Master MVA
Convex Optimization

---

**Homework 3**: Algorithms Implementation

---

Victor Busa
victor.busa@ens-paris-saclay.fr

November 13, 2017

# 1 Interior Points Method

**Compute $\nabla_{\phi_t(x)}$ and $\nabla^2_{\phi_t(x)}$** Let's define $\phi_t(x)$ by:

$$\phi_t(x) = t\left(\frac{1}{2}x^\intercal Qx + p^\intercal x\right) + \mathcal{B}(b - Ax)$$

then if we note $A = \begin{bmatrix} — & a_1^T & — \\ — & a_2^T & — \\ & \vdots & \\ — & a_m^T & — \end{bmatrix}, \in \mathbb{R}^{m\times n}$, we can compute $\nabla_{\phi_t(x)}$ as follow:

$$\begin{aligned}\nabla_{\phi_t(x)} &= t(p + Qx) - \sum_{i=1}^{m} \frac{-a_i}{b_i - a_i^\intercal x} \\ &= t(p + Qx) + \sum_{i=1}^{m} a_i \frac{1}{b_i - a_i^\intercal x} \\ &= t(p + Qx) + A^\intercal d\end{aligned}$$

Where we defined $d$, the vector in $\mathbb{R}^m$, s.t $d = \begin{pmatrix} 1/(b_1 - a_1^\intercal x) \\ \vdots \\ 1/(b_m - a_m^\intercal x) \end{pmatrix}$

Then, we can compute the Hessian of $\phi_t(x)$, as follow:

$$\begin{aligned}\nabla^2_{\phi_t(x)} &= tQ + \sum_{i=1}^{m} \frac{a_i a_i^\intercal}{(b_i - a_i^\intercal x)^2} \\ &= tQ + A^\intercal diag(d)^2 A\end{aligned}$$

Where we've reused the vector $d$ defined earlier.

**implementation of phi, grad and hess** See Python code

## 2 Newton Method

**implementation of dampedNewtonStep** See Python code

**implementation of dampedNewton** See Python code

**implementation of newtonLS** See Python code

## 3 Support Vector Machine Problem

### 3.1 Strictly feasible points for primal and dual

**a) For the dual:** the dual problem is:

$$\begin{aligned} \text{maximize} \quad & \mathbf{1}^\intercal \lambda - \frac{1}{2}\left\| \sum_{i=1}^{n} \lambda_i y_i x_i \right\|_2^2 \\ \text{subject to} \quad & 0 \leq \lambda \leq \frac{1}{\tau n} \end{aligned} \tag{SVM-D}$$

So, as we are maximizing over $\lambda$ only, we can take $\lambda$ that satisfies $0 < \lambda < 1/(\tau n)$, to have a strictly feasible point. For example one can use $\lambda = 1/(2\tau n)$ as a strictly feasible point for (SVM-D).

**b) For the primal:** the primal problem is:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{\tau n}\sum_{i=1}^{n} z_i + \frac{1}{2}\|w\|_2^2 \\ \text{subject to} \quad & y_i(w^\intercal x_i) \geq 1 - z_i, \quad i = 1, \ldots, n \\ & z \geq 0 \end{aligned} \tag{SVM-P}$$

Here, we are maximizing over both $w$ and $z$. We can see that if we define $Y = diag(y_1, \ldots, y_n)$, then: $(Xw)_i = \sum_{k=1}^{d} X_{ik} w_k = x_i^\intercal w = w^\intercal x_i$, so $y_i(w^\intercal x_i) = (YXw)_i$. And the strictly feasible points are such that $\forall i \in [1, n]$, $z_i + y_i(w^\intercal x_i) > 1$ and $z > 0$, so in particular, strictly feasible points must satisfy:

$$z + YXw > \mathbf{1}$$
$$z > 0$$

So, for example, one can use $z = c \times \mathbf{1}$ and $w = \mathbf{0}_d$ where $c$ is a constant with $c > 1$ as a strictly feasible point for (SVM-P).

### 3.2 Barrier method implementation

**a) Transform (SVM-D) and (SVM-P) as (QP)** Let's define $X = (x_1, \ldots, x_n) \in \mathbb{R}^{d \times n}$, $y \in \{-1, 1\}^n$ and $\tau > 0$. Let's first transform the primal problem into a QP.

