Constrained optimization using the Frank-Wolfe algorithm

Giuseppe Lombardi, Emanuele Ludovico Ungaro

Master Degree in Computer Science.

g.lombardi11@studenti.unipi.it, e.ungaro1@studenti.uipi.it.

Computational Mathematics, Academic Year: 2022/2023

October 27, 2023

Contents

1 Introduction

In this project, we develop a Frank-Wolfe type algorithm for the minimization of a quadratic function with a constrained domain. Specifically, we have to minimize a quadratic function $f : \mathcal{D} \subseteq \mathbb{R}^n \to \mathbb{R}$ of the form

$$f(\mathbf{x}) = \mathbf{x}^T \mathbf{Q} \mathbf{x} + \mathbf{q}^T \mathbf{x}, \tag{1}$$

where $\mathbf{Q} \in \mathbb{R}^{n \times n}$ positive semi-definite and $\mathbf{q} \in \mathbb{R}^n$. The constrained domain $\mathcal{D}$ is defined as follow:

$$\mathcal{D} = \left\{ \mathbf{x} \in \mathbb{R}^n \;\middle|\; x_i \geq 0, \quad \sum_{i \in \mathcal{I}^k} x_i = 1 \; \forall \, k \in \{1, \dots, K\} \right\}, \tag{2}$$

where $\{\mathcal{I}^k\}_{k=1,\dots,K}$ is a partition of the set $\{1, \dots, n\}$:

- $\mathcal{I}^k \subseteq \{1, \dots, n\}$, $\mathcal{I}^k \neq \emptyset \quad \forall \, k \in \{1, \dots, K\}$;
- $\mathcal{I}^i \cap \mathcal{I}^j = \emptyset \quad \forall \, i, j \in \{1, \dots, K\}$ with $i \neq j$;
- $\bigcup_{k=1}^{K} \mathcal{I}^k = \{1, \dots, n\}$.

1.1 Notation

We use the apex ^{T} to denote the transpose of a vector or matrix and the apex $^{(t)}$ to denote the computation at step/time t. We denote a general vector $\mathbf{x} \in \mathbb{R}^n$ as $\mathbf{x} = [x_1, \dots, x_n]^T$, where x_i is the i-th component of $\mathbf{x}$. A general matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ is denoted as $\mathbf{A} = (A_{i,j})$, where $A_{i,j}$ is the element of the i-th row and the j-th column of the matrix $\mathbf{A}$. We denote the subvector of a vector $\mathbf{x} \in \mathbf{R}^n$ as $\mathbf{x}[\mathcal{I}]$, with $\mathcal{I} \subseteq \{1, \dots, n\}$ is the set of indices selected. Similarly, the submatrix of a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ is $\mathbf{A}[\mathcal{I}; \mathcal{J}]$ where $\mathcal{I} \subseteq \{1, \dots, m\}$ and $\mathcal{J} \subseteq \{1, \dots, n\}$ are respectively the indices of rows and the indices of columns selected. Given a set $\mathcal{S} \subseteq \mathbb{R}^n$, $\partial \mathcal{S}$ denotes its boundary. The sum operator of a finite number of sets is the Minkowski addition. The standard basis of $\mathbb{R}^n$ is denoted $\{\mathbf{e}_1, \dots, \mathbf{e}_n\}$. We use the numerical notation $\mathbf{x}^T \mathbf{y}$ to denote the dot product between two vectors $\mathbf{x}$ and $\mathbf{y}$. The norm used is the 2-norm $\| \cdot \|_2$ that we denote $\| \cdot \|$ for simplicity. The convex hull of a set of vertices $\mathbf{v}_1, \dots, \mathbf{v}_k$ is denoted $\operatorname{conv}(\{\mathbf{v}_1, \dots, \mathbf{v}_k\})$. We use D to denote the derivative operator, therefore $D^k f$ denotes the k-order derivative of the function f. The gradient of f in $\mathbf{x}$ is denoted $\nabla f(\mathbf{x}) = \left[\frac{\partial f(\mathbf{x})}{\partial x_1}, \dots, \frac{\partial f(\mathbf{x})}{\partial x_n} \right]^T$. Both the absolute value of a number and the cardinality of a set are given by $| \cdot |$.

2 Preliminary study

First, we describe the function f. The quadratic function f is infinitely differentiable and has derivatives of all orders ($f \in C^\infty(\mathbb{R}^n)$). We can derive a useful property of the function for the convergence analysis (Section 4.1) via the following proposition.

Proposition 1. *The gradient ∇f is Lipschitz with costant $L = 2 \cdot \rho(\mathbf{Q})$, where $\rho(\mathbf{Q})$ is the spectral radius of $\mathbf{Q}$.*

Proof.
$$\max_{\mathbf{x}\in\mathbb{R}^n} \|D[\nabla f(\mathbf{x})]\| = \|D^2 f(\mathbf{x})\| = \|2\mathbf{Q}\| = 2\cdot\rho(\mathbf{Q}),$$
being $\mathbf{Q}$ a symmetric matrix. □

Analysing our domain $\mathcal{D}$ (Definition 2), we observe that each $\mathcal{I}^k$ defines the *polytope* in $\mathbb{R}^n$
$$\mathcal{P}^k = \operatorname{conv}\left(\mathcal{V}^k\right),$$
where $\mathcal{V}^k$ is the set of $n_k = |\mathcal{I}^k|$ vertices $\left\{\mathbf{e}_{i_1},\dots,\mathbf{e}_{i_{n_k}}\right\}$ and $\sum_{k=1}^K n_k = n$. Each $\mathcal{P}^k$ is a compact and convex set, as it is the convex hull of its $\mathcal{V}^k$ vertices.

Proposition 2. *1. The domain $\mathcal{D}$ is the Minkowski sum of the polytopes $\mathcal{P}^1,\dots,\mathcal{P}^k$:*
$$\mathcal{D} = \mathcal{P}^1 + \cdots + \mathcal{P}^K.$$

2. The domain $\mathcal{D}$ is a polytope, and $\mathcal{D} = \operatorname{conv}(\mathcal{V})$ with $\mathcal{V}$ the set of its vertices.

Proof. 1. $\forall \mathbf{x}\in\mathcal{D}$
$$\mathbf{x} = \sum_{i=1}^n x_i\mathbf{e}_i = \sum_{k=1}^K \left(\sum_{i\in\mathcal{I}^k} x_i\mathbf{e}_i\right).$$
For the properties of $\mathcal{D}$ (Definition 2), $\mathbf{x}^k = \sum_{i\in\mathcal{I}^k} x_i\mathbf{e}_i \in \mathcal{P}^k$ and
$$\mathbf{x} = \sum_{k=1}^K \mathbf{x}^k.$$

2. The domain $\mathcal{D}$ is a polytope because it is the Minkowski sum of K polytopes [8] and we can define it as convex hull of its vertices.

□

Corollary 1. *The domain $\mathcal{D}$ is a convex and compact set.*

Proposition 3. *Let $\mathcal{V}$ the set of vertices of $\mathcal{D}$, we have that $\mathcal{V}\subseteq\sum_{k=1}^K\mathcal{V}^k$, where $\mathcal{V}^k$ is the set of vertices of each $\mathcal{P}^k$ and the sum is the Minkowski addition. Therefore, we can write $\mathcal{D} = \operatorname{conv}(\mathcal{V}) = \operatorname{conv}\left(\sum_{k=1}^K\mathcal{V}^k\right)$, with $1\le|\mathcal{V}|\le\left(\frac{n}{K}\right)^K$.*

Proof. $\mathcal{V}\subseteq\sum_{k=1}^K\mathcal{V}^k$ because a vertex of $\mathcal{D}$ must be the sum of K vertices each one in a different $\mathcal{V}^k$ [8]. Being $\mathcal{V}\subseteq\sum_{k=1}^K\mathcal{V}^k$, $\operatorname{conv}(\mathcal{V})\subseteq\operatorname{conv}\left(\sum_{k=1}^K\mathcal{V}^k\right)$. At the same time $\operatorname{conv}\left(\sum_{k=1}^K\mathcal{V}^k\right)\subseteq\operatorname{conv}\left(\sum_{k=1}^K\mathcal{P}^k\right)=\mathcal{D}$, being $\mathcal{D}$ convex. So, we have $\operatorname{conv}(\mathcal{V}) = \operatorname{conv}\left(\sum_{k=1}^K\mathcal{V}^k\right)$. Finally, $|\mathcal{V}|\le|\sum_{k=1}^K\mathcal{V}^k|$ with $|\sum_{k=1}^K\mathcal{V}^k| = \prod_{k=1}^K n_k \le \left(\frac{\sum_{k=1}^K n_k}{K}\right)^K = \left(\frac{n}{K}\right)^K$ for the inequality of arithmetic and geometric means. □

Given that the function f is continuous, it has a minimum in the compact domain $\mathcal{D}$ [20]. Moreover, a compact set in a metric space is equivalent to a sequentially compact space [7], and therefore an iterative optimization method is suitable for our purpose. Moreover, the function f is a convex function for the second-order conditions ($D^2 f = 2\mathbf{Q}\succcurlyeq 0$) [3]; hence we can implement the Frank-Wolfe iterative algorithm to approximate the minimum of the function in $\mathcal{D}$.

Without loss of generality, we consider the case where all polytope $\mathcal{P}^k$ have at least two vertices ($|\mathcal{I}^k|\ge 2$, $\forall k\in\{1,\dots,K\}$). We observe, indeed, that $\forall\mathbf{x}\in\mathcal{D}$

the components related to polytopes with only one vertex are equal to 1; therefore, we can optimize the objective function restricting to the free components related to polytopes with at least two vertices. Denoting $\mathcal{I} = \cup_{|\mathcal{I}^k|\geq 2}\mathcal{I}^k$ and $\mathcal{J} = \cup_{|\mathcal{I}^k|=1}\mathcal{I}^k$ we represent $\mathbf{x}[\mathcal{I}], \mathbf{Q}[\mathcal{I};\mathcal{I}]$ and $\mathbf{q}[\mathcal{I}]$ as $\hat{\mathbf{x}}, \hat{\mathbf{Q}}$ and $\hat{\mathbf{q}}$ respectively, and we can rewrite f as a function $\hat{f} : \mathbb{R}^{|\mathcal{I}|} \longrightarrow \mathbb{R}$, with

$$f(\mathbf{x}) = \hat{f}(\hat{\mathbf{x}}) = \hat{\mathbf{x}}^T\hat{\mathbf{Q}}\hat{\mathbf{x}} + \hat{\mathbf{q}}^T\hat{\mathbf{x}} + 2\cdot\left(\sum_{j\in\mathcal{J}}\mathbf{Q}[\mathcal{I};j]\right)^T\hat{\mathbf{x}} + \sum_{(i,j)\in\mathcal{J}\times\mathcal{J}} Q_{i,j} + \sum_{j\in\mathcal{J}} q_j.$$

3 Frank-Wolfe algorithm

The *Frank–Wolfe algorithm* (FW) is an iterative first-order optimization algorithm for constrained convex optimization [13]. In each iteration, the algorithm considers a linear approximation of the function and moves towards a point that minimizes this linear function.

The function f can be approximated linearly in $\mathcal{D}$ by its first-order approximation

$$\mathcal{L}_{\mathbf{x}}(\mathbf{y}) \coloneqq f(\mathbf{x}) + (\mathbf{y}-\mathbf{x})^T\nabla f(\mathbf{x}), \qquad \forall \mathbf{y}\in\mathcal{D}. \tag{3}$$

Given the point $\mathbf{x}\in\mathcal{D}$, the algorithm minimizes the linear function $\mathcal{L}_{\mathbf{x}}$ in $\mathcal{D}$ by selecting the vector $\bar{\mathbf{y}}\in\mathcal{D}$ that minimizes the dot product with the gradient of the function (Algorithm 2, Line 3). Then, the algorithm moves towards the point $\bar{\mathbf{y}}\in\mathcal{D}$ along the direction $\mathbf{d} = \bar{\mathbf{y}} - \mathbf{x}$. The following proposition assures that the vector $\mathbf{d}$ is a descent direction.

Proposition 4. *Given $\mathbf{x}\in\mathcal{D}$, $\bar{\mathbf{y}} \coloneqq \arg\min_{\mathbf{y}\in\mathcal{D}} \mathbf{y}^T\nabla f(\mathbf{x})$ minimizes $\mathcal{L}_{\mathbf{x}}$ in $\mathcal{D}$ and the vector $\mathbf{d} = \bar{\mathbf{y}} - \mathbf{x}$ is a descent direction.*

Proof.

$$\min_{\mathbf{y}\in\mathcal{D}}\mathcal{L}_{\mathbf{x}}(\mathbf{y}) = f(\mathbf{x}) + \min_{\mathbf{y}\in\mathcal{D}}\mathbf{y}^T\nabla f(\mathbf{x}) - \mathbf{x}^T\nabla f(\mathbf{x}) = \mathcal{L}_{\mathbf{x}}(\bar{\mathbf{y}}),$$

and $\mathbf{d}$ is such that

$$\mathbf{d}^T\nabla f(\mathbf{x}) = (\bar{\mathbf{y}}-\mathbf{x})^T\nabla f(\mathbf{x}) = (\bar{\mathbf{y}})^T\nabla f(\mathbf{x}) - \mathbf{x}^T\nabla f(\mathbf{x}) \leq 0.$$

□

To find the vector $\bar{\mathbf{y}}$, we leverage the structure of the domain. The intuitive idea is to take the indices of every $\mathcal{I}^k$ and use them to select the index of the minimum component of the gradient limited to those indices. After proceeding that way for every $\mathcal{I}^k$, we end up with a set of indices $S = \{j_1, j_2, \ldots, j_K\}$. Finally, to minimize the dot product, we select $\mathbf{y}$ such that $y_i = 1$ if $i\in S$ and $y_i = 0\;\forall\, i\notin S$. Now, we state this more formally and prove it.

Proposition 5. *The dot product between a vector $\mathbf{y}\in\mathcal{D}$ and a vector $\mathbf{v}\in\mathbb{R}^n$ can be minimized by taking, for each $\mathcal{I}^k$, $y_{j_k} = 1$ where $j_k = \arg\min_{i\in\mathcal{I}^k}\{v_i\}$ and 0 to all other $\mathbf{y}$ components.*

Proof. For each vector $\mathbf{y}\in\mathcal{D}$, the dot product between the $\mathbf{y}$ and a vector $\mathbf{v}\in\mathbb{R}^n$ is such that:

$$\begin{aligned}\mathbf{y}^T\mathbf{v} &= y_1v_1 + y_2v_2 + \cdots + y_nv_n \\ &= \sum_{k=1}^{K}\sum_{i\in\mathcal{I}^k} y_iv_i \geq \sum_{k=1}^{K}\min_{i\in\mathcal{I}^k}\{v_i\} = \sum_{k=1}^{K} v_{j_k}.\end{aligned}$$

Therefore $\mathbf{y} \in \mathbb{R}^n$ such that

$$y_i = \begin{cases} 1 & \text{if } i = j_k, \text{ for a } k \in \{1, \dots, K\} \\ 0 & \text{otherwise} \end{cases}$$

minimizes the dot product $\mathbf{y}^T\mathbf{v}$ in $\mathcal{D}$ (it is trivial to prove that such $\mathbf{y} \in \mathcal{D}$). □

Corollary 2. *The vector $\bar{\mathbf{y}} \in \mathcal{D}$ such that*

$$\bar{y}_i = \begin{cases} 1 & \textit{if } i = \arg\min_{i \in \mathcal{I}^k} \left\{ \frac{\partial f(\mathbf{x})}{\partial x_i} \right\}, \textit{ for a } k \in \{1, \dots, K\} \\ 0 & \textit{otherwise} \end{cases}$$

minimizes the dot product $\mathbf{y}^T\nabla f(\mathbf{x})$.

The Algorithm 1 computes the vector $\bar{\mathbf{y}}$, taking as input the function f, a point $\mathbf{x}$ in $\mathcal{D}$, and the `partition` of indices relative to polytopes $\mathcal{P}^k$. The algorithm computes the index relative to the minimum component of the gradient restricted to each polytope $\mathcal{P}^k$ (from Line 4 to Line 7).

