2}$$

Question 3 : Graphic and comments

Influence of μ on w

Problem (1), being convex and strictly feasible, satisfies the conditions for strong duality. Furthermore, by applying the Karush-Kuhn-Tucker (KKT) conditions, we established that $Xw^* - y = -v^*$ (as detailed in equation (2)). Consequently, the optimal solution w^* can be expressed as $w^* = X^+(y - v^*)$, where X^+ denotes the pseudoinverse of X.

In the bellow figure, we have depicted the norms of the differences between the vectors w for two consecutive values of μ. It is observed that μ has almost no impact on w.

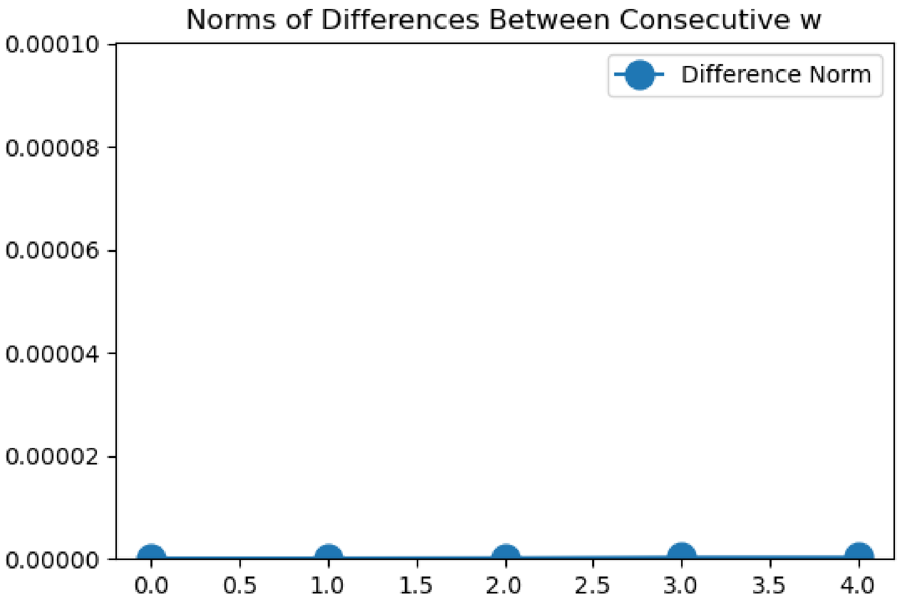


FIGURE 1 – Impact of Parameter μ on the Variation of Vectors w

Influence of μ on the convergence and the choice of μ

As display in the figure below, when μ is set to a low value, fewer steps are required in each outer iteration, yet the overall number of outer iterations increases. Conversely, a higher μ value reduces the number of outer iterations but increases the steps needed in each one. Therefore, the ideal μ value represents a balance between these two factors. As illustrated in the previous example, a μ value of either 50 or 100 might be optimal.