In the primal problem, we have $x = \begin{pmatrix} w \\ z \end{pmatrix}$

and we want:

$$\frac{1}{2} x^\intercal Q x = \frac{1}{2} \|w\|_2^2$$

by identification it comes:

$$Q = \begin{pmatrix} I_{d,d} & 0_{d,n} \\ 0_{n,d} & 0_{n,n} \end{pmatrix}$$

Where the subscripts represent the sizes of the bloc matrices.

In the same way, we want:

$$p^\intercal x = p^\intercal \begin{pmatrix} w \\ z \end{pmatrix} = \frac{1}{\tau n} \mathbf{1}^\intercal z$$

And by identification, we have naturally:

$$p = \frac{1}{\tau n} \begin{pmatrix} 0_d \\ 1_n \end{pmatrix}$$

Finally, the constraint $Ax \leq b$ can be rewritten:

$$\begin{pmatrix} A11_{n,d} & A12_{n,n} \\ A21_{n,d} & A22_{n,n} \end{pmatrix} \begin{pmatrix} w_d \\ z_n \end{pmatrix} \leq \begin{pmatrix} b1_n \\ b2_n \end{pmatrix}$$

and by identification, we want:

$$\begin{cases} A_{11}w + A_{12}z \leq b_1 \\ A_{21}w + A_{22}z \leq b_2 \end{cases} \quad = \quad \begin{cases} YXw + z \geq \mathbf{1}_n \\ z \geq 0_n \end{cases}$$

So, it comes:

$$A = \begin{pmatrix} -YX & -I_{n,n} \\ 0_{d,n} & -I_{n,n} \end{pmatrix} \quad \text{and} \quad b = \begin{pmatrix} -1_n \\ 0_n \end{pmatrix}$$

Let's now transform the dual problem into a QP, in the dual problem we have $x = \lambda$ only, and as we are maximizing a quantity, **it terms of the optimal parameters**, maximizing f is the same as minimizing -f, so we want:

$$p^\intercal \lambda = -\mathbf{1}^\intercal \lambda \Rightarrow p = -\mathbf{1}_n$$

and we also want:

$$\frac{1}{2}\lambda^\intercal Q \lambda = \frac{1}{2}\left\| \sum_{i=1}^{n} \lambda_i y_i x_i \right\|_2^2$$

So let's compute $\lambda^\intercal Q \lambda$:

$$\begin{aligned} \lambda^\intercal Q \lambda &= \sum_{j=1}^{n} \lambda_j (Q\lambda)_j = \sum_{j=1}^{n} \lambda_j \sum_{k=1}^{n} Q_{j,k}\lambda_k \\ &= \sum_{j=1}^{n}\sum_{k=1}^{n} \lambda_j \lambda_k Q_{j,k} \end{aligned}$$

And, on the other hand:

$$\left\| \sum_{i=1}^{n} \lambda_i y_i x_i \right\|_2^2 = \sum_{j=1}^{n}\sum_{k=1}^{n} \lambda_j \lambda_k (y_j x_j^\intercal x_k y_k)$$

So, by identification $\forall (j,k), \quad Q_{j,k} = (y_j x_j^\intercal x_k y_k)$, And we can see that, if we define $Y = diag(y_1, \ldots, y_n)$, $Q = YXX^\intercal Y$ satisfies this equality, as:

$$Q_{j,k} = (YXX^\intercal Y)_{j,k} = \sum_{i=1}^{n} (YX)_{j,i}(X^\intercal Y)_{i,k}$$
$$= \sum_{i=1}^{n} y_k x_{j,i}(x^\intercal)_{i,k} y_k = y_j(\underbrace{\sum_{i=1}^{n} x_{j,i}x_{k,i}}_{x_j^\intercal x_k})y_k$$

So, we have:

$$Q = YXX^\intercal Y$$

Finally, we want:

$$0 \leq \lambda \leq \frac{1}{\tau n} \Leftrightarrow Ax \leq b$$
$$-\lambda \leq 0 \quad \text{and} \quad \lambda \leq 1/(\tau n) \Leftrightarrow Ax \leq b$$

And, by identification, ($x = \lambda$ here), it leads to:

$$A = \begin{pmatrix} -I_n \\ I_n \end{pmatrix} \quad \text{and} \quad b = \frac{1}{\tau n}\begin{pmatrix} 0_n \\ 1_n \end{pmatrix}$$

See Python Code for the implementation

**b) Barrier method implementation** See Python code

**3.3 Test on Iris Dataset** We can rewrite (SVM-P) as the follow:

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{n} z_i + \frac{\tau n}{2}\|w\|_2^2 \\ \text{subject to} \quad & y_i(w^\intercal x_i) \geq 1 - z_i, \quad i = 1, \ldots, n \\ & z \geq 0 \end{aligned} \tag{SVM-P'}$$

Such that $\frac{\tau n}{2}$ can be seen as the regularization parameter of the (SVM-P') problem where $n$ is the number of samples. Increasing $\tau$ will have for effect to increase the regularization parameter and hence to reduce overfitting. However, it does that at the expense of adding bias to the estimate. Here, the experiment [3.1] had shown that we seek to have $\tau$ to be around $10^{-3}$ to $10^{-1}$. For the calculation I chose $\mu = 20$ and $tol = 10^{-6}$.