Algorithm 1 Linearization minimizer

```
1: procedure LinearizationMinimizer($f$,$\mathbf{x}$,partition)
2:     grad ← $\nabla f(\mathbf{x})$
3:     $\mathbf{y} \leftarrow \mathbf{0} \in \{0\}^n$;
4:     for $\mathcal{I}^k$ in partition do
5:         $j \leftarrow \arg\min$ grad[$\mathcal{I}^k$]
6:         $\mathbf{y}[\mathcal{I}^k[j]] \leftarrow 1$
7:     end for
8: end procedure
```

Algorithm 2 FW algorithm

```
1: procedure FW($f$,$\mathbf{x}$,partition,$\epsilon$)
2:     repeat
3:         $\bar{\mathbf{y}}$ ← LinearizationMinimizer($f$,$\mathbf{x}$,partition)
4:         $\mathbf{d} \leftarrow \bar{\mathbf{y}} - \mathbf{x}$
5:         Select $\alpha \in [0, 1]$
6:         $\mathbf{x} \leftarrow \mathbf{x} + \alpha\mathbf{d}$
7:     until -$\mathbf{d}^T\nabla f(\mathbf{x}) < \epsilon$
8: end procedure
```

The FW algorithm also requires the selection of a step size α (Algorithm 2, Line 5). The step size must satisfy the condition $0 \leq \alpha \leq 1$, to select the new point along the descent direction $\mathbf{d}$ and inside the convex domain $\mathcal{D}$.

3.1 Duality gap

We introduce the notion of duality gap, a commonly used measure for the stop condition of the FW algorithm. The *duality gap*, admitted by the FW algorithm [10], is defined as a function:

$$g(\mathbf{x}) := (\mathbf{x} - \bar{\mathbf{y}})^T\nabla f(\mathbf{x}) = -\mathbf{d}^T\nabla f(\mathbf{x}). \tag{4}$$

At iteration t the duality gap is

$$g(\mathbf{x}^{(t)}) = (\mathbf{x}^{(t)} - \bar{\mathbf{y}}^{(t)})^T \nabla f(\mathbf{x}^{(t)}) = -(\mathbf{d}^{(t)})^T \nabla f(\mathbf{x}^{(t)}), \tag{5}$$

where $\bar{\mathbf{y}}^{(t)} = \arg\min_{\mathbf{y}\in\mathcal{D}} \mathbf{y}^T \nabla f(\mathbf{x}^{(t)})$.

Proposition 6. *The duality gap is an upper bound of $f(\mathbf{x}^{(t)}) - f^*$, where $f^* := f(\mathbf{x}^*) = \min_{\mathbf{x}\in\mathcal{D}} f(\mathbf{x})$.*

Proof. Being f a convex function, for the first-order condition [4]

$$\forall \mathbf{x}, \mathbf{y} \in \mathcal{D} \qquad f(\mathbf{y}) \geq \mathcal{L}_{\mathbf{x}}(\mathbf{y}).$$

Therefore, at each iteration t

$$\forall \mathbf{y} \in \mathcal{D} \qquad f(\mathbf{y}) \geq f(\mathbf{x}^{(t)}) + (\mathbf{y} - \mathbf{x}^{(t)})^T \nabla f(\mathbf{x}^{(t)})$$

and minimizing both sides over all $\mathbf{y} \in \mathcal{D}$ we obtain

$$f^* \geq f(\mathbf{x}^{(t)}) + (\bar{\mathbf{y}}^{(t)} - \mathbf{x}^{(t)})^T \nabla f(\mathbf{x}^{(t)}).$$

Hence, by rearranging the inequality, we have

$$f(\mathbf{x}^{(t)}) - f^* \leq g(\mathbf{x}^{(t)}).$$

□

Corollary 3. *If $\mathbf{x} \in \mathcal{D}$ is such that the duality gap $g(\mathbf{x}) = 0$, $\mathbf{x}$ is a global minimum in $\mathcal{D}$.*

Exploiting the Proposition 6 and the Corollary 3, fixed a threshold $\epsilon > 0$, the algorithm iterates as long as the duality gap is greater than the threshold (Algorithm 2, Line 7).

In the end, the general step t of the FW algorithm is defined as

$$\begin{aligned} \mathbf{x}^{(t+1)} &= \mathbf{x}^{(t)} + \alpha^{(t)} \cdot (\bar{\mathbf{y}}^{(t)} - \mathbf{x}^{(t)})^T \nabla f(\mathbf{x}^{(t)}) \\ &= \mathbf{x}^{(t)} + \alpha^{(t)} \cdot \mathbf{d}^{(t)}, \qquad t = 0, 1, 2, \dots \end{aligned} \tag{6}$$

where $\alpha^{(t)} \in [0, 1]$ and $\bar{\mathbf{y}}^{(t)} = \arg\min_{\mathbf{y}\in\mathcal{D}} \mathbf{y}^T \nabla f(\mathbf{x}^{(t)})$.

3.2 Step size selection

For the step size selection, we implement the exact line-search. We select the optimal point starting from $\mathbf{x}^{(t)}$ and moving along the descent direction $\mathbf{d}^{(t)}$ while staying within the domain $\mathcal{D}$. More precisely, the FW algorithm with the *exact line-search* selects at each iteration t:

$$\alpha^{(t)} = \underset{\alpha\in[0,\alpha_{max}]}{\arg\min}\ f(\mathbf{x}^{(t)} + \alpha \cdot \mathbf{d}^{(t)}),$$

where $\mathbf{d}^{(t)}$ is the descent direction, and α_{max} is the largest step size to stay in $\mathcal{D}$.

To perform the exact line-search we differentiate the function

$$\phi^{(t)}(\alpha) = f(\mathbf{x}^{(t)} + \alpha \mathbf{d}^{(t)})$$

with respect to α, and we set the derivative to zero:

$$\begin{gathered}
\frac{d\phi^{(t)}(\alpha)}{d\alpha} = 0 \\
2(\mathbf{x}^{(t)})^T\mathbf{Q}\mathbf{d}^{(t)} + 2\alpha(\mathbf{d}^{(t)})^T\mathbf{Q}\mathbf{d}^{(t)} + \mathbf{q}^T\mathbf{d}^{(t)} = 0 \\
2\alpha(\mathbf{d}^{(t)})^T\mathbf{Q}\mathbf{d}^{(t)} = -2(\mathbf{x}^{(t)})^T\mathbf{Q}\mathbf{d}^{(t)} - \mathbf{q}^T\mathbf{d}^{(t)} \\
\alpha = -\frac{(\mathbf{d}^{(t)})^T(2\mathbf{Q}\mathbf{x}^{(t)} + \mathbf{q})}{2(\mathbf{d}^{(t)})^T\mathbf{Q}\mathbf{d}^{(t)}} \\
\alpha^{(t)} = -\frac{(\mathbf{d}^{(t)})^T \cdot \nabla f(\mathbf{x}^{(t)})}{2(\mathbf{d}^{(t)})^T\mathbf{Q}\mathbf{d}^{(t)}}.
\end{gathered}$$

Therefore, exploiting the duality gap (Equation 5), the exact minimizer $\alpha^{(t)}$ at iteration t is

$$\alpha^{(t)} = \frac{g(\mathbf{x}^{(t)})}{2(\mathbf{d}^{(t)})^T\mathbf{Q}\mathbf{d}^{(t)}}. \tag{7}$$

Since the matrix $\mathbf{Q}$ is semi-definite positive, computing the denominator of the Equation 7 requires special attention and gives us helpful information. If the denominator equals 0, the function along the $\mathbf{d}$ direction is linear, and since $\mathbf{d}$ is a descent direction (Proposition 4), the function is unbounded below. In this case, we will set α to α_{max} to take the step size upper bound without leaving the domain (Algorithm 3, Line 3). Similarly, we avoid alphas greater than α_{max} (Algorithm 3, Line 6).

Algorithm 3 Exact Line-Search

```
1: procedure ExactLineSearch($\mathbf{Q}, g(\mathbf{x}), \mathbf{d}, \alpha_{max}$)
2:     den ← $2\mathbf{d}^T\mathbf{Q}\mathbf{d}$
3:     if den == 0 then
4:         $\alpha \leftarrow \alpha_{max}$
5:     else
6:         $\alpha \leftarrow \min\{\frac{g(\mathbf{x})}{\text{den}}, \alpha_{max}\}$
7:     end if
8: end procedure
```

3.3 Away step

In the FW algorithm, descent directions are always directed toward extreme points of $\mathcal{D}$. When one gets close to the optimum, and when this optimum is a point on the boundary of $\mathcal{D}$, these directions become more and more orthogonal to the gradient vector without reaching the optimal face, resulting in a sublinear (Proposition 11) convergence rate. In other words, the method exhibits a zig-zagging behaviour as the sequence of iterates approaches a solution on the boundary of the feasible set. To remedy this situation, [15] suggests enlarging the set of admissible directions by including directions pointing 'away' from bad vertices.

Proposition 7. *Given* $\mathbf{x} \in \mathcal{D}$, $\overset{+}{\mathbf{y}} := \arg\max_{\mathbf{y}\in\mathcal{D}} \mathbf{y}^T\nabla f(\mathbf{x})$ *maximizes* $\mathcal{L}_\mathbf{x}(\mathbf{y})$ *in* $\mathcal{D}$ *and the vector* $\mathbf{d} = \mathbf{x} - \overset{+}{\mathbf{y}}$ *is a descent direction.*

Proof.

$$\max_{\mathbf{y}\in\mathcal{D}} \mathcal{L}_\mathbf{x}(\mathbf{y}) = f(\mathbf{x}) + \max_{\mathbf{y}\in\mathcal{D}} \mathbf{y}^T\nabla f(\mathbf{x}) - \mathbf{x}^T\nabla f(\mathbf{x}) = \mathcal{L}_\mathbf{x}(\overset{+}{\mathbf{y}}),$$

and $\mathbf{d}$ is such that

$$\mathbf{d}^T \nabla f(\mathbf{x}) = (\mathbf{x} - \overset{+}{\mathbf{y}})^T \nabla f(\mathbf{x}) = (\mathbf{x})^T \nabla f(\mathbf{x}) - \overset{+}{\mathbf{y}}^T \nabla f(\mathbf{x}) \leq 0.$$

□

In the *away step*, the descent direction is the opposite of the direction towards the point $\overset{+}{\mathbf{y}}$. In other words, we go away from the worst vertex $\overset{+}{\mathbf{y}}$ that maximizes the linear approximation $\mathcal{L}_{\mathbf{x}}(\mathbf{y})$ in $\mathcal{D}$. To compute $\overset{+}{\mathbf{y}}$, we leverage the characteristics of the domain $\mathcal{D}$. Given $\mathbf{x} \in \mathcal{D}$, we consider for each set of indices $\mathcal{I}^k$ the subset of indices

$$\mathcal{A}^k(\mathbf{x}) \coloneqq \{i \in \mathcal{I}^k \mid x_i > 0\},$$

and the following Proposition.

Proposition 8. *The dot product between a vector $\mathbf{y} \in \mathcal{D}$ and a vector $\mathbf{v} \in \mathbb{R}^n$ can be maximized by taking, for each $\mathcal{A}^k(\mathbf{v})$, $y_{j_k} = 1$ where $j_k = \arg\max_{i \in \mathcal{A}^k(\mathbf{v})} \{v_i\}$ and 0 to all other components of $\mathbf{y}$.*

Proof. For each vector $\mathbf{y} \in \mathcal{D}$, the dot product between the $\mathbf{y}$ and a vector $\mathbf{v} \in \mathbb{R}^n$ is such that:

$$\begin{aligned}\mathbf{y}^T\mathbf{v} &= y_1 v_1 + y_2 v_2 + \cdots + y_n v_n \\ &= \sum_{k=1}^{K} \sum_{i \in \mathcal{A}^k(\mathbf{v})} y_i v_i \leq \sum_{k=1}^{K} \max_{i \in \mathcal{A}^k(\mathbf{v})} \{v_i\} = \sum_{k=1}^{K} v_{j_k}.\end{aligned}$$

Therefore $\mathbf{y} \in \mathbb{R}^n$ such that

$$y_i = \begin{cases} 1 & \text{if } i = j_k, \text{ for a } k \in \{1, \dots, K\} \\ 0 & \text{otherwise} \end{cases}$$

maximizes the dot product $\mathbf{y}^T\mathbf{v}$ in $\mathcal{D}$ (it is trivial to prove that such $\mathbf{y} \in \mathcal{D}$). □

We must properly compute the *step size upper bound* α_{max} that allows us to move along the direction $\mathbf{d} = \mathbf{x} - \overset{+}{\mathbf{y}}$ without leaving the domain $\mathcal{D}$.

Proposition 9. *Given $\mathbf{x} \in \mathcal{D}$, the largest step size selectable among the descent direction $\mathbf{x} - \overset{+}{\mathbf{y}}$ is $\alpha_{max} = \min_k \left\{\frac{x_{j_k}}{1 - x_{j_k}}\right\}$, where $j_k = \arg\max_{i \in \mathcal{A}^k(\mathbf{x})} \{x_i\}$ $\forall k \in \{1, \dots, K\}$.*

Proof. (Hint) Exploiting the structure of the domain $\mathcal{D}$ given by the Proposition 2, for each $\mathcal{A}^k(\mathbf{x})$ the largest step size selectable to stay in $\mathcal{P}^k$ is $\frac{x_{j_k}}{1 - x_{j_k}}$ [2], therefore, to stay in $\mathcal{D}$ we take the minimum among these values. □

Following the same approach of the Algorithm 1, the Algorithm 4 computes the vector $\overset{+}{\mathbf{y}}$ for the away-step. The algorithm computes the index relative to the maximum component of the gradient restricted to each polytope $\mathcal{P}^k$, restricting to the active indices $\mathcal{A}^k$ (Line 8). At the same time, the algorithm computes the step size upper bound α_{max}, updating it whenever the maximum step size related to a specific polytope $\mathbf{P}^k$ is smaller than the previous ones observed (Line 10).

Algorithm 4 Linearization maximizer

```
1: procedure LinearizationMaximizer($f$,$\mathbf{x}$,partition)
2:     grad $\leftarrow \nabla f(\mathbf{x})$
3:     $\mathbf{y} \leftarrow \mathbf{0} \in \{0\}^n$;
4:     actives $\leftarrow \mathbf{x} > 0$
5:     $\alpha_{max} \leftarrow inf$
6:     for $\mathcal{I}^k$ in partition do
7:         $\mathcal{A}^k \leftarrow \mathcal{I}^k$[actives]
8:         $j \leftarrow \arg\max \texttt{grad}[\mathcal{A}^k]$
9:         $\mathbf{y}[\mathcal{A}^k[j]] \leftarrow 1$
10:        $\alpha_{max} \leftarrow \min\left\{\alpha_{max}, \frac{\mathbf{x}[\mathcal{A}^k[j]]}{1-\mathbf{x}[\mathcal{A}^k[j]]}\right\}$
11:    end for
12: end procedure
```

Given a point $\mathbf{x} \in \mathcal{D}$, we define the *duality gap of a descent direction* $\mathbf{d}$ as

$$g_{\mathbf{x}}(\mathbf{d}) = -\mathbf{d}^T \nabla f(\mathbf{x}). \tag{8}$$

The value of $g_{\mathbf{x}}(\mathbf{d})$ expresses the 'goodness' of the descent direction $\mathbf{d}$ starting from the point $\mathbf{x}$. The *Away-step Frank-Wolfe algorithm* (AFW, Algorithm 3) makes a decision at each step between taking the descent direction that minimizes the function's linear approximation (Lines 6 – 8) or using the direction obtained from the away step (Lines 10 – 12) based on the best duality gap between the two directions (Line 5). To determine the step size for both directions, we use the exact line-search algorithm (Line 14), selecting $\alpha = \arg\min_{\alpha \in [0,\alpha_{max}]} f(\mathbf{x} + \alpha\mathbf{d})$.