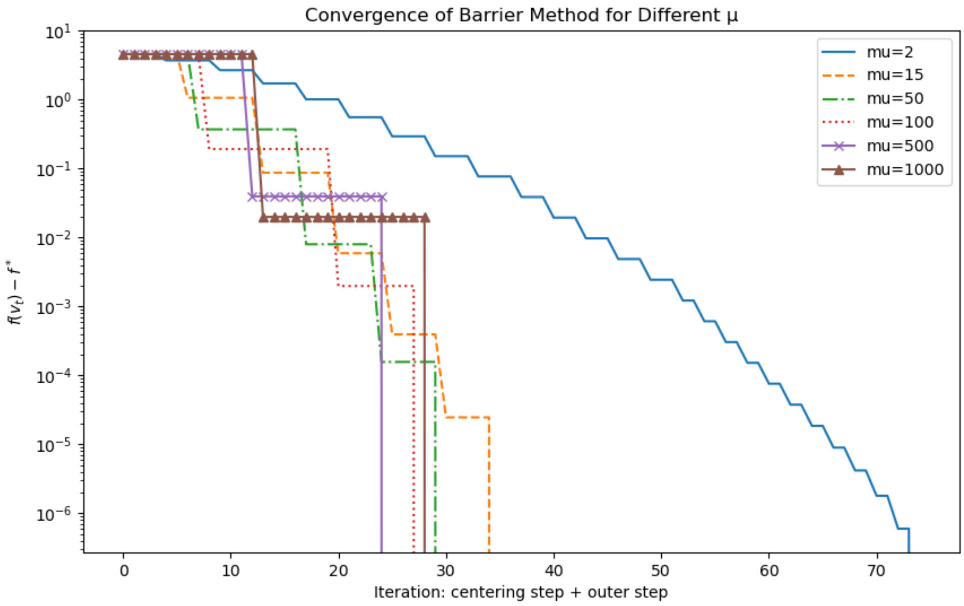


FIGURE 2 – Convergence of the Barrier Method for Different μ.

Name : **NGUIMATSIA TIOFACK Franki**

Code for Homework 3

For mathematical prove and comment, see the second pdf

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [14]:
```
import pandoc
```

In [2]:
```
def centering_step(Q, p, A, b, t, v0, eps):
    def f(v):
        return t * (v.T @ Q @ v + p.T @ v) - np.sum(np.log(b - A @ v))

    def grad_f(v):
        return t * (2 * Q @ v + p) + A.T @ (1 / (b - A @ v))

    def hess_f(v):
        diag_terms = 1 / ((b - A @ v) ** 2)
        return 2 * t * Q + A.T @ np.diag(diag_terms) @ A

    v = v0
    v_seq = []
    while True:
        grad = grad_f(v)
        hess = hess_f(v)
        delta_v = np.linalg.solve(hess, -grad)
        lambda_sq = -grad.T @ delta_v
        if lambda_sq / 2 <= eps:
            break
        # Backtracking line search
        alpha = 0.2
        beta = 0.5
        st = 1
        while np.min(b - A @ (v + st * delta_v)) <= 0 or f(v + st * delta_v) >= f(v
            st *= beta
        v += st * delta_v
        v_seq.append(v)
    return v_seq
```

In [3]:
```
def barr_method(Q, p, A, b, v0, eps, mu):
    m = len(b)
    t = 0.5
    v = v0
    v_seq = []
    num_New_step = []
    while m / t >= eps:
        centering = centering_step(Q, p, A, b, t, v, eps)
        v = centering[-1]
        v_seq.append(list(v))
        num_New_step.append(len(centering))
        t *= mu
    return v_seq, num_New_step
```