I trained the model on 80% of the data of the iris dataset using the 2 last classes only (Iris-versicolor and Iris-virginica) and I computed the accuracy on the remaininig 20%.

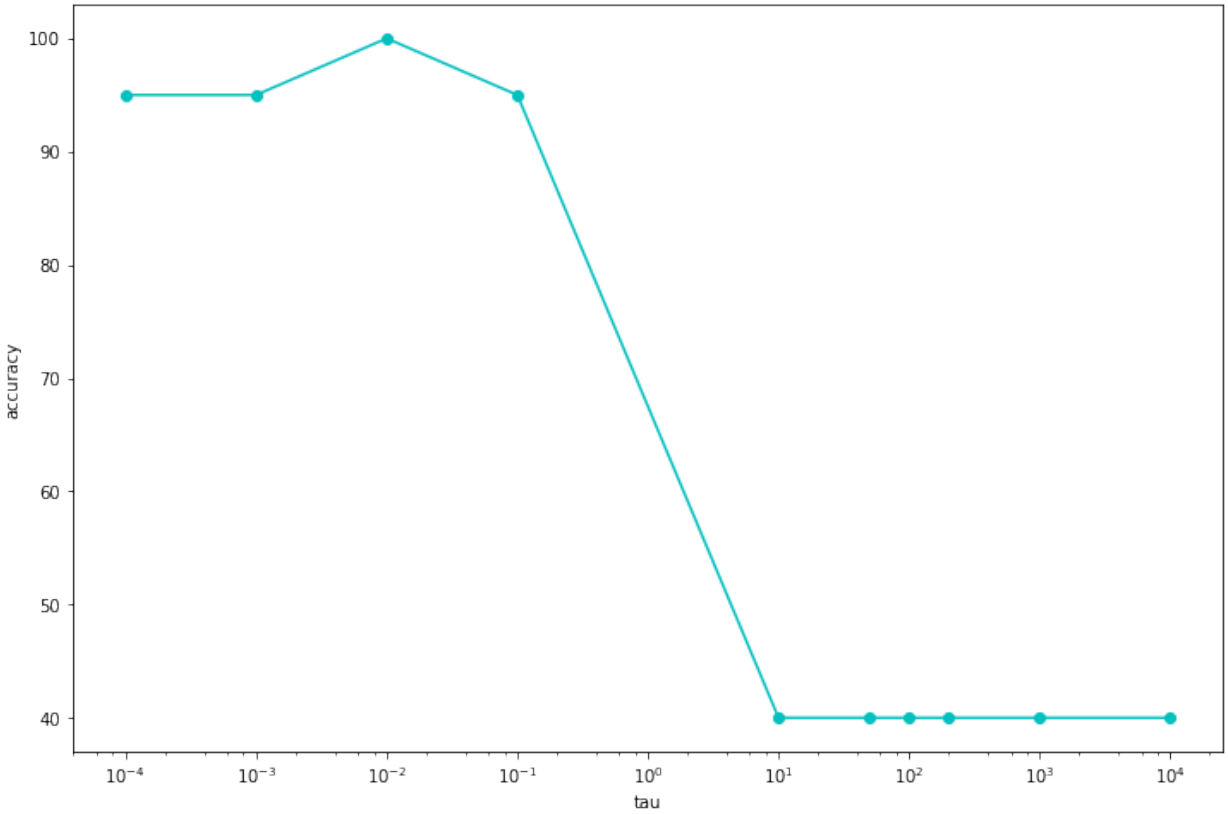


Figure 3.1: Performance on the Iris dataset function of $\tau$

**Optional Part** I randomly generated 2 clusters of data and I randomly picked tau in `[1e-3, 1e-2, 1e-1, 10, 100, 1e3]`. I plotted the data, the separating hyperplans found by our `barr_method` and the results we get using `LinearSVC` from `scikit-learn`. The results are displayed in Figure 3.2