Algorithm 5 AFW algorithm

```
1: procedure AFW($f$,$\mathbf{x}$,partition,$\epsilon$)
2:     repeat
3:         $\bar{\mathbf{y}} \leftarrow$ LinearizationMinimizer($f$,$\mathbf{x}$,partition)
4:         $\overset{+}{\mathbf{y}} \leftarrow$ LinearizationMaximizer($f$,$\mathbf{x}$,partition)
5:         if $g_{\mathbf{x}}(\bar{\mathbf{y}} - \mathbf{x}) \geq g_{\mathbf{x}}(\mathbf{x} - \overset{+}{\mathbf{y}})$ then
6:             gap $\leftarrow g_{\mathbf{x}}(\bar{\mathbf{y}} - \mathbf{x})$;
7:             $\mathbf{d} \leftarrow \bar{\mathbf{y}} - \mathbf{x}$
8:             $\alpha_{max} \leftarrow 1$;
9:         else
10:            gap $\leftarrow g_{\mathbf{x}}(\mathbf{x} - \overset{+}{\mathbf{y}})$;
11:            $\mathbf{d} \leftarrow \mathbf{x} - \overset{+}{\mathbf{y}}$
12:            $\alpha_{max} \leftarrow \min_k \left\{\frac{x_{j_k}}{1-x_{j_k}}\right\}$;
13:        end if
14:        $\alpha \leftarrow$ ExactLineSearch($\mathbf{Q}$, gap, $\mathbf{d}$, $\alpha_{max}$);        ▷ $f(\mathbf{x}) = \mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{q}^T\mathbf{x}$
15:        $\mathbf{x} \leftarrow \mathbf{x} + \alpha\mathbf{d}$
16:    until gap $< \epsilon$
17: end procedure
```

4 Convergence analysis

4.1 FW convergence analysis

The original research paper by Frank and Wolfe [13] proves that the function values converge to the optimal value at a sublinear rate. Additionally, [12] demonstrates that this convergence rate applies even when the feasible set is a general compact convex set. To provide more detail and limit ourselves to the problem domain $\mathcal{D}$, we present these findings and define the curvature constant [10] of $f(\mathbf{x})$ over $\mathcal{D}$.

$$C = \max_{\substack{\alpha\in[0,1] \\ \mathbf{x},\mathbf{x}'\in\mathcal{D}}} \frac{2}{\alpha^2} \left\{ f(\mathbf{y}) - \mathcal{L}_{\mathbf{x}}(\mathbf{y}) \mid \mathbf{y} = \mathbf{x} + \alpha(\mathbf{x}' - \mathbf{x}) \right\}. \tag{9}$$

The curvature constant C of a differentiable function $f(\mathbf{x})$ satisfies, $\forall \mathbf{x}, \mathbf{x}' \in \mathcal{D}$ and $\forall\alpha \in [0, 1]$, the inequality

$$f(\mathbf{x} + \alpha(\mathbf{x}' - \mathbf{x})) \leq f(\mathbf{x}) + \alpha(\mathbf{x}' - \mathbf{x})^T \nabla f(\mathbf{x}) + \frac{C}{2}\alpha^2. \tag{10}$$

We note that for a linear function, $C = 0$. In our case, we can find an upper bound on the curvature constant.

Proposition 10. *For the curvature constant C, it holds that*

$$C \leq L \cdot (\mathrm{diam}(\mathcal{D}))^2.$$

Proof. For Lipschitz $\nabla f(\mathbf{x})$ of constant L, we have that

$$\forall \mathbf{x}, \mathbf{y} \in \mathcal{D} \qquad f(\mathbf{y}) - \mathcal{L}_{\mathbf{x}}(\mathbf{y}) \leq \frac{L}{2}\|\mathbf{y} - \mathbf{x}\|^2,$$

therefore, multiplying by $\frac{2}{\alpha^2}$ and maximizing over all $\alpha \in [0, 1]$ and $\mathbf{x}, \mathbf{x}' \in \mathcal{D}$ with $\mathbf{y} = \mathbf{x} + \alpha(\mathbf{x}' - \mathbf{x})$

$$\begin{aligned} C &\leq \max_{\substack{\alpha\in[0,1] \\ \mathbf{x},\mathbf{x}'\in\mathcal{D}}} \frac{L}{2} \cdot \frac{2}{\alpha^2} \left\{ \|\mathbf{y} - \mathbf{x}\|^2 \,\middle|\, \mathbf{y} = \mathbf{x} + \alpha(\mathbf{x}' - \mathbf{x}) \right\} \\ &= \max_{\mathbf{x},\mathbf{x}'\in\mathcal{D}} L \cdot \|\mathbf{x}' - \mathbf{x}\|^2 = L \cdot (\mathrm{diam}(\mathcal{D}))^2. \end{aligned}$$

□

The curvature constant C limits the deviation of the function from the linear approximation by $\nabla f(\mathbf{x})$, related to the Proposition 10.

Now, we can deepen the convergence analysis through the following theorem.

Theorem 1. *(Theorem 1, [10]) The FW algorithm with line-search, setting $\alpha_t = \arg\min_{\alpha\in[0,1]} f(\mathbf{x}^{(t)} + \alpha\mathbf{d}^{(t)})$ at each iteration t (Equation 6), satisfies*

$$f(\mathbf{x}^{(t)}) - f^* \leq \frac{2C}{t+2}, \qquad t = 0, 1, 2, \ldots.$$

Proof. (Appendix A, [10]). □

Proposition 11. *The FW algorithm with line-search converges sublinearly, with an order of convergence $O\left(\frac{1}{t}\right)$.*

Proof. For each $t = 0, 1, 2, \ldots,$

$$f(\mathbf{x}^{(t)}) - f^* \leq \frac{2C}{t+2}.$$

By the computation of the limit, we obtain the sublinearly convergence

$$\lim_{t\to\infty} \frac{\left(f(\mathbf{x}^{(t+1)}) - f^*\right)}{\left(f(\mathbf{x}^{(t)}) - f^*\right)} = \lim_{t\to\infty} \frac{2C}{t+3} \cdot \frac{t+2}{2C} = 1.$$

Observing the bound, the number of iterations t needed for $f(\mathbf{x}^{(t)}) - f^* \leq \epsilon$ is $O\left(\frac{1}{\epsilon}\right)$. □

The FW algorithm shows a linear convergence order under the assumption that objective function f is strongly convex (Definition 4.1) and the assumption on the location of the optimal solution. A function $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ is *μ-strongly convex* over a convex set $\mathcal{D} \subseteq \mathbb{R}^n$ if it satisfies the following two equivalent conditions

1. $\forall \mathbf{x}, \mathbf{y} \in \mathcal{D}$:
$$f(\mathbf{y}) \geq f(\mathbf{x}) + \nabla f(\mathbf{x}) \cdot (\mathbf{y} - \mathbf{x}) + \frac{\mu}{2}\|\mathbf{y} - \mathbf{x}\|^2;$$

2. $\forall \mathbf{x}, \mathbf{y} \in \mathcal{D}, \alpha \in \mathcal{D}$:
$$f(\alpha\mathbf{x} + (1-\alpha)\mathbf{y}) \leq \alpha f(\mathbf{x}) + (1-\alpha) f(\mathbf{y}) - \frac{\mu}{2}(1-\alpha)\|\mathbf{y} - \mathbf{x}\|^2.$$

Theorem 2. *(Theorem 2, [9]) Assuming that the optimal solution belongs to the relative interior of the domain* $\operatorname{relint}(\mathcal{D})$ *(see Definition in [5]) and f is Lipschitz and strongly-convex, then the sequence $\{\mathbf{x}^{(t)}\}_{t=0,1,\ldots}$ generated by the FW algorithm converges linearly.*

Corollary 4. *If $\mathbf{Q}$ is positive definite and $\mathbf{x}^* \in \operatorname{relint}(\mathcal{D})$, the FW algorithm converges linearly.*

Proof. Our function f (Task 1) is Lipschitz and strongly convex with constant $\mu = 2\lambda_{min}$, being $2\lambda_{min}$ the smallest eigenvalue of the Hessian matrix $D^2(f) = 2\mathbf{Q}$, where $\lambda_{min} > 0$ is the smallest eigenvalue of $\mathbf{Q}$. □

4.2 AFW convergence analysis

Several papers show linear convergence results for the away step variant of the Frank-Wolfe algorithm when the domain is a polytope $\operatorname{conv}(\mathcal{V})$, with $\mathcal{V}$ a set of vertices in $\mathbb{R}^n$ and the objective function is strongly convex. Since the matrix $\mathbf{Q}$ is positive semidefinite, our function f, in general, (Equation 1) is not strongly convex; the function f, indeed, is linear along the eigenvectors corresponding to 0 eigenvalue of $\mathbf{Q}$. [17], [11], [1], and [16] establish a linear convergence result for the case when the objective is a composite function of the form

$$f(\mathbf{x}) = h(\mathbf{E}\mathbf{x}) + \mathbf{q}^T\mathbf{x}, \tag{11}$$

where $E \in \mathbb{R}^{p\times n}$, $b \in \mathbb{R}^n$ and $h : \mathbb{R}^p \longrightarrow \mathbb{R}$ is a strongly convex function with Lipschitz gradient. Moreover, the linear convergence result does not depend on the particular position of the optimal solution.

Proposition 12. *[16], [1], [17], [11] Suppose that the objective function has the Form 11. Suppose the domain is a polytope $\mathcal{D} = \operatorname{conv}(\mathcal{V})$, with $\mathcal{V}$ a set of vertices. If $\mathbf{x}^{(0)}$ is a vertex of $\mathcal{D}$ and the step size $\alpha^{(t)}$ is computed by the exact line-search (Algorithm 3), then it exists a constant $r \in \left(0, \frac{1}{2}\right]$ such that the sequence of points $\{\mathbf{x}^{(t)}\}_{t=0,1,\dots}$ generated by the AFW algorithm (5) satisfies:*

$$f(\mathbf{x}^{(t)}) - f^* \leq \left(f(\mathbf{x}^{(0)}) - f^*\right) \left(\sqrt{1-r}\right)^t. \tag{12}$$

Proof. (Hint) The AFW algorithm performs in a generic step t two possible types of steps: a *good step* if $\alpha^{(t)} < \alpha_{max}$ or $\alpha^{(t)} = \alpha_{max} \geq 1$; or a *drop step* if $\alpha^{(t)} = \alpha_{max} < 1$. In a good step, the error decreases geometrically:

$$f(\mathbf{x}^{(t+1)}) - f^* \leq \left(f(\mathbf{x}^{(t)}) - f^*\right)(1-r). \tag{13}$$

Moreover, the number of drop steps up to N iterations is bounded by $\frac{N}{2}$. This results in a linear convergence with rate $\sqrt{1-r}$ (Statement 12). □

Corollary 5. *The AFW algorithm applied to our Task 1 converges linearly, with an order $O\left(R^t\right)$ and a constant rate $R = \sqrt{1-r} \in \left[\frac{1}{\sqrt{2}}, 1\right)$.*

Proof. We can rewrite our function in the Form 11

$$f(\mathbf{x}) = h(\mathbf{Q}^{\frac{1}{2}}\mathbf{x}) + \mathbf{q}^T\mathbf{x}, \tag{14}$$

setting $h : \mathbb{R}^n \longrightarrow \mathbb{R}$ with $h(\mathbf{v}) = \|\mathbf{v}\|^2$. The function h satisfies the hypothesis, being twice continuously differentiable and $D^2h = \mu I$ positive definite with $\mu = 2$. Therefore we apply Proposition 12 and we rewrite the Statement 12 as

$$f(\mathbf{x}^{(t)}) - f^* \leq C \cdot R^t,$$

with $R = \sqrt{1-r} \in \left[\frac{1}{\sqrt{2}}, 1\right)$, being the costant $r \in \left(0, \frac{1}{2}\right]$. □

4.2.1 Convergence rate

The papers [17], [11], [1], and [16] all share a common feature - they express the linear convergence rate in terms of the condition number of the objective function f and the polytope $\operatorname{conv}(\mathcal{V})$. The condition number of the polytope is a condition on the alignment of the search direction, selected by the AFW algorithm at each iteration, with the gradient of the objective function at the current iterate. The alignment should be comparable to that of a direct step toward the optimal solution.

We exploit the definition of *pyramidal width* in [11], which is more suitable for the convergence analysis concerning our domain.

Proposition 13. *(Section 4 in [11]) Related to our Task 1 the constant $r \in \left[0, \frac{1}{2}\right)$ is*

$$r = \min\left\{\frac{\tilde{\mu}}{8} \cdot \left(\frac{\delta}{M}\right)^2, \frac{1}{2}\right\},$$

where $\tilde{\mu}$ is the generalized strong-convexity constant for f, $\delta = \operatorname{PWidth}(\mathcal{V})$ is the pyramidal width of $\mathcal{V}$, and $M = \operatorname{diam}(\operatorname{conv}(\mathcal{V}))$ is the diameter.

The condition number of the function f is given by the *generalized strong-convexity constant* $\tilde{\mu}$ (Lemma 9 of Appendix F in [11]) such that for any $\mathbf{x} \in \mathcal{D}$ and $\mathbf{x}^*$ in the set of solutions $\mathcal{X}^* = \{\mathbf{x} \in \mathcal{D} \mid f(\mathbf{x}) = f^*\}$

$$f(\mathbf{x}^*) \geq f(\mathbf{x}) + 2\nabla f(\mathbf{x}) \cdot (\mathbf{x}^* - \mathbf{x}) + 2\tilde{\mu}\|\mathbf{x}^* - \mathbf{x}\|^2.$$

The constant $\tilde{\mu}$ is obtained by exploiting the strong convexity constant μ of the function $h(\cdot) = \|\cdot\|^2$ in the Formula 14 and the convexity property of f. Particularly for our Task 1,

$$\tilde{\mu} = \frac{1}{2\theta^2(\|\mathbf{q}\| \operatorname{diam}(\mathcal{D}) + 6\rho(\mathbf{Q}) \operatorname{diam}(\operatorname{conv}(\{\mathbf{Q}^{1/2}\mathbf{v} \mid \mathbf{v} \in \mathcal{V}\})) + 4\rho(\mathbf{Q})^2 + 1)},$$

where θ is the *Hoffman constant* (Lemma 2.2 in [1]) associated with the matrix $\begin{bmatrix} \mathbf{Q}^{1/2} \\ \mathbf{q}^T \\ \mathbf{A} \end{bmatrix}$, where the rows of $\mathbf{A}$ are the linear inequality constraints defining the set $\mathcal{D}$.
A higher value of $\tilde{\mu}$ indicates a faster convergence. Observing that $\operatorname{diam}(\operatorname{conv}(\{\mathbf{Q}^{1/2}\mathbf{v} \mid \mathbf{v} \in \mathcal{V}\})) \approx \sqrt{\rho(\mathbf{Q})} \operatorname{diam}(\mathcal{D})$, the main terms of $\tilde{\mu}$ formula are θ and $\rho(\mathbf{Q})$. The convergence is slower if the values of θ and $\rho(\mathbf{Q})$ are high. The Hoffman constant θ is equal to

$$\theta = \max_{\mathbf{B} \in \mathcal{B}} \left\{ \frac{1}{\lambda_{min}(\mathbf{B}\mathbf{B}^T)} \right\},$$

where $\mathcal{B}$ is the set of all matrices constructed by taking linearly independent rows by $\begin{bmatrix} \mathbf{Q}^{1/2} \\ \mathbf{q}^T \\ \mathbf{A} \end{bmatrix}$, and $\lambda_{min}(\cdot)$ is the smallest eigenvalue. The exact computation of the Hoffman constant is demanding from the point of view of time complexity. However, we can briefly analyse the value that this constant can assume. In our case, the Hoffman constant is associated with the matrix $\begin{bmatrix} \mathbf{Q}^{1/2} \\ \mathbf{q}^T \\ \mathbf{P} \\ \mathbf{I} \end{bmatrix}$, being the domain $\mathcal{D}$ defined as

$$\mathcal{D} = \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{P}\mathbf{x} \leq 1, -\mathbf{P}\mathbf{x} \leq -1, -\mathbf{I}\mathbf{x} \leq 0\},$$

where $\mathbf{P} \in \mathbb{R}^{K \times n}$ is the matrix defined in 15, and $\mathbf{I} \in \mathbb{R}^{n \times n}$ is the identity matrix. If $\mathbf{Q}$ is diagonal (i.e. setting $\delta = \frac{1}{n}$ in the Algorithm 6), the eigenvalues of the matrices $\mathbf{B}\mathbf{B}^T$ formed by the linearly independent rows of $\mathbf{Q}$ are equal to the squared eigenvalues of $\mathbf{Q}$. So, in this case, if the smallest strictly positive eigenvalue $\lambda^>_{min}(\mathbf{Q})$ is sufficiently low, the Hoffman constant is equal to $\frac{1}{\lambda^>_{min}(\mathbf{Q})^2}$. Regardless of the matrix $\mathbf{Q}$, on the other hand, if $\mathbf{q} \neq \mathbf{0}$ and the value of $\|\mathbf{q}\|$ is sufficiently low, θ becomes equal to $\frac{1}{\|\mathbf{q}\|^2}$. Moreover, if $\dim(\ker(\mathbf{Q}))$ is high, i.e. $\approx n$, the cardinality of the set $\mathcal{B}$ decreases, so for the formula, the value of θ can be lower. Finally, if $\|\mathbf{q}\|$ is high, the linear term of f becomes dominant, causing a faster convergence. Therefore we consider $\rho(\mathbf{Q})$, $\lambda^>_{min}(\mathbf{Q})$, $\|\mathbf{q}\|$, and $\dim(\ker(\mathbf{Q}))$, the more critical values concerning the function's conditioning.