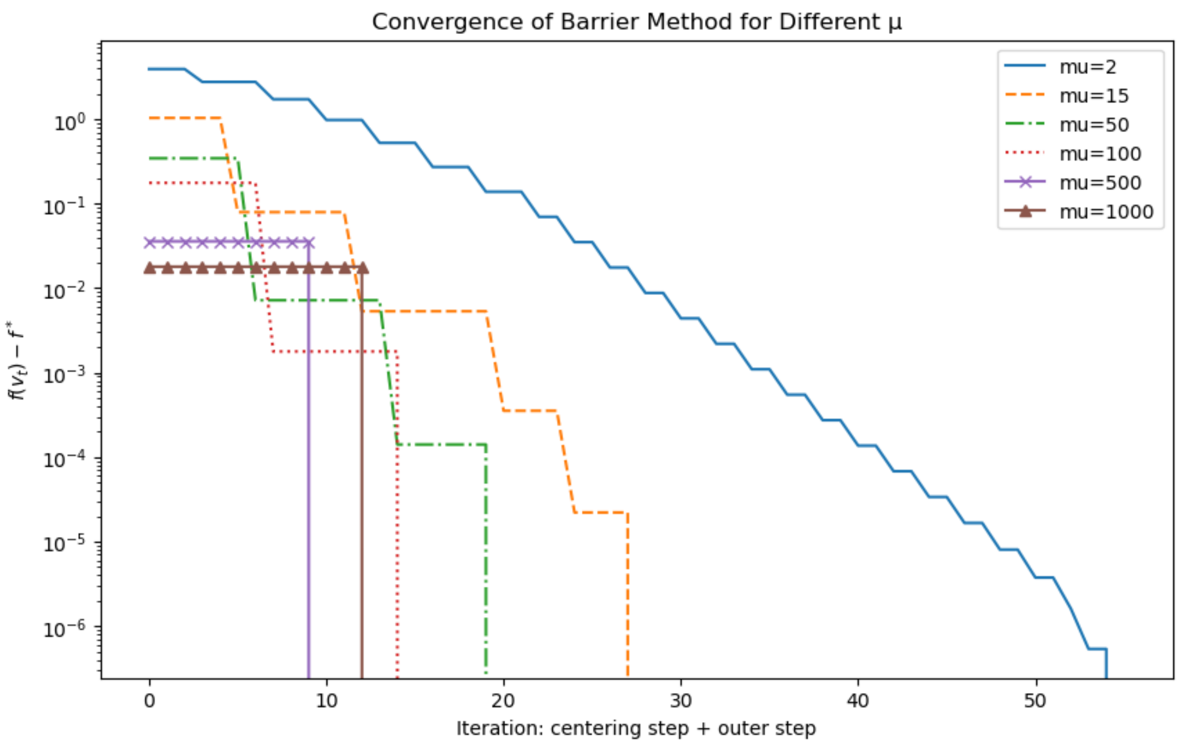
In [4]:
```
n, d = 100, 10  # Dimensions des données
X = 7*np.random.randn(n, d)
```

```python
y = -5 + 1.5*np.random.randn(n)
lambda_val = 10
```

In [5]:
```python
Q = 0.5 * np.eye(n)
p = - y
A = np.concatenate([X.T, -X.T])
b = lambda_val * np.ones(2 * d)
```

In [6]:
```python
def objective_function(v, Q, p):
    return  v.T @ Q @ v + p.T @ v
```

In [7]:
```python
mu_values = [2, 15, 50, 100, 500, 1000]
line_styles = ['-', '--', '-.', ':', 'x-', '^-']


plt.figure(figsize=(10, 6))

for mu, style in zip(mu_values, line_styles):
    v0 = np.zeros(len(p))
    eps = 1e-6
    v_seq, num_New_step = barr_method(Q, p, A, b, v0, eps, mu)

    f_vals = [objective_function(np.array(v), Q, p) for v in v_seq]
    f_valss = []
    for i in range(1, len(f_vals)-1):
        f_valss.extend([f_vals[i]] * num_New_step[i])
    f_valss.extend([f_vals[-1]])

    plt.semilogy(np.array(f_valss) - f_vals[-1], style, label=f"mu={mu}")

plt.xlabel("Iteration: centering step + outer step")
plt.ylabel('$f(v_t) - f^*$')
plt.title("Convergence of Barrier Method for Different μ")
plt.legend()

plt.savefig("convergence_barrier_method.png")
plt.show()
```



Impact of μ on w

See Pdf for explanaition $w^* = X^+(y - v^*)$

In [8]:
```
mu_values = [2, 15, 50, 100, 500, 1000]
ws = []
for mu in mu_values:
    v_seq, num_New_step = barr_method(Q, p, A, b, v0, eps, mu)
    v = v_seq[-1]
    # Compute w from v*
    ws.append(list(np.dot(np.linalg.pinv(X),(y-v))))
```

In [9]:
```
# Convertir la liste des w en une matrice
W = np.array(ws)

# Calculer les différences entre les w consécutifs
diffs = np.diff(W, axis=0)

# Calculer la norme de chaque différence
norms = np.linalg.norm(diffs, axis=1)
```

In [10]:
```
plt.figure(figsize=(6, 4))
plt.plot(norms, marker='o', markersize=12, label="Difference Norm")
plt.ylabel('Difference Norm')
plt.title('Norms of Differences Between Consecutive w')

# Set the scale of the y-axis
plt.ylim([0, max(norms) + 0.0001])  # Adjust based on your data
plt.legend()
plt.savefig("w.png")
plt.show()
```