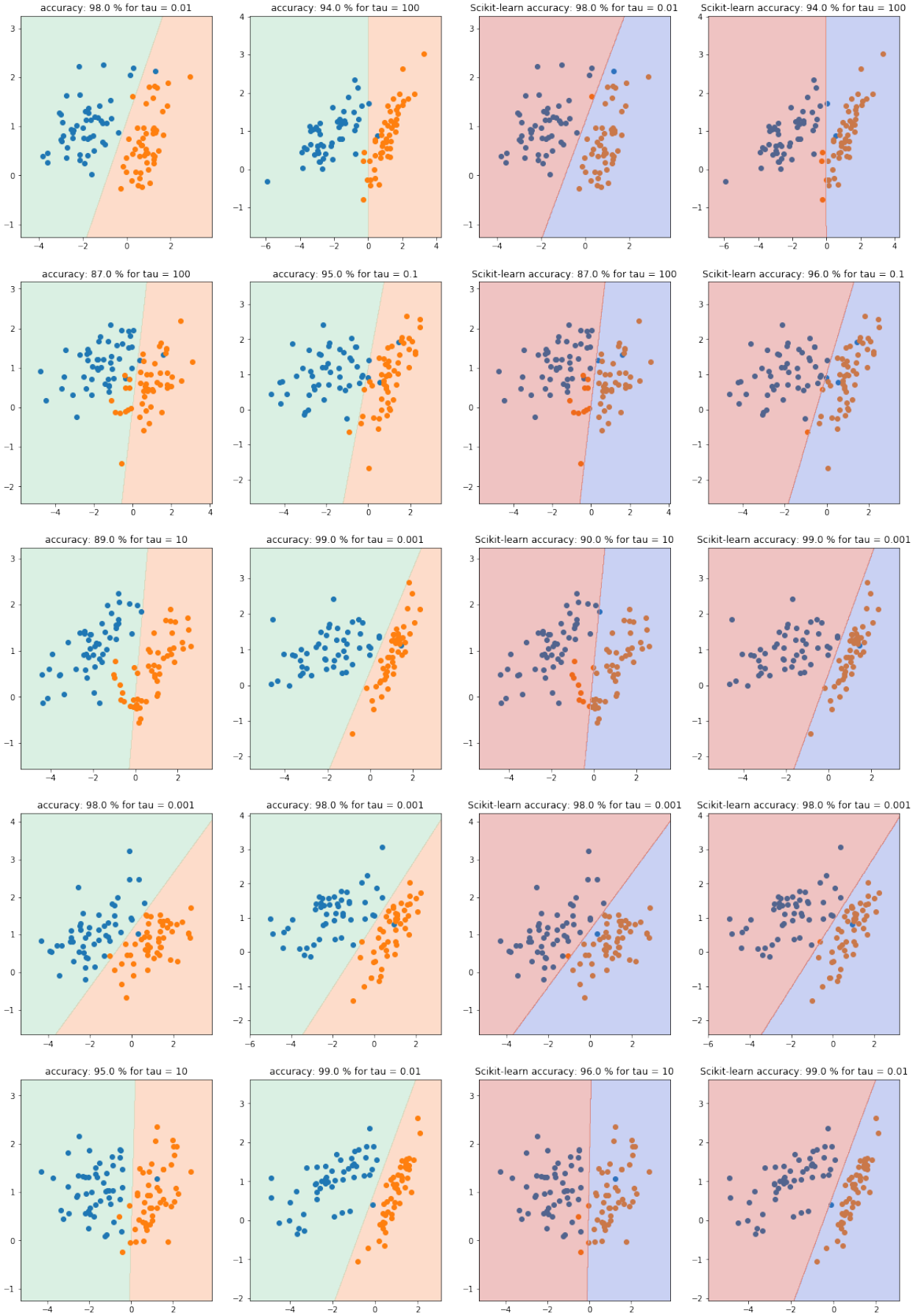


Figure 3.2: **Right**: Accuracies and hyperplans obtained using our implementation. **Left**: Accuracies and hyperplans obtained using `LinearSVC` from scikit-learn with the same parameters

We can see that the results between our implementation and scikit-learn is roughly the same. Yet, I don't understand the little variation between the two as I used the same regularization parameter for both the `bar_method` and the `LinearSVC` function. Moreover I've chosen the tolerance to be $10^{-9}$ for this question.