The condition number of the domain $\mathcal{D}$, also known as its *eccentricity*, is determined by the quantity $\left(\frac{M}{\delta}\right)^2$, where M is the diameter and δ is the pyramidal width [11]. A higher eccentricity indicates a slower convergence [11]. The diameter of $\mathcal{D}$ is upper bounded by $\sqrt{2} \cdot K$, being $\mathcal{D}$ the Minkowski sum of K non-point unitary simplices, each one with a diameter of $\sqrt{2}$. The pyramidal width of a finite set of vertices is always greater than zero, decreasing when a new vertex is added [11]. Therefore, a few vertices may result in a higher pyramidal width. The number of vertices $|\mathcal{V}|$, being upper bounded by $(n/K)^K$, can grow more as K varies with respect to the value that the diameter can assume. For this reason, the number of vertices of the domain is the most critical value regarding the domain's conditioning. Ultimately, a low number of vertices indicates better domain conditioning.

5 Complexity analysis

In this section, we analyze the time complexity of the FW and AFW algorithms. The complexity of both algorithms is determined by the number of iterations, the computation of the descent direction, and the computation of the step size.

To compute the descent direction $\mathbf{d}^{(t)}$ for each iteration t, first, we need the computation of the function's gradient at the current point $\nabla f(\mathbf{x}^{(t)}) = 2\mathbf{Q}\mathbf{x}^{(t)} + \mathbf{q}^T\mathbf{x}$, which requires $O(n^2)$ time for the matrix-vector multiplication. After, we must compute the vector $\bar{\mathbf{y}}^{(t)}$ or the vector $\overset{+}{\mathbf{y}}^{(t)}$ for the away step, which select respectively the minimum and maximum components of $\nabla f(\mathbf{x}^{(t)})$ restricted to the indices of $\mathcal{I}^k$ for each $k \in \{1, \dots, K\}$. Recalling that $\left\{\mathcal{I}^k\right\}_{k=1,\dots,K}$ is a partition of the indices $\{1, \dots, n\}$, we must scan the entire gradient vector with $O(n)$ computational cost.

The computation of the step size $\alpha^{(t)}$ is given by the complexity of the exact line-search., which is $O(n^2)$ for the matrix-vector multiplication in the denominator of the Formula 7.

Thus, the total time complexity of the FW algorithm is $O(T \cdot (n^2 + n + n^2)) = O(T \cdot n^2)$, where T is the total number of iterations performed by the algorithm.

6 Experiments

6.1 Experimental setup

We perform experiments on quadratic functions $f(\mathbf{x})$, obtained by randomly generating the matrix $\mathbf{Q}$ and the vector $\mathbf{q}$ (Equation 1). The constrained domain $\mathcal{D}$ is obtained by randomly choosing the vertices of each polytope $\mathcal{P}^k$ (Proposition 2).

The Algorithm 6 shows how we randomly generate the matrix $\mathbf{Q}$. The algorithm takes as input the space dimension `n`, the kernel dimension `dim_ker`, the spectral radius $\rho > 0$, the smallest strictly positive eigenvalue $0 < \lambda^{>}_{min} \leq \rho$ and the density $0 < \delta \leq 1$. We generate the list of positive eigenvalues **rc** in the interval $[\lambda^{>}_{min}, \rho]$ (Line 6) and, next, we add the eigenvalue zero many times as the kernel dimension (Line 7). The possibility of setting $\lambda^{>}_{min}$ can affect the value assumed by the Hoffman constant θ, relevant for the convergence rate (see Section 4.2.1). If the desired density is less than one, we generate the positive semi-definite sparse matrix $\mathbf{Q}$ via the built-in function of `MATLAB` `sprandsym` [21] with the list of eigenvalues **rc** and density δ (Line 14). If the density equals one, we generate the positive semi-definite dense matrix $\mathbf{Q}$ by placing it congruent to the diagonal matrix with diagonal elements equal to the **rc** eigenvalues (from Line 8 to Line 12).

Algorithm 6 Matrix $\mathbf{Q}$ generation

```
1: procedure MatrixGeneration(seed, n, dim_ker, $\rho$, $\lambda^{>}_{min}$, $\delta$)
2:     rng(seed)
3:     if dim_ker $=$ n $-$ 1 then
4:         $\lambda^{>}_{min} \leftarrow \rho$
5:     end if
6:     Generate the vector $\mathbf{rc} \in [\lambda^{>}_{min}, \rho]^{\texttt{n}-\texttt{dim\_ker}}$
7:     $\mathbf{rc} \leftarrow [\mathbf{rc}, \mathbf{0}]$, with $\mathbf{0} \in \{0\}^{\texttt{dim\_ker}}$
8:     if $\delta = 1$ then
9:         Generate the dense orthogonal matrix $\mathbf{U}$
10:        $\mathbf{S} \leftarrow \texttt{diag}(\mathbf{rc})$
11:        $\mathbf{Q} \leftarrow \mathbf{U}^T \cdot \mathbf{S} \cdot \mathbf{U}$
12:        $\mathbf{Q} \leftarrow (\mathbf{Q} + \mathbf{Q}^T)/2$                ▷ Avoid numerical errors
13:    else
14:        $\mathbf{Q} \leftarrow \texttt{sprandsym}(\texttt{n}, \delta, \texttt{rc})$;
15:    end if
16: end procedure
```

We compactly represent the domain $\mathcal{D}$ by the matrix $\mathbf{P} \in \{0,1\}^{K\times n}$. The k-th row of the matrix $\mathbf{P}$ represents the subset of indices $\mathcal{I}^k = \{i_1, \dots, i_{n_k}\}$, hence the polytope $\mathcal{P}^k = \operatorname{conv}\left\{\mathbf{e}_{i_1}, \dots, \mathbf{e}_{i_{n_k}}\right\}$. More precisely, the logical matrix $\mathbf{P} = (p_{k,i})$ is such that

$$p_{k,i} = \begin{cases} 1 & \text{if } i \in \mathcal{I}^k \\ 0 & \text{otherwise.} \end{cases} \tag{15}$$

Therefore, we randomly generate the logical matrix $\mathbf{P}$ to generate the domain $\mathcal{D}$. We give, in addition, the option of selecting the number K of polytopes of which the domain is the Minkowski sum (Proposition 2).

Denoting the unconstrained optimum of the function f in $\mathbb{R}^n$ as $\mathbf{z}$, we can rewrite the global optimization task as

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{R}^n} (\mathbf{x}-\mathbf{z})^T\mathbf{Q}(\mathbf{x}-\mathbf{z}) &= \min_{\mathbf{x}\in\mathbb{R}^n} \mathbf{x}^T\mathbf{Q}\mathbf{x} - 2\mathbf{z}^T\mathbf{Q}\mathbf{x} + \mathbf{z}^T\mathbf{Q}\mathbf{z} \\ &= \min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}) \end{aligned}$$

setting $\mathbf{q}^T = -2\mathbf{z}^T\mathbf{Q}$. In this way, we can set a priori the unconstrained optimum, obtaining a solution $\mathbf{x}^* \in \mathcal{D}$ of our Task 1 with some specific characteristics. More precisely, we can choose, with enough precision, how many constraints are active for $\mathbf{x}^*$ in $\mathcal{D}$. We set $\beta \in [0,1]$ as the fraction of the number K of unitary simplices $\mathcal{P}^k$, which the unconstrained optimum $\mathbf{z} \in \mathbb{R}^n$ violates:

$$\mathbf{z}^* = \sum_{k=1}^K \mathbf{z}^k, \quad \text{and } \mathbf{z}^k \in \begin{cases} \mathbb{R}^n \setminus \mathcal{P}^k & \text{if } k \le \lfloor \beta K \rfloor \\ \operatorname{relint}(\mathcal{P}^k) & \text{otherwise.} \end{cases}$$

In this way, rewriting the solution $\mathbf{x}^* \in \mathcal{D}$ as

$$\mathbf{x}^* = \sum_{k=1}^K \mathbf{x}^k, \quad \text{with } \forall k \in \{1,\dots,K\}\ \mathbf{x}^k \in \mathcal{P}^k \text{ and } x_i^k = \begin{cases} x_i^* & \text{if } i \in \mathcal{I}^k \\ 0 & \text{otherwise,} \end{cases} \tag{16}$$

we expect that around $\lfloor \beta K \rfloor$ constraints are active and, therefore, the respectives $\mathbf{x}^k$ belong to the *relative boundary* of $\mathcal{P}^k$ denoted as $\mathcal{P}^k \setminus \operatorname{relint}(\mathcal{P}^k)$ [5]. We can be sure to obtain exactly $\lfloor \beta K \rfloor$ active constraints only by setting $\beta = 0$ because the unconstrained optimum is surely feasible and because $\text{relint}(\mathcal{P}^1) + \cdots + \text{relint}(\mathcal{P}^K) = \text{relint}(\mathcal{D})$, being $\mathcal{P}^1, \ldots, \mathcal{P}^K$ convex sets [19]. To randomly generate a point in a generic unary simplex $\mathcal{P}^k$, we use [22].

We choose as the initial point of the FW algorithm the vertex $\mathbf{x}^{(0)}$ belonging to the domain $\mathcal{D}$ such that

$$x_i^{(0)} = \begin{cases} 1 & \text{if } i = \min \mathcal{I}^k, k \in \{1, \ldots, K\} \\ 0 & \text{otherwise.} \end{cases}$$

For the stop criterion of the Frank-Wolfe type algorithms, we define the *relative duality gap* of the descent direction at step t as (refer to Eq. 8):

$$\frac{g_{\mathbf{x}^{(t)}}(\mathbf{d}^{(t)})}{\max\{1, |f(\mathbf{x}^{(t)})|\}}. \tag{17}$$

In the experiments, we denote the relative duality gap of the Frank-Wolfe type algorithms as the *dual error* of the approximation. Therefore, for the stop criterion of the FW algorithm, we stop the iterations when the dual error is less than ϵ_D, where ϵ_D is the desired relative precision required for the dual problem solution. In addition, we also set the maximum number of steps (`max_steps`) undertaken by the algorithm, preventing potential infinite loops.

To ascertain the accuracy of algorithm approximation within the primal problem, we compute the *relative error*, given by

$$\frac{f(\mathbf{x}^{(t)}) - f^*}{\max\{1, |f^*|\}}, \tag{18}$$

where f^* denotes the optimum derived with high precision using the `interior-point-convex` method provided by `MATLAB` [18]. Therefore, we call *primal error* the relative error of the approximation. A satisfactory approximation of the solution in the primal problem is generally characterized by a primal error in the order of 10^{-10}.

6.2 Experimental results and discussion

6.2.1 Cases where FW works

First, we run the FW algorithm, showing the different behaviour of the algorithm concerning the fact that the matrix $\mathbf{Q}$ is positive definite or semidefinite and the location of the optimal solution $\mathbf{x}^*$. If $\mathbf{Q}$ is positive definite and $\mathbf{x}^* \in \text{relint}(\mathcal{D})$, the FW algorithm converges linearly, as we can see from the Figure 1. After the first step of the algorithm, where the slope of the error curve decreases, the error decreases constantly in the log scale. The dual error (Eq. 17) becomes less than 10^{-7} in 47198 iterations and 3.93 seconds, obtaining an approximation of the solution with a primal error (Eq. 18) $\approx 10^{-11}$.

Therefore, the standard FW algorithm can converge linearly without using the away step, assuming that the optimal solution is in the relative interior of the domain and the objective function is strongly convex (Theorem 2).

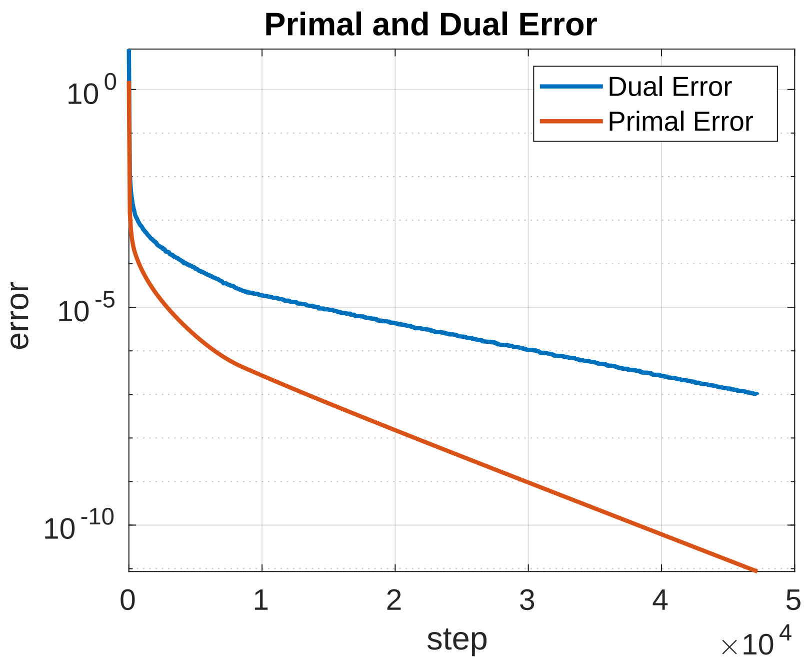


Figure 1: The Figure shows the dual error and the primal error curve of the FW algorithm. The instance parameters are $n = 100$, $K = 20$, $\beta = 0$, and as regards the matrix $\mathbf{Q}$, `dim_ker` $= 0$, $\rho = 2$, $\lambda^{>}_{min} = 1$ and $\delta = 1$. We set $\epsilon_D = 10^{-7}$ for the stop condition (see Eq. 17).

6.2.2 AFW overcomes the limitations of FW

Although the algorithm FW converges geometrically when the hypotheses of the Theorem 2 are satisfied, the algorithm AFW converges geometrically with a better constant. Figure 2 shows how both the dual error curve (left of Figure 2) and the primal error curve (right of Figure 2) of the AFW algorithm have a better slope than the error curves of the FW algorithm. Table 1 shows the better performance of the AFW algorithm compared to FW, in terms of time, number of steps and primal error of the approximation with the same stop condition $\epsilon_D = 10^{-7}$.

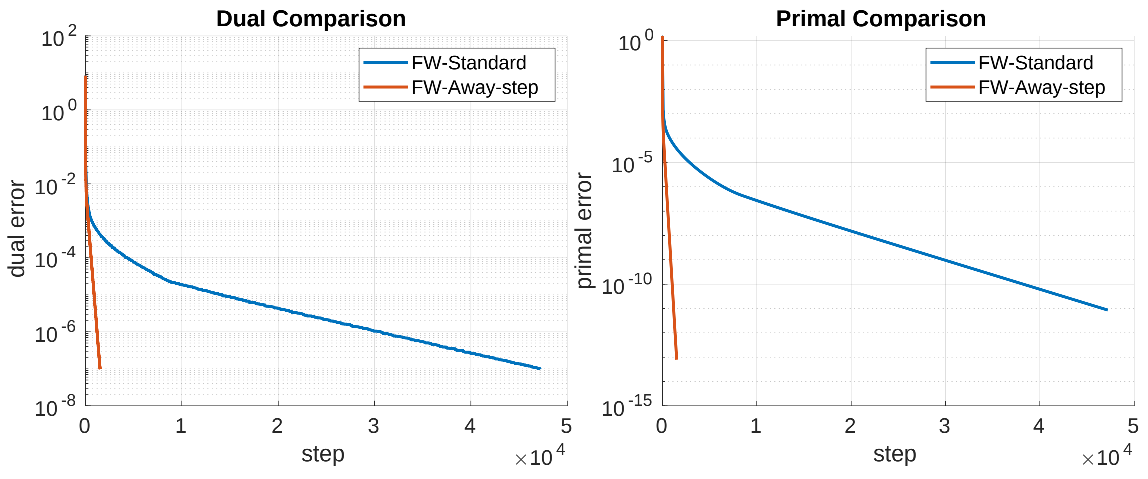


Figure 2: Comparison of the dual and primal error curves of the AFW and FW algorithms. The instance parameters and the stop condition are the same as the experiment shown in Figure 1.

n	K	β	`dim_ker`	Method	Steps	Time	Primal Error	$\lvert\mathcal{V}\rvert$	$\lVert\mathbf{q}\rVert$
100	20	0	0	FW	47198	3.52	$\approx 10^{-11}$	$\approx 10^{11}$	8.82
				AFW	1513	0.26	$\approx 10^{-13}$		

Table 1: Comparative performance analysis of FW and AFW algorithms based on Figure 2 instance.

If we relax the two conditions of the Theorem 2, the FW algorithm error decreases no more geometrically, and to obtain a linear convergence order, we have to apply the AFW algorithm. Let us see in detail the cases in which the away step is necessary to obtain linear convergence. First, we run the FW algorithm setting $\mathbf{Q}$ positive definite and $\beta = 0.5 > 0$, some subvectors $\mathbf{x}^k$ of $\mathbf{x}^*$ (see decomposition 16) belongs to the relative boundary of $\mathcal{P}^k$, therefore the condition that $\mathbf{x}^* \in \text{relint}(\mathcal{D})$ is relaxed. Being the solution in the relative boundary of the domain, so in a face having as extremes different vertices, the algorithm approaches the optimum changing every time the vertex goes toward, showing a zig-zagging behaviour and, therefore, a sublinear convergence. As we can see in Figure 3, the FW algorithm reaches the maximum number of steps set to 2000 without converging, with a dual error and a primal error in the same order of 10^{-3}. On the other hand, the AFW algorithm converges, reaching a dual error of 10^{-6} and a primal error $\approx 10^{-12}$ (see Table 2).

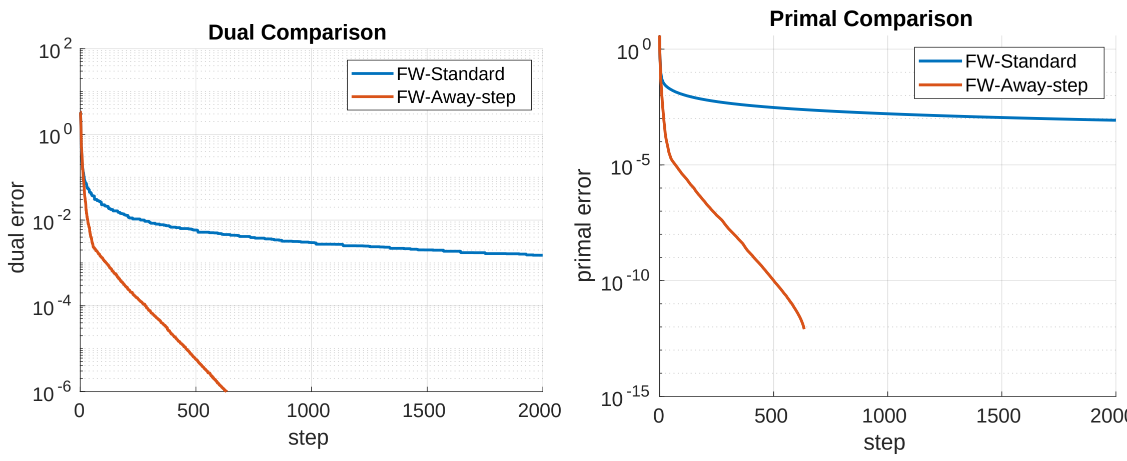


Figure 3: Sublinear convergence of the FW versus linear convergence of the AFW when the solution belongs to the relative boundary of the domain and the matrix $\mathbf{Q}$ is singular. The experiment instance parameters are $n = 100$, $K = 20$, $\beta = 0.5$, and as regards the matrix $\mathbf{Q}$, `dim_ker` $= 0$, $\rho = 2$, $\lambda_{min}^{>} = 1$ and $\delta = 1$. We set $\epsilon_D = 10^{-6}$ for the stop condition of the dual error and the `max_steps` $= 2000$.

n	K	β	`dim_ker`	Method	Steps	Time	Primal Error	$\lvert\mathcal{V}\rvert$	$\lVert\mathbf{q}\rVert$
100	20	0.5	0	FW	`max_steps`	0.16	$\approx 10^{-3}$	$\approx 10^{11}$	18.10
				AFW	634	0.11	$\approx 10^{-12}$		

Table 2: Comparative performance analysis of FW and AFW algorithms based on Figure 3 instance.

In the same way, if we keep the optimal solution in the relative interior of the domain but the matrix $\mathbf{Q}$ is singular ($\texttt{dim\_ker} > 0$), the function f is no more strongly-convex and the error of FW algorithm does not decrease geometrically; meanwhile the error of AFW decreases linearly (see Figure 4 and Table 3 for the performance analysis).

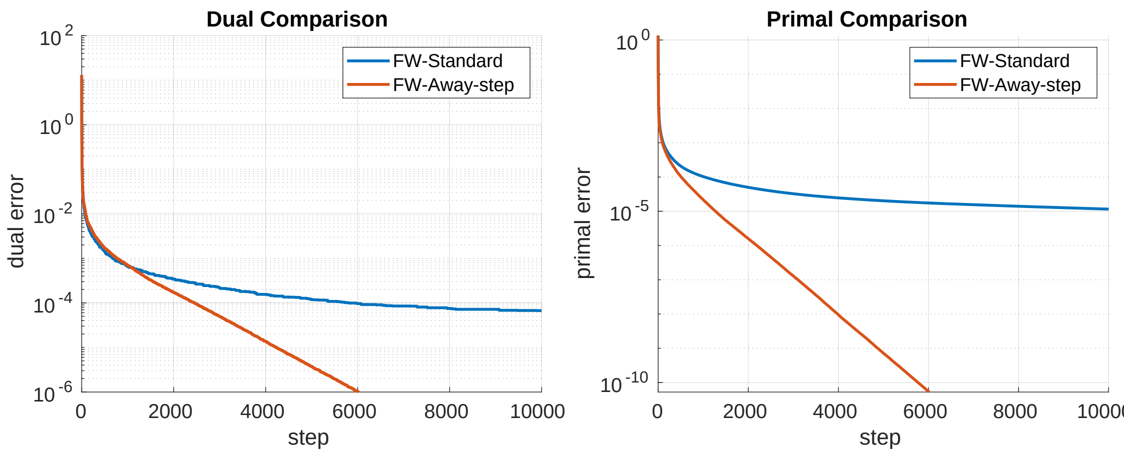
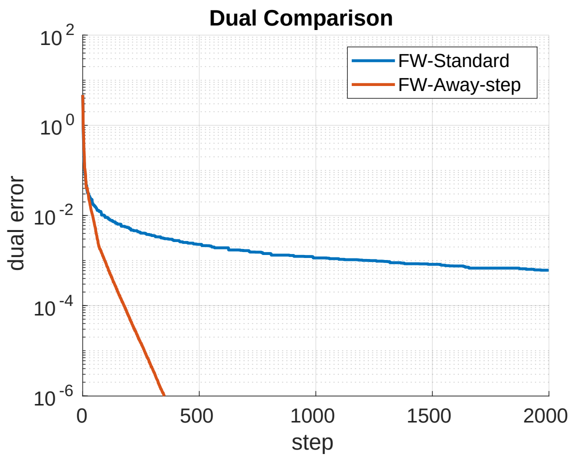


Figure 4: Sublinear convergence of the FW versus linear convergence of the AFW when the solution belongs to the relative interior of the domain but the matrix $\mathbf{Q}$ is not singular. The experiment instance parameters are $n = 100$, $K = 20$, $\beta = 0$, and as regards the matrix $\mathbf{Q}$, $\texttt{dim\_ker} = 10$, $\rho = 2$, $\lambda^{>}_{min} = 1$ and $\delta = 1$. We set $\epsilon_D = 10^{-6}$ for the stop condition of the dual error and the $\texttt{max\_steps} = 10000$.

n	K	β	`dim_ker`	Method	Steps	Time	Primal Error	$\lvert\mathcal{V}\rvert$	$\lVert\mathbf{q}\rVert$
100	20	0	10	FW	`max_steps`	0.76	$\approx 10^{-5}$	$\approx 10^{11}$	8.58
				AFW	6019	0.99	$\approx 10^{-10}$		

Table 3: Comparative performance analysis of FW and AFW algorithms based on Figure 4 instance.

Now we eliminate both hypotheses of Theorem 2, setting the matrix $\mathbf{Q}$ singular ($\texttt{dim\_ker>0}$) and the solution $\mathbf{x}^*$ belonging to the relative boundary of the domain setting ($\beta > 0$). Figure 5 shows the sublinear trend of the FW algorithm error curves; the algorithm stops without convergence, having reached the maximum number of steps. On the other hand, the AFW algorithm converges linearly in 351 steps and 0.05 seconds to the optimal solution $\mathbf{x}^*$, although the function f is not strongly-convex and the solution belongs to the relative boundary of the domain $\mathcal{D}$. For the performance analysis see Table 4.



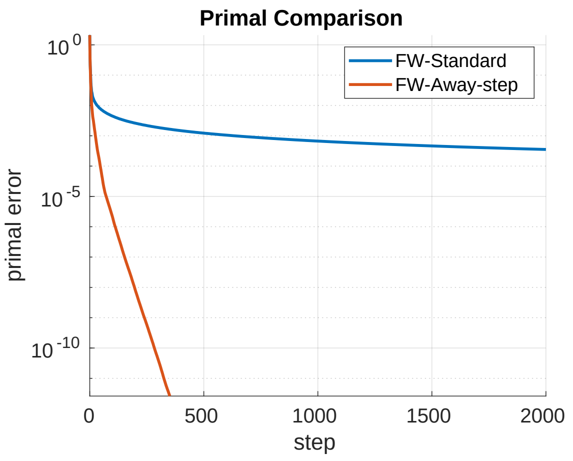


Figure 5: Sublinear convergence of the FW versus linear convergence of the AFW when the solution belongs to the relative interior of the domain but the matrix $\mathbf{Q}$ is not singular. The instance parameters are $n = 100$, $K = 10$, $\beta = 0.5$, and as regards the matrix $\mathbf{Q}$, $\texttt{dim\_ker} = 10$, $\rho = 2$, $\lambda_{min}^{>} = 1$ and $\delta = 1$. We set $\epsilon_D = 10^{-6}$ for the stop condition of the dual error and the $\texttt{max\_steps} = 2000$.

n	K	β	`dim_ker`	Method	Steps	Time	Primal Error	$\lvert\mathcal{V}\rvert$	$\lVert\mathbf{q}\rVert$
100	10	0.5	10	FW	`max_steps`	0.19	$\approx 10^{-4}$	$\approx 10^{9}$	17.24
				AFW	351	0.05	$\approx 10^{-12}$		

Table 4: Comparative performance analysis of FW and AFW algorithms based on Figure 5 instance.

6.2.3 Primal and dual behaviour

Comparing the results of FW with AFW, we notice that with the same dual error of the approximation, the primal error of the approximation provided by FW is greater. Therefore, from the primal point of view, the approximation provided by AFW is better with the same stop condition. At the same time, however, this means that if the approximations of the respective algorithms have similar primal errors, the FW approximation is better in the dual. We show this discrepancy in the behaviour of the two algorithms for instances where both FW and AFW converge linearly (Figure 6). However, we obtain the same behaviour even when FW converges sublinearly. Furthermore, the better the convergence rate of AFW compared to that of FW, the better the approximation of AFW in the primal problem. For example, in the first row of Figure 6, the difference between the convergence rates of AFW and FW is small (left plot), leading to a similar quality of the approximations in the primal (right plot). On the other hand, in the second row, the difference between the convergence rates of AFW and FW is more significant, and the quality of the approximation of AFW in the primal is much better than that of FW.

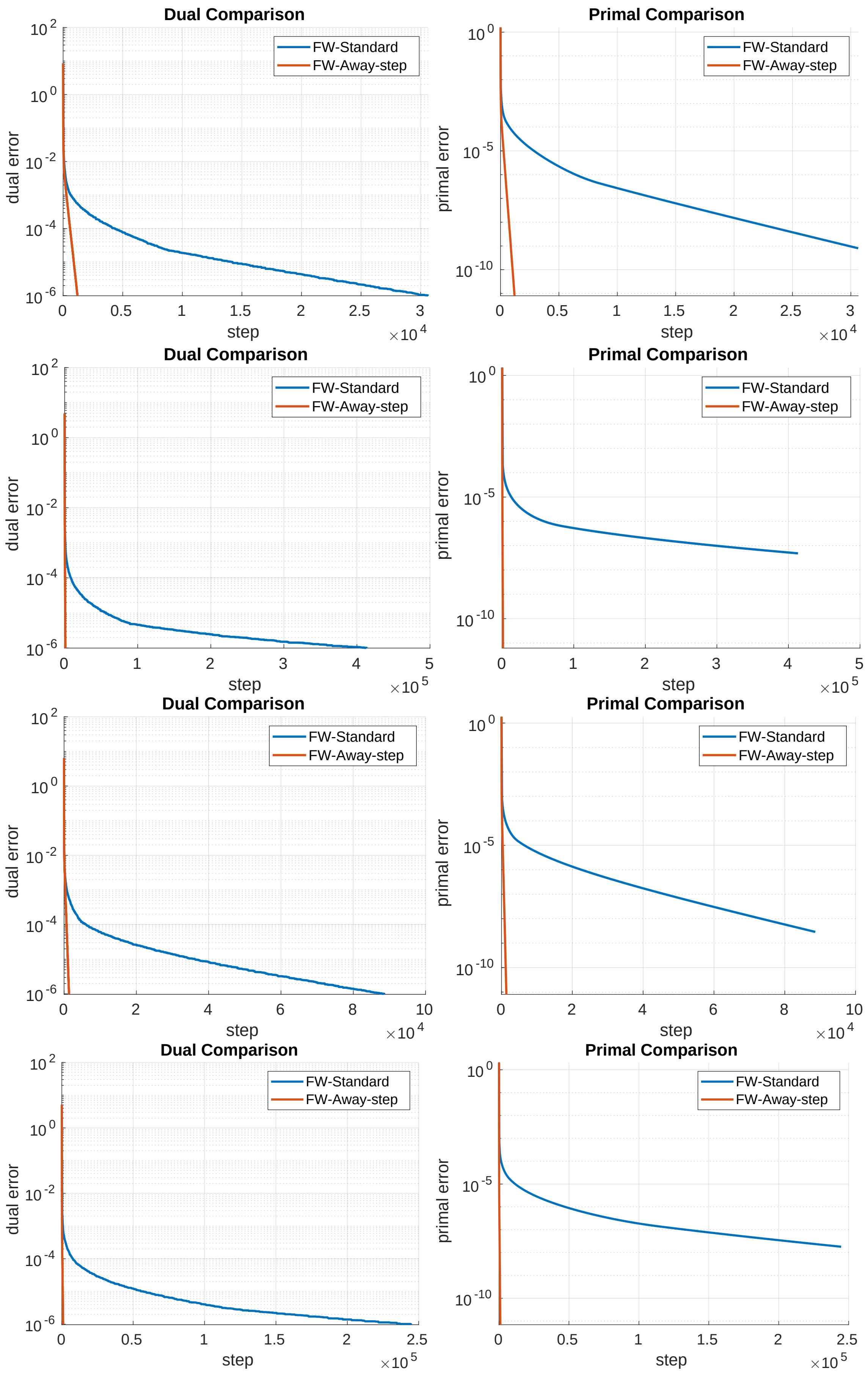


Figure 6: Better quality of the AFW approximation than that of FW from the primal perspective with the same dual error. The instance parameters and the stop condition are the same as the experiment shown in Figure 1. A different random seed is used for each experiment.