**3.4 Duality gap versus Newton Iterations** Here, the tolerance is: tol = 1e-6, $\tau = 10$, $X$ is a matrix of features of Iris-versicolor and Iris-virginica only, and $y$ is a vector of classes (Iris-versicolor and Iris-virginica only). Using the barrier method with both the damped Newton Method and the classic Newton Method (with backtracking line search) we can plot the duality gap versus Newton iterations for different values of $\mu$ as shown in Figure 3.3

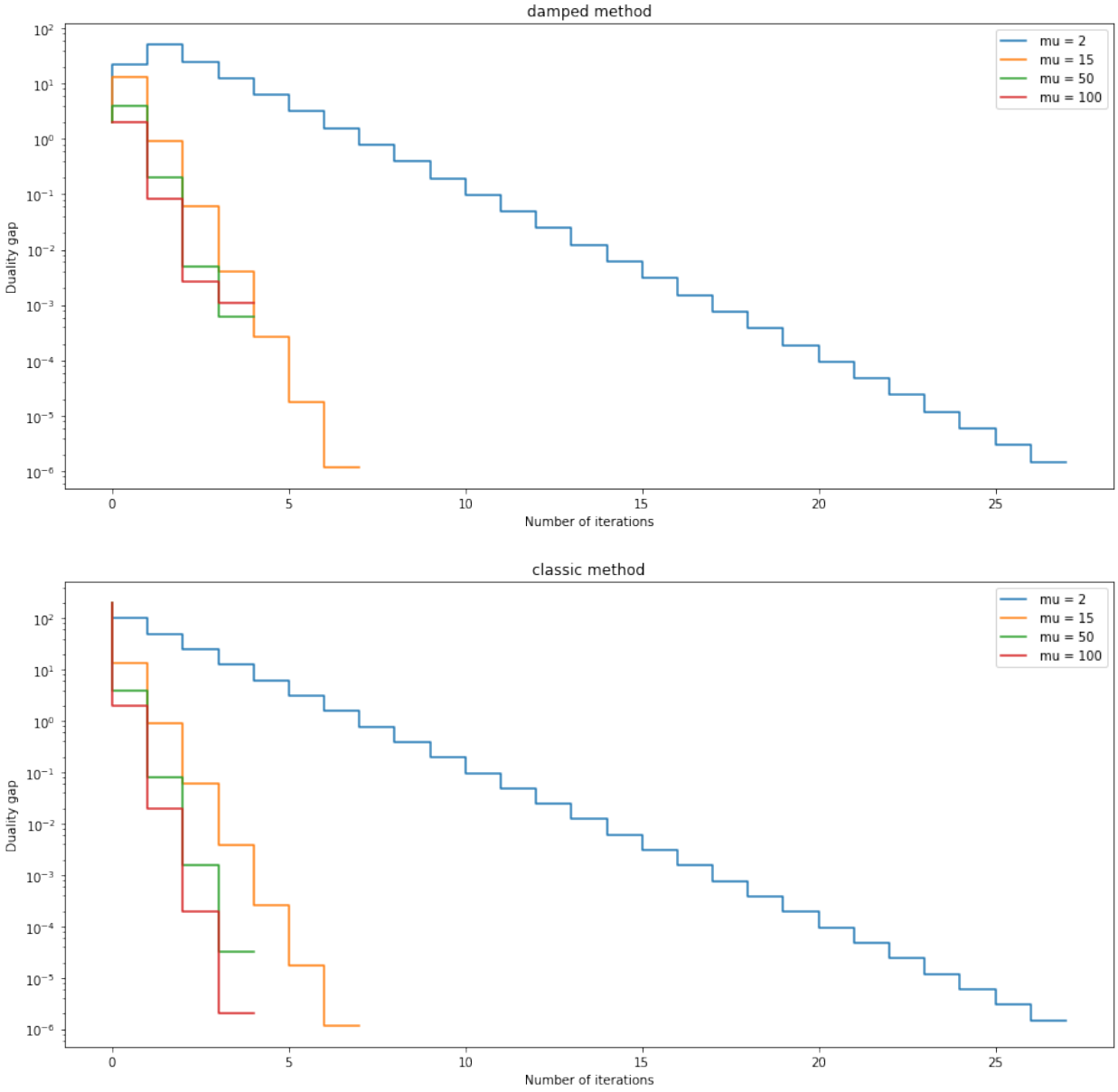


Figure 3.3: **Up**: duality gaps using damped Newton Method, **Down**: duality gaps using Newton Method with line search

We can see that the methods have more or less the same behavior. Yet, depending on the value of

$\tau$ we can notice that the **Damped Newton method** takes usually more iterations to converge. So in a sense is not 'optimal'.