6.2.4 AFW scalability

We show the scalability of the AFW algorithm as the dimension of the space n and the spectral radius ρ of the matrix $\mathbf{Q}$ increase, both in the case where $\mathbf{Q}$ is non-singular and both in the case where `dim_ker` > 0. Table 5 shows how the algorithm's number of steps to converge increases as the space dimension increases. Surprisingly, however, as the spectral radius increases, the number of iterations remains similar; this can be motivated by the fact that as the spectral radius increases, $\|\mathbf{q}\|$ also increases since the vector $\mathbf{q}$ depends on the matrix $\mathbf{Q}$, being $\mathbf{q} = -2\mathbf{Qz}$, where $\mathbf{z}$ is the unconstrained optimum (Section 6.1). This phenomenon balances the negative effect of the greater spectral radius. Finally, as the size of the matrix kernel increases, the number of iterations decreases, as already noted in Section 4.2.

n	`dim_ker`	ρ	Steps	Time	Primal Error	$\lvert\mathcal{V}\rvert$	$\|\mathbf{q}\|$
10	0	10	71	0.01	$\approx 10^{-14}$	16	5.68
		100	83	0.01	$\approx 10^{-14}$	//	48.27
	1	10	56	0.01	$\approx 10^{-13}$	//	7.81
		100	18	0.01	$\approx 10^{-15}$	//	73.83
100	0	10	357	0.07	$\approx 10^{-12}$	$\approx 10^{11}$	73.03
		100	195	0.04	$\approx 10^{-12}$	//	700.00
	10	10	142	0.15	$\approx 10^{-12}$	//	69.78
		100	89	0.02	$\approx 10^{-12}$	//	668.74
1000	0	10	2856	14.75	$\approx 10^{-10}$	$\approx 10^{119}$	308.61
		100	3322	17.16	$\approx 10^{-10}$	//	2.93×10^3
	100	10	1017	5.44	$\approx 10^{-10}$	//	294.00
		100	2438	12.60	$\approx 10^{-10}$	//	2.80×10^3

Table 5: Performance scalability of the AFW algorithm when n, `dim_ker` and ρ vary. The values assumed by the parameters n, `dim_ker` and ρ are in the first three columns of the Table. For each experiment we fix $K = 0.2 \times n$, $\beta = 0.5$, and as regards the matrix $\mathbf{Q}$ we fix $\lambda^{>}_{min} = 1$ and $\delta = 1$. The stop condition ϵ_D for the dual error is set to 10^{-6}. The plots of the error curves of the experiments listed in the Table are in Plots A.1.

6.2.5 AFW on sparce matrices

The behaviour of the AFW algorithm can be sensitive to the value of the smallest strictly positive eigenvalue $\lambda^{>}_{min}$ of the matrix $\mathbf{Q}$. In particular, as already seen in Section 4.2, the Hoffman constant can grow if $\lambda^{>}_{min}$ is small and the matrix $\mathbf{Q}$ is sparse, slowing down the convergence of the AFW algorithm (Table 6). The worst case occurs when $\lambda^{>}_{min}$ is very small and the matrix $\mathbf{Q}$ is diagonal (setting $\delta = \frac{1}{n}$ in Algorithm 6). Thus, even if storing the matrix $\mathbf{Q}$ in sparse form causes a shorter time in the execution of a single iteration of the AFW algorithm thanks to the lower time complexity of matrix-vector multiplication, in the case where $\lambda^{>}_{min}$ is very small the number of iterations grows a lot increasing the total execution time of the algorithm.

$\lambda_{min}^{>}$	δ	Steps	Time	Primal Error	$\lvert\mathcal{V}\rvert$	$\lVert\mathbf{q}\rVert$
5	1	351	0.08	$\approx 10^{-12}$	$\approx 10^{11}$	86.19
	0.1	152	0.03	$\approx 10^{-13}$	//	79.67
	0.01	3235	0.52	$\approx 10^{-11}$	//	80.39
0.1	1	89	0.02	$\approx 10^{-12}$	//	66.87
	0.1	2480	0.41	$\approx 10^{-10}$	//	60.07
	0.01	3160	0.50	$\approx 10^{-10}$	//	60.22
0.001	1	95	0.02	$\approx 10^{-12}$	//	66.58
	0.1	2630	0.44	$\approx 10^{-11}$	//	59.79
	0.01	4372	0.68	$\approx 10^{-10}$	//	59.94

Table 6: Performance of the AFW algorithm with sparse matrices. The values assumed by the parameters $\lambda_{min}^{>}$ and δ regarding the matrix $\mathbf{Q}$ are in the first two columns of the Table. For each experiment we fix $n = 100$, $K = 20$, $\beta = 0.5$, and as regards the matrix $\mathbf{Q}$ we fix `dim_ker` $= 10$ and $\rho = 10$. The stop condition ϵ_D for the dual error is set to 10^{-6}. The plots of the error curves of the experiments listed in the Table are in Plots A.2.

6.2.6 Position of the solution and domain conditioning

The characteristics of the domain $\mathcal{D}$ and the position of the optimal solution $\mathbf{x}^*$ with respect to the domain influence the convergence of the AFW algorithm. Fixing $n = 100$, we vary the number of unitary simplices K and the fraction β of the number of simplices for which the respective subvectors of the solution belong to the relative edges.

Possible values of β are 0, 0.5, 1 (Table 7). In the case in which $\beta = 0$, the solution belongs to the relative interior of the domain, while if $\beta > 0$, approximately $\lfloor \beta K \rfloor$ constraints are active (see 16). Possible values of K are 1, $\frac{n}{10}$, $\lfloor e^{-1} n \rfloor$ and $\frac{n}{2}$ (Table 7). In case $K = 1$, the domain coincides with the unitary simplex Δ^n, and limiting β to values 0 and 1 due to the single constraint; the number of vertices $\lvert\mathcal{V}\rvert$ is minimum and equal to n. In case $K = \lfloor e^{-1} n \rfloor = 36$, the upper bound of the number of vertices of the domain is maximized (Proposition 3). In the last case where $K = \frac{n}{2}$, the simplices all have dimension two and are as small as possible (we only consider simplices that have at least two vertices, Section 2).

We also directly calculate the number of vertices of the domain, knowing that it is equal to the product of the number of vertices of each simplex. In the case $K = \frac{n}{2}$, $\lvert\mathcal{V}\rvert = 2^{\frac{n}{2}}$ turns out to be maximum when the values of K vary. As the number of vertices increases, the number of iterations performed for convergence increases (Table 7), as shown in Section 4.2 with the conditional number of the domain. However, the parameter that most influences the convergence is the value of β (Table 7). If the optimal solution is in the relative interior of the domain ($\beta = 0$), the AFW algorithm converges more slowly than when $\mathbf{x}^*$ belongs to the relative boundary of the domain (β=1) or when part of it belongs to the relative boundaries of the respective simplexes ($\beta = 0.5$).

K	β	Steps	Time	Primal Error	$\lvert\mathcal{V}\rvert$	$\lVert\mathbf{q}\rVert$
1	0	19603	0.82	$\approx 10^{-10}$	100	1.66
	1	12	0.01	$\approx 10^{-12}$	//	178.98
10	0	24805	2.49	$\approx 10^{-9}$	$\approx 10^{9}$	19.16
	0.5	258	0.03	$\approx 10^{-12}$	//	80.46
	1	97	0.02	$\approx 10^{-12}$	//	156.58
36	0	1649	0.43	$\approx 10^{-11}$	$\approx 10^{14}$	55.62
	0.5	72	0.02	$\approx 10^{-12}$	//	112.46
	1	84	0.03	$\approx 10^{-12}$	//	140.60
50	0	259	0.10	$\approx 10^{-12}$	$\approx 10^{15}$	66.93
	0.5	254	0.10	$\approx 10^{-12}$	//	113.21
	1	341	0.13	$\approx 10^{-11}$	//	147.82

Table 7: Performance of the AFW algorithm with respect to different characteristics of the domain. The values assumed by the parameters K and β are in the first two columns of the Table. For each experiment we fix $n = 100$, and as regards the matrix $\mathbf{Q}$ we fix `dim_ker` $= 10$, $\rho = 10$, $\lambda_{min}^{>} = 1$ and $\delta = 1$. The stop condition ϵ_D for the dual error is set to 10^{-6}. The plots of the error curves of the experiments listed in the Table are in Plots A.3.

6.2.7 AFW vs. off-the-shelf

In this section, we conduct a comparative analysis of the performance and behavior of the AFW algorithm and the MATLAB `interior-point-convex` (IPC) algorithm, the latter of which is readily accessible through the `quadprog` function [18]. The IPC algorithm is the default solver in `quadprog` for minimizing quadratic objective functions subject to linear constraints [6].

For the execution of the AFW algorithm, we set the dual error tolerance, ϵ_D, to 10^{-6} (see Definition 17). In contrast, when implementing the IPC algorithm via `quadprog`, we adjust the optimization settings, specifically setting the `OptimalityTolerance`, `ConstraintTolerance`, and `StepTolerance` parameters to 10^{-9}, utilizing the `optimoptions` function for this purpose [14], [23]. It is important to note that the IPC algorithm lacks a direct measure for assessing the approximation error from a dual perspective. Consequently, in our comparative analysis, we focus solely on the primal error comparison between the two algorithms (see Definition 18).

We perform several experiments, structured into three distinct sets, to analyze the behaviour of the AFW and IPC algorithms across varying space dimension n. Specifically, we categorize the space dimension into three sizes: small ($n = 100$, Section 6.2.7.1), medium ($n = 600$, Section 6.2.7.2), and large ($n = 3600$, Section 6.2.7.3). We vary the number of simplices K and the coefficient β within each set of experiments. This approach enables a comprehensive comparison of the AFW and IPC algorithms, taking into account variations in the characteristics of the domain $\mathcal{D}$, as well as different positional relationships of the optimal solution $\mathbf{x}^*$ with respect to the domain. The values for K are 1, $\frac{n}{10}$, $\lfloor e^{-1}n \rfloor$, and $\frac{n}{2}$, while the values for β are 0, 0.5, and 1. A detailed discussion regarding the selection of these specific values for K and β are provided in Section 6.2.6.

In all experiments (Tables 8, 9, 10), the primal quality of the approximations provided by IPC and AFW is excellent, thanks to the appropriately chosen tolerances. The primal quality of the AFW algorithm's approximations exhibits a direct correlation with its convergence rate. Specifically, a higher rate of convergence results in a diminished primal error, whereas a slower convergence rate leads to an augmentation in the primal error.

We note that while the IPC algorithm takes a small and approximately constant number of iterations to converge, the number of iterations of the AFW algorithm can be very high due to linear convergence and the variation of the respective convergence rate. At the same time, however, AFW performs more iterations per unit of time than IPC, thanks to the time complexity of $O(n^2)$ per iteration of AFW.

Regarding the experiments with $n = 100$ (Table 8), AFW turns out to be performing using similar times to IPC. For $n = 600$ (Table 9), the use of AFW is reasonable only for $\beta > 0$, while for $n = 3600$ (Table 10), AFW is competitive only for $K = 1$ and $\beta = 1$. As n increases, AFW loses some of its efficiency due to the increasing number of iterations T required for convergence. From the experiments, we can conjecture that for AFW the steps required to converge are $T = O(g(n))$, with g not a constant function, while for IPC it seems to be $T = O(1)$. So, ultimately, AFW is a good option for instances of our task with small space dimension; meanwhile, it converges faster than IPC for large space dimension only when the domain coincides with the unit simplex Δ^n and the solution is in its relative interior.

6.2.7.1 Small size dimension

K	β	Method	Steps	Time	Primal Error	$\lvert\mathcal{V}\rvert$	$\lVert\mathbf{q}\rVert$
1	0	IPC	8	0.04	$\approx 10^{-11}$	100	1.66
		AFW	19603	0.84	$\approx 10^{-10}$		
//	1	IPC	11	0.01	$\approx 10^{-8}$	//	178.98
		AFW	12	0.01	$\approx 10^{-12}$		
10	0	IPC	10	0.02	$\approx 10^{-12}$	$\approx 10^{9}$	19.16
		AFW	24805	2.57	$\approx 10^{-9}$		
//	0.5	IPC	8	0.02	$\approx 10^{-13}$	//	80.46
		AFW	258	0.03	$\approx 10^{-12}$		
//	1	IPC	11	0.01	$\approx 10^{-11}$	//	156.58
		AFW	97	0.02	$\approx 10^{-12}$		
36	0	IPC	9	0.01	$\approx 10^{-13}$	$\approx 10^{14}$	55.63
		AFW	1649	0.48	$\approx 10^{-11}$		
//	0.5	IPC	9	0.01	$\approx 10^{-12}$	//	112.46
		AFW	72	0.03	$\approx 10^{-12}$		
//	1	IPC	7	0.01	$\approx 10^{-9}$	//	140.59
		AFW	84	0.03	$\approx 10^{-12}$		
50	0	IPC	6	0.01	$\approx 10^{-13}$	$\approx 10^{15}$	66.93
		AFW	259	0.1	$\approx 10^{-12}$		
//	0.5	IPC	8	0.01	$\approx 10^{-11}$	//	113.21
		AFW	254	0.1	$\approx 10^{-12}$		
//	1	IPC	10	0.01	$\approx 10^{-10}$	//	147.82
		AFW	341	0.13	$\approx 10^{-12}$		

Table 8: Performance comparison between the AFW algorithm and the IPC algorithm. The values assumed by the parameters K and β are in the first two columns of the Table. For each experiment we fix $n = 100$, and as regards the matrix $\mathbf{Q}$ we fix `dim_ker` $= 10$, $\rho = 10$, $\lambda_{min}^{>} = 1$ and $\delta = 1$. The plots of the error curves of the experiments listed in the Table are in Plots A.4.1.

6.2.7.2 Medium size dimension

K	β	Method	Steps	Time	Primal Error	$\lvert\mathcal{V}\rvert$	$\lVert\mathbf{q}\rVert$
1	0	IPC	8	0.05	$\approx 10^{-12}$	600	0.70
		AFW	53489	19.55	$\approx 10^{-10}$		
//	1	IPC	11	0.05	$\approx 10^{-10}$	//	438.00
		AFW	14	0.01	$\approx 10^{-13}$		
60	0	IPC	11	0.07	$\approx 10^{-11}$	$\approx 10^{52}$	49.81
		AFW	70386	70.07	$\approx 10^{-9}$		
//	0.5	IPC	13	0.06	$\approx 10^{-12}$	//	245.45
		AFW	615	0.64	$\approx 10^{-11}$		
//	1	IPC	11	0.05	$\approx 10^{-9}$	//	322.08
		AFW	520	0.54	$\approx 10^{-11}$		
220	0	IPC	9	0.07	$\approx 10^{-15}$	$\approx 10^{87}$	137.02
		AFW	8233	18.49	$\approx 10^{-11}$		
//	0.5	IPC	12	0.10	$\approx 10^{-12}$	//	275.34
		AFW	3124	7.09	$\approx 10^{-10}$		
//	1	IPC	11	0.09	$\approx 10^{-11}$	//	337.54
		AFW	2345	5.23	$\approx 10^{-10}$		
300	0	IPC	8	0.08	$\approx 10^{-13}$	$\approx 10^{90}$	160.72
		AFW	1810	5.46	$\approx 10^{-11}$		
//	0.5	IPC	9	0.08	$\approx 10^{-11}$	//	259.28
		AFW	639	1.89	$\approx 10^{-11}$		
//	1	IPC	10	0.08	$\approx 10^{-14}$	//	338.77
		AFW	416	1.19	$\approx 10^{-11}$		

Table 9: Performance comparison between the AFW algorithm and the IPC algorithm. The values assumed by the parameters K and β are in the first two columns of the Table. For each experiment we fix $n = 600$, and as regards the matrix $\mathbf{Q}$ we fix `dim_ker` $= 60$, $\rho = 10$, $\lambda^{>}_{min} = 1$ and $\delta = 1$. The plots of the error curves of the experiments listed in the Table are in Plots A.4.2.

6.2.7.3 Large size dimension

K	β	Method	Steps	Time	Primal Error	$\lvert\mathcal{V}\rvert$	$\lVert\mathbf{q}\rVert$
1	0	IPC	8	2.37	$\approx 10^{-10}$	3600	0.28
		AFW	175261	1.06×10^4	$\approx 10^{-9}$		
//	1	IPC	14	3.71	$\approx 10^{-8}$	//	401.49
		AFW	27	1.57	$\approx 10^{-12}$		
360	0	IPC	11	3.81	$\approx 10^{-13}$	$\approx 10^{304}$	120.72
		AFW	464893	2.74×10^4	$\approx 10^{-8}$		
//	0.5	IPC	12	4.05	$\approx 10^{-13}$	//	550.03
		AFW	3957	228.91	$\approx 10^{-10}$		
//	1	IPC	15	4.93	$\approx 10^{-15}$	//	777.46
		AFW	3377	192.82	$\approx 10^{-10}$		
1324	0	IPC	32	17.68	$\approx 10^{-13}$	$>$ `realmax`	324.30
		AFW	51669	3.34×10^3	$\approx 10^{-10}$		
//	0.5	IPC	11	6.50	$\approx 10^{-12}$	//	585.79
		AFW	6575	422.03	$\approx 10^{-10}$		
//	1	IPC	11	6.56	$\approx 10^{-13}$	//	786.67
		AFW	4521	288.06	$\approx 10^{-10}$		
1800	0	IPC	23	16.24	$\approx 10^{-16}$	$>$ `realmax`	394.16
		AFW	4458	296.62	$\approx 10^{-10}$		
//	0.5	IPC	11	8.35	$\approx 10^{-12}$	//	655.23
		AFW	8105	537.70	$\approx 10^{-10}$		
//	1	IPC	10	7.76	$\approx 10^{-12}$	//	783.33
		AFW	509	34.09	$\approx 10^{-12}$		

Table 10: Performance comparison between the AFW algorithm and the IPC algorithm. The values assumed by the parameters K and β are in the first two columns of the Table. For each experiment we fix $n = 3600$, and as regards the matrix $\mathbf{Q}$ we fix `dim_ker` $= 360$, $\rho = 10$, $\lambda_{min}^{>} = 1$ and $\delta = 1$. The plots of the error curves of the experiments listed in the Table are in Plots A.4.3.

References

[1] Amir Beck and Shimrit Shtern. “Linearly convergent away-step conditional gradient for non-strongly convex functions”. In: *Mathematical Programming* 164 (2017), pp. 1–27.

[2] Immanuel M. Bomze, Francesco Rinaldi, and Damiano Zeffiro. “Active Set Complexity of the Away-Step Frank–Wolfe Algorithm”. In: *SIAM Journal on Optimization* 30.3 (2020), pp. 2470–2500. DOI: 10.1137/19M1309419. eprint: https://doi.org/10.1137/19M1309419. URL: https://doi.org/10.1137/19M1309419.

[3] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. 3.1.4 Second-order conditions. Cambridge University Press, 2004, p. 71.

[4] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. 3.1.3 First-order conditions. Cambridge University Press, 2004, p. 69.

[5] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. 2.3.1 Affine dimension and relative interior. Cambridge University Press, 2004, p. 23.

[6] *Choosing the algorithm*. Choosing the algorithm by MATLAB. URL: https://uk.mathworks.com/help/optim/ug/choosing-the-algorithm.html (visited on 06/23/2023).

[7] Patrick M. Fitzpatrick. *Advanced calculus*. 2nd ed. Bolzano-Weierstrass theorem. American Mathematical Society, 2006, p. 300.

[8] Komei Fukuda. “From the zonotope construction to the Minkowski addition of convex polytopes”. In: *Journal of Symbolic Computation* 38.4 (2004). Symbolic Computation in Algebra and Geometry, pp. 1261–1272. ISSN: 0747-7171. DOI: https://doi.org/10.1016/j.jsc.2003.08.007. URL: https://www.sciencedirect.com/science/article/pii/S0747717104000409.

[9] J Guélat and P Marcotte. “Some Comments of Wolfe’s ‘away Step’”. In: *Math. Program.* 35.1 (May 1986), pp. 110–119. ISSN: 0025-5610.

[10] Martin Jaggi. “Revisiting Frank-Wolfe: Projection-Free Sparse Convex Optimization”. In: Proceedings of Machine Learning Research 28.1 (2013). Ed. by Sanjoy Dasgupta and David McAllester, pp. 427–435. URL: https://proceedings.mlr.press/v28/jaggi13.html (visited on 03/06/2022).

[11] Simon Lacoste-Julien and Martin Jaggi. “On the Global Linear Convergence of Frank-Wolfe Optimization Variants”. In: *NIPS 2015 - Advances in Neural Information Processing Systems 28*. Montreal, Canada, Dec. 2015. URL: https://inria.hal.science/hal-01248675.

[12] E.S. Levitin and Boris Polyak. “Constrained Minimization Methods”. In: *USSR Computational Mathematics and Mathematical Physics* 6 (Dec. 1966), pp. 1–50. DOI: 10.1016/0041-5553(66)90114-5.

[13] Frank M. and Wolfe P. “An algorithm for quadratic programming”. In: *Naval Research Logistics Quarterly* 3.1–2 (1956), pp. 95–110. URL: https://doi.org/10.1002/nav.3800030109.

[14] *optimoptions*. Create optimization options. URL: https://uk.mathworks.com/help/optim/ug/optim.problemdef.optimizationproblem.optimoptions.html (visited on 06/23/2023).

[15] Wolfe P. “Convergence theory in nonlinear programming”. In: (1970). Ed. by Abadie J., pp. 1–36.

[16] Javier Pena and Daniel Rodriguez. *Polytope conditioning and linear convergence of the Frank-Wolfe algorithm*. Aug. 2018. URL: https://doi.org/10.1287/moor.2017.0910.

[17] Javier Penã, Daniel Rodriguez, and Negar Soheili. “On the von Neumann and Frank–Wolfe Algorithms with Away Steps”. In: *SIAM Journal on Optimization* 26.1 (2016), pp. 499–512. DOI: 10.1137/15M1009937. eprint: https://doi.org/10.1137/15M1009937. URL: https://doi.org/10.1137/15M1009937.

[18] *quadprog*. Quadratic programming by MATLAB. URL: https://uk.mathworks.com/help/optim/ug/quadprog.html (visited on 06/23/2023).

[19] R. Tyrrell Rockafellar. *Convex Analysis*. Corollary 6.6.2. Princeton University Press, 1997. ISBN: 978-0-691-01586-6.

[20] Walter Rudin. *Principles of Mathematical Analysis*. 3rd ed. Extreme value theorem, generalization to metric space. McGraw Hill, 1976, pp. 89–90.

[21] *sprandsym*. Sparse symmetric random matrix by MATLAB. URL: https://uk.mathworks.com/help/matlab/ref/sprandsym.html (visited on 06/23/2023).

[22] Roger Stafford. *randfixedsum*. Random Vectors with Fixed Sum by MATLAB Central File Exchange. URL: https://www.mathworks.com/matlabcentral/fileexchange/9700-random-vectors-with-fixed-sum (visited on 06/23/2023).

[23] *Tolerance details*. URL: https://uk.mathworks.com/help/optim/ug/tolerance-details.html (visited on 06/23/2023).

A Plots

A.1 AFW scalability

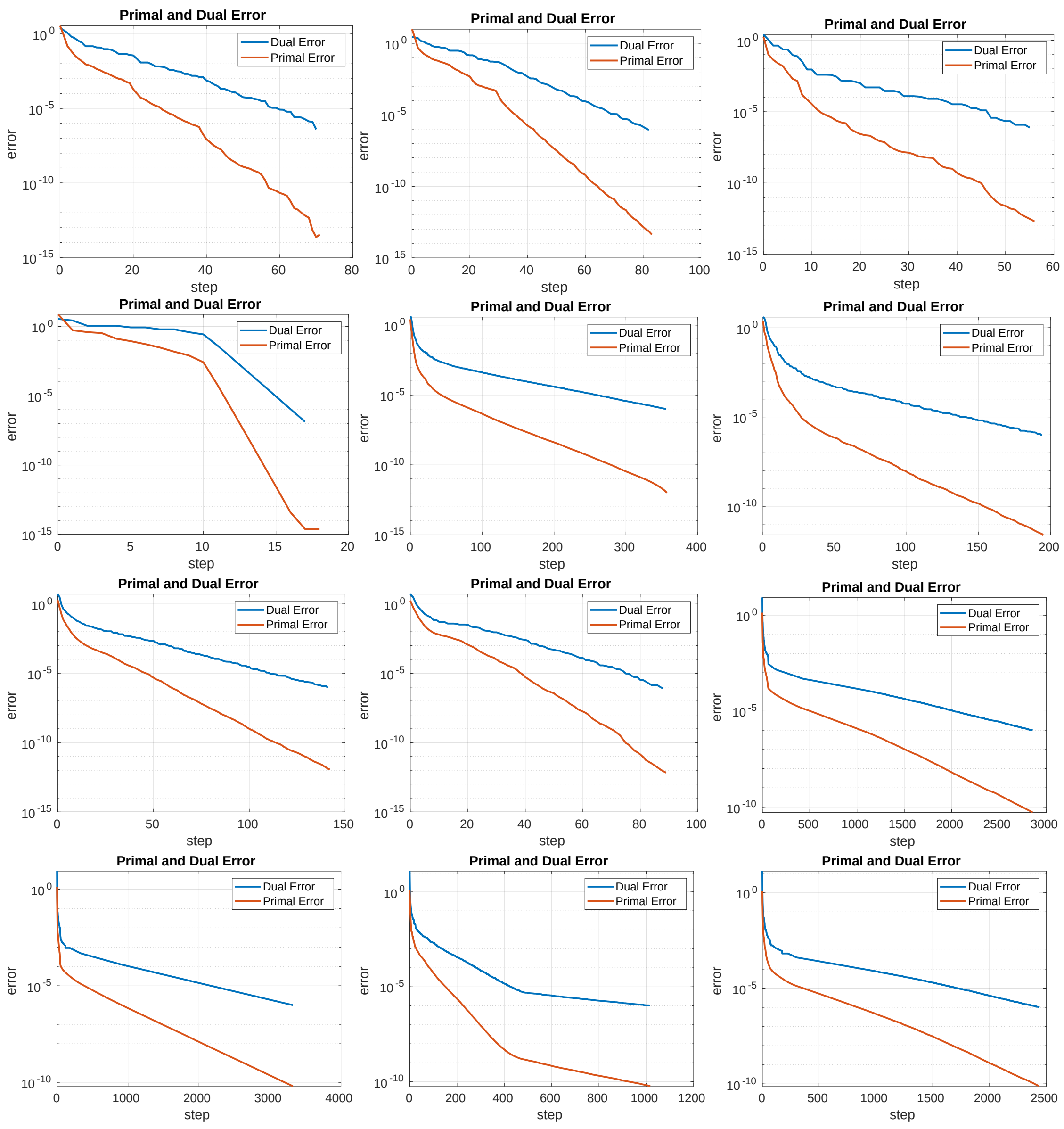


Figure 7: The Figure shows, in "left-to-right then top-to-bottom" order, the plots of the error curves of the experiments listed in Table 5.

A.2 AFW on sparse matrices

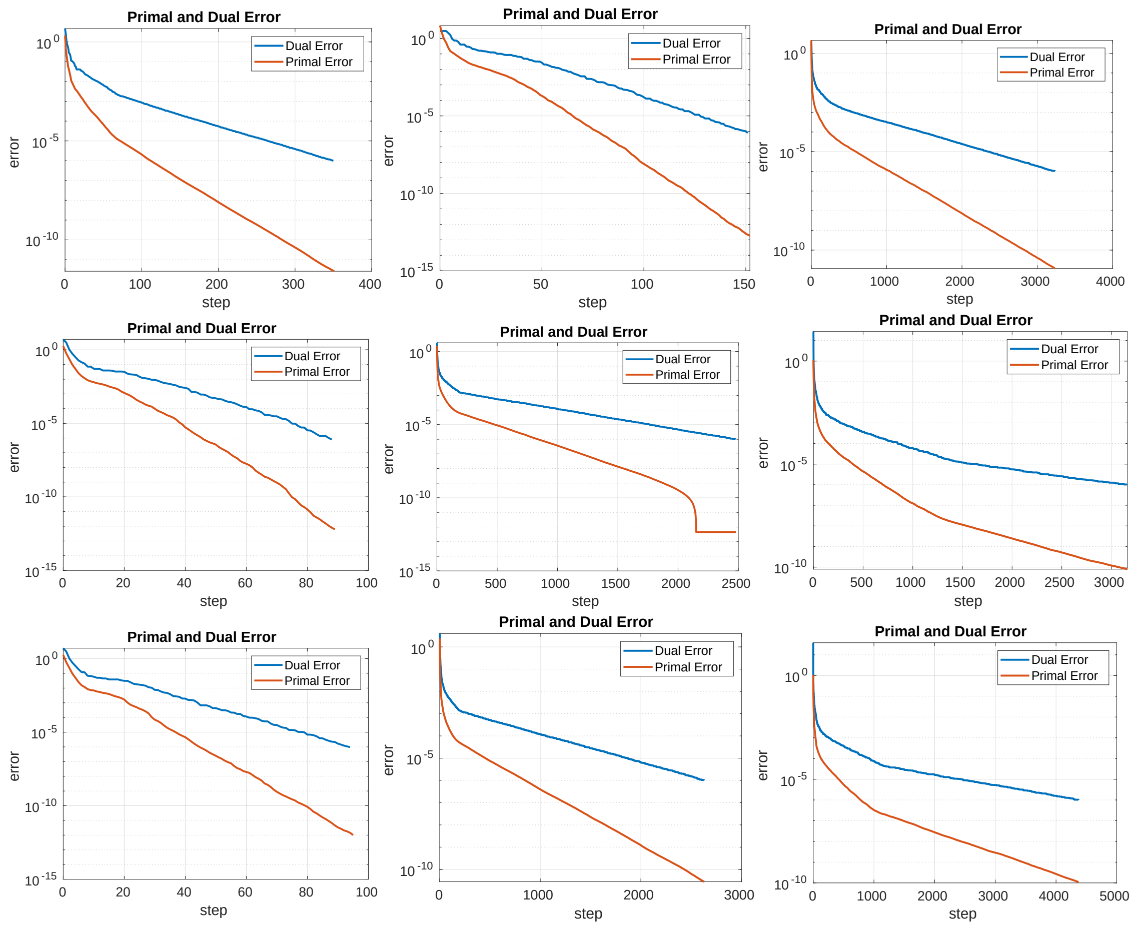


Figure 8: The Figure shows, in "left-to-right then top-to-bottom" order, the plots of the error curves of the experiments listed in Table 6.

A.3 Position of the solution and domain conditioning

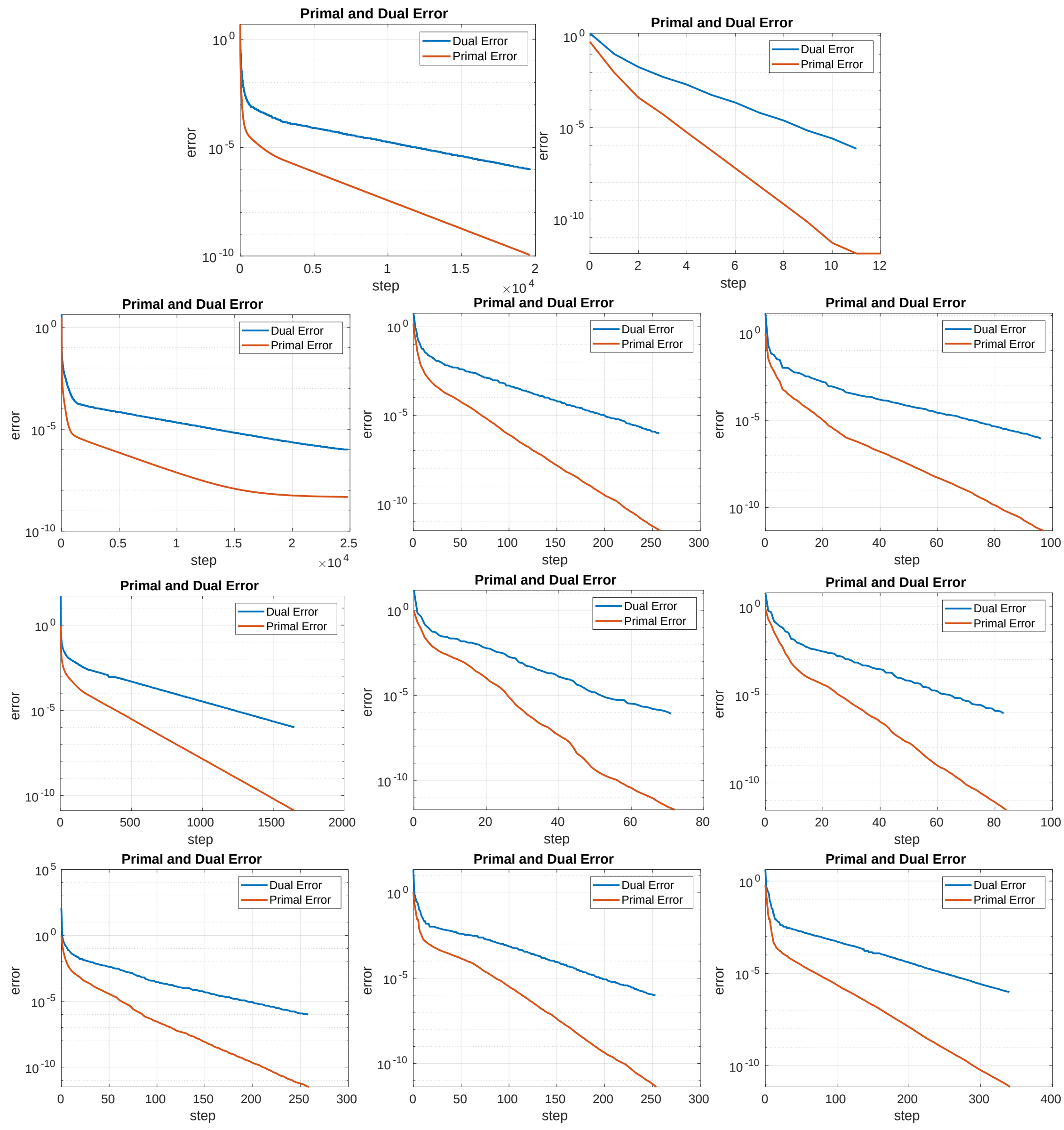


Figure 9: The Figure shows, in "left-to-right then top-to-bottom" order, the plots of the error curves of the experiments listed in Table 7.

A.4 AFW vs off-the-shelf

A.4.1 Small size dimension

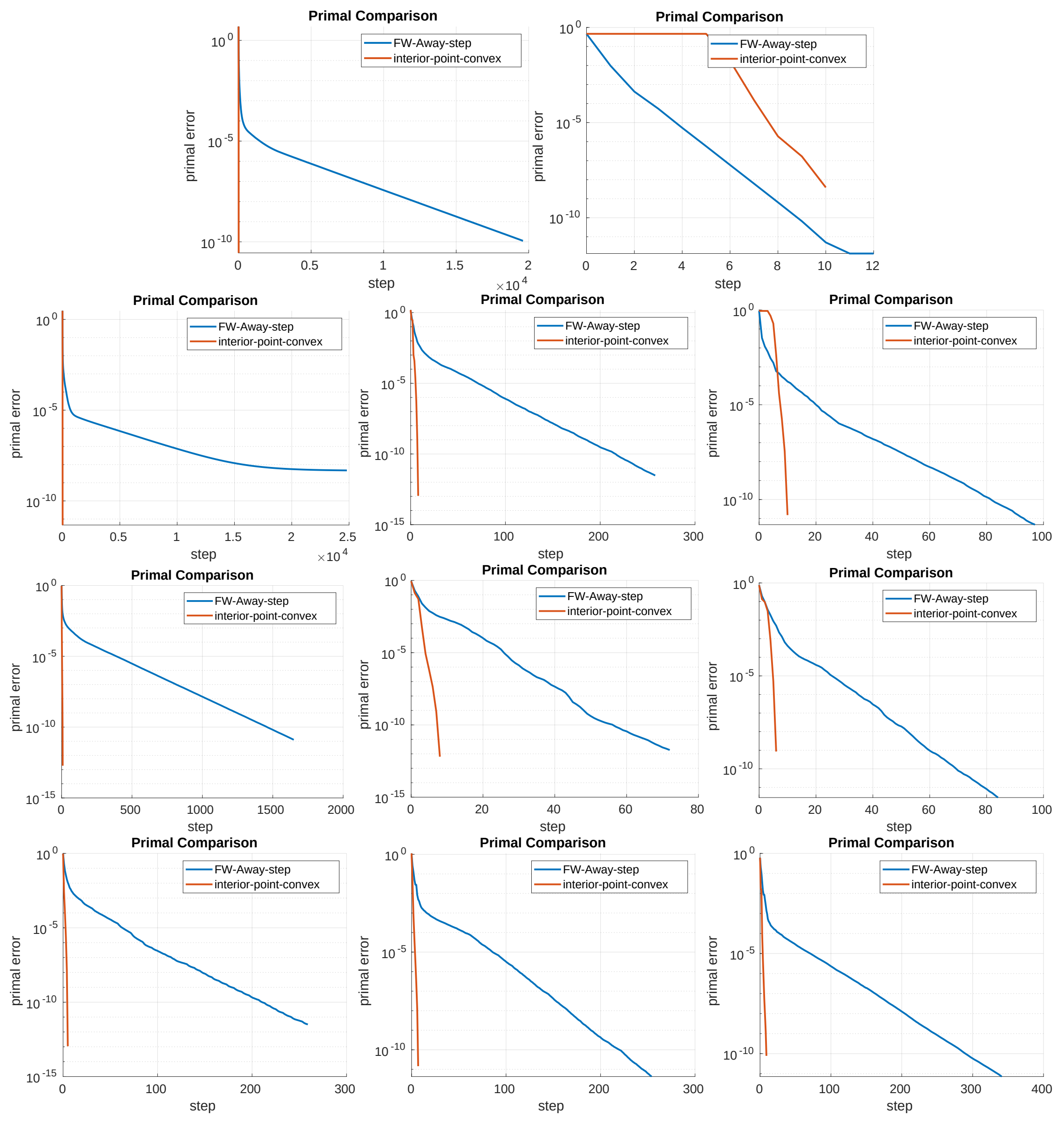


Figure 10: The Figure shows, in "left-to-right then top-to-bottom" order, the plots of the error curves of the experiments listed in Table 8.

A.4.2 Medium size dimension

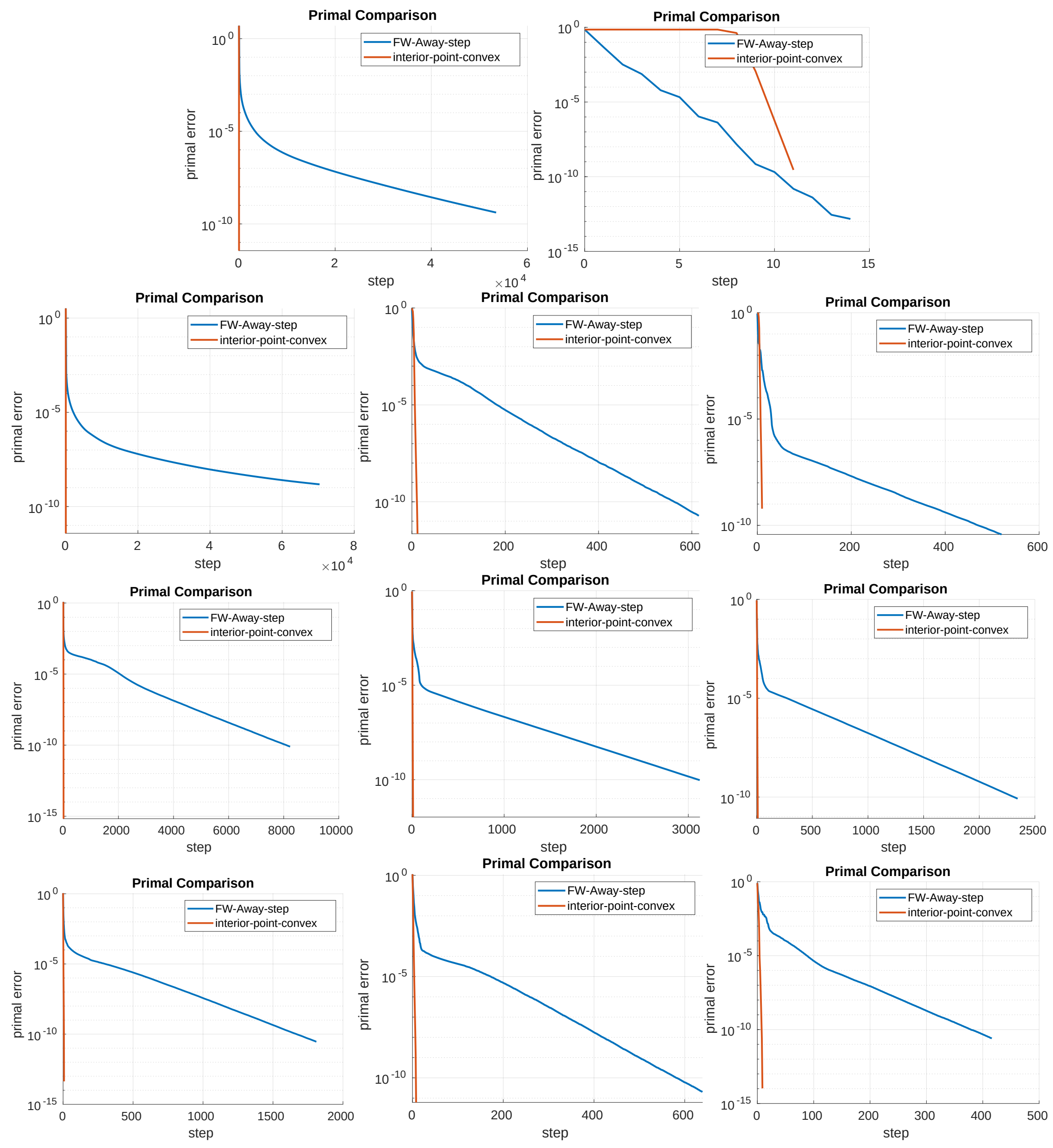


Figure 11: The Figure shows, in "left-to-right then top-to-bottom" order, the plots of the error curves of the experiments listed in Table 9.

A.4.3 Large size dimension

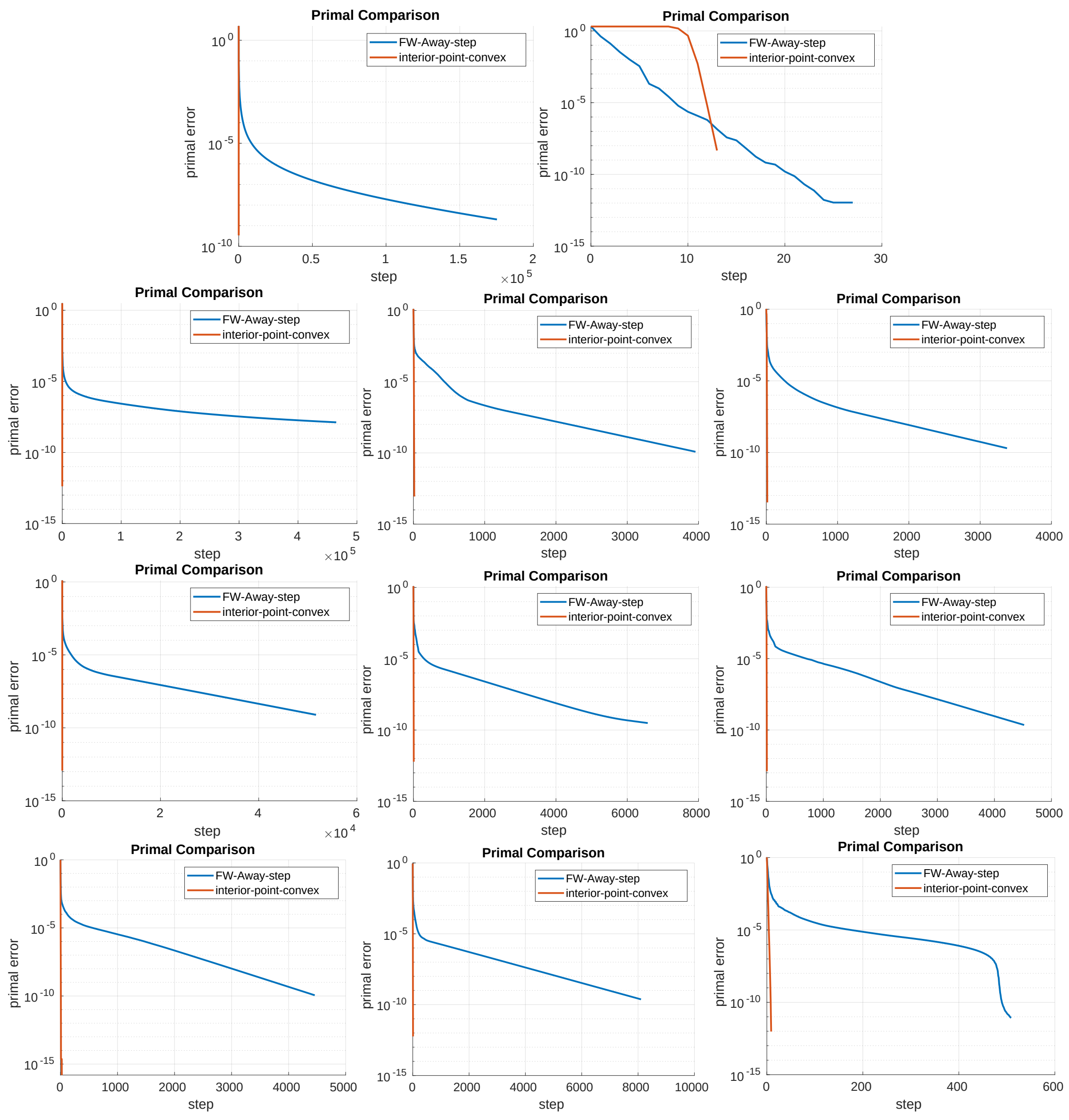


Figure 12: The Figure shows, in "left-to-right then top-to-bottom" order, the plots of the error curves of the experiments listed in Table 10.