Project Non-ML 5
Final Project for the CM course 2020/21

Matteo De Francesco

Contents

1 Introduction

In this report we will analize the convex quadratic problem

$$\min \left\{ x^T Q x + q^T x : \sum_{i \in I^k} x_i = 1, k \in K, x \geq 0 \right\} \tag{P}$$

with the following constraints: $x \in \mathbb{R}^n$, the index sets I^k form a partition of the set $\{1, \ldots, n\}$ (i.e.$\cup_{k \in K} I^k = \{1, \ldots, n\}$, and $I^h \cap I^k = \emptyset$ for all h and k), and Q is positive semidefinite.
The aim of this project is to exploit the Lagrangian dual problem and solve it via one of the *deflected subgradient* methods.
We can identify the inequality and equality constraints and rewrite them in the typical form

$$G(x) \to g_i(x) \leq 0 \implies -x_i \leq 0 \; \forall i \in \{1, \ldots, n\}$$
$$H(x) \to h_j(x) = 0 \implies \sum_{i \in I^k} x_i - 1 = 0 \; \forall k \in K$$

Hence we can rewrite the problem in the following form:

$$\begin{cases} \min \; x^T Q x + q^T x \\ -x_i \leq 0 \; \forall \, i \in \{1, \ldots, n\} \\ \sum_{i \in I^k} x_i - 1 = 0 \; \forall \, k \in K \end{cases}$$

In order to have less dual variables, we can rewrite the above problem as a lagrangian function of only the inequality constraints:

$$\mathcal{L}(x; \lambda) = x^T Q x + q^T x - \langle \lambda, x \rangle$$

Now, we can easily construct the lagrangian dual function considering the equality constraints

$$\psi(\lambda) = \min_{x \in \mathcal{Y}} \{ \mathcal{L}(x; \lambda), \; \lambda \geq 0 \}$$

where $\mathcal{Y} = \{ \sum_{i \in I^k} x_i = 1 \; \forall \, k \in K \}$. Hence the following optimization problem

$$\begin{cases} \max_\lambda \; \psi(\lambda) \\ \text{subject to } \lambda \geq 0 \end{cases} \tag{D}$$

We are assuming that optimizing over the set $\mathcal{Y}$ can be done very easily.
In the next section, we will briefly recall the properties of the `ADAGRAD` family of algorithms[1].

2 Deflected subgradient Analysis

2.1 `ADAGRAD` introduction

The projection rule of a point y onto the constraint set $\mathcal{X}$ according to a positive semidefinite matrix A amounts to:

$$\prod_{\mathbb{X}}^{A}(y) = \arg\min_{\lambda \in X} \|\lambda - y\|_A = \arg\min_{\lambda \in X} \langle \lambda - y, A(\lambda - y) \rangle$$

and it's aimed to minimize the Mahalanobis norm.
`ADAGRAD` applies the following projection rule:

$$\lambda_{t+1} = \prod_{\mathcal{X}}^{diag(G_t)^{1/2}} (\lambda_t - \eta \, diag(G_t)^{-1/2} g_t) \tag{1}$$

where the matrix A coincides with the diagonal square root of the full outer product of the subgradients $G_t = \sum_{\tau=1}^{t} g_\tau g_\tau^T$. As we stated in the introduction, we should exploit `ADAGRAD`

on the dual problem, whose constraint set consists in the simple inequality constraint $\lambda \geq 0$, which is addressable by `ADAGRAD`, given the fact that the algorithm can be applied to any convex set $\mathcal{X} \subseteq \mathbb{R}^n$, respected by our problem (we are in $\mathbb{R}^n_+$).
In addition, as the general projection rule tell us, the new update λ_{t+1} is projected over the constraint set $\mathcal{X}$ according to the matrix of the gradients.
We will recall in the next section some algorithmic properties of the algorithm convergence for diagonal matrices, and we will derive the update rule followed by the projection over the constraint set $\mathcal{X} = \{\lambda \geq 0\}$.

2.2 Algorithm characteristics

First of all, we reiterate that `ADAGRAD` is suitable for the dual problem, given its application in a convex setting. The goal is to attain a small regret bound:

$$R_\phi(T) \triangleq \sum_{t=1}^{T} \psi(\lambda_t) - \psi(\lambda^*)$$

between the actual value of the dual function ψ with the corresponding iterate λ at step t and the value of ψ with the optimal solution λ^*.
The point of `ADAGRAD` is that not all features are equal, hence they must be treated differently. That's the purpose of using an adaption to the geometry of space, so do not use anymore a standard gradient descent but conditioning the different values based on a positive semidefinite matrix A (the G_t in this case).
Two algorithm versions are analyzed in [1], where we omitted the regularization term due to our problem setting. The first update, referred to as *primal-dual subgradient method*, is

$$\lambda_{t+1} = \arg\min_{\lambda \in \mathcal{X}} \{\eta \langle \overline{g}_t, \lambda \rangle + \frac{1}{t} \Psi_t(\lambda)\} \tag{2}$$

coming from [2], where $\overline{g}_t = \frac{1}{t} \sum_{\tau=1}^{t} g_\tau$ is the average gradient, η is a fixed stepsize and Ψ_t is the *proximal term*.
The second update instead

$$\lambda_{t+1} = \arg\min_{\lambda \in \mathcal{X}} \{\eta \langle g_t, \lambda \rangle + B_{\Psi_t}(\lambda, \lambda_t)\} \tag{3}$$

where B refers to the Bregman divergence. This second update comes from [3].
Finally, also the previous mentioned projection rule can be used as an update:

$$\lambda_{t+1} = P_{\mathcal{X}}\{\lambda_t + \eta \, diag(G_t)^{-1/2} g_t\} \tag{4}$$

where P denotes the projection operation. The proximal term Ψ_t is the key point of the algorithm. Instead of using a fixed proximal functions, both the updates use squared Mahalanobis norms as their proximal functions, setting then $\Psi_t(\lambda) = \langle \lambda, H_t \lambda \rangle$ for a symmetric matrix $H_t \succeq 0$. In particular, the diagonal case which we will recall here make use of:

$$H_t = \delta I + diag(G_t)^{1/2}$$

for some smalled fixed $\delta \geq 0$.
The usage of a strongly convex proximal function is also remarked in [4] in appendix A.

2.3 Proximal term discussion

In order to attain a lower regret bound and to adapt to the geometry of the space, the objective of the authors is to not use anymore a standard proximal functions but a modified version of it.
The proximal function act as a regularization term, typically the Euclidian projection over a convex set. Given the indicator function of a convex set C:

$$I_C(\lambda) = \begin{cases} 0 & \lambda \in C \\ +\infty & \text{otherwise} \end{cases}$$

the standard proximal mapping of I_C is the Euclidean projection on C:

$$prox_{I_C}(\lambda) = \arg\min_{u \in C} \|u - \lambda\|_2^2 = P_C(\lambda)$$

Instead in this case the authors noticed that some local region of the function to be optimized need more "attention than others". What they come up with then is the modification (also remarked in the `ADAGRAD` introduction) using a different proximal function where

$$prox_{I_C}(\lambda) = \arg\min_{u \in C} \|u - \lambda\|_A^2 = P_C(\lambda)$$

the Euclidean projection is not computed anymore according to a 2-norm, but according to a matrix A which in the case of `ADAGRAD` coincide with the matrix of the subgradients. A summary of the properties and aspects of some proximal functions can be found in [5]. Below is reported the analysis of the proximal function regarding [1].
Examining the regret bounds for the updates in [6] and [2], it is quite obvious that they depends on dual norms of the derivative of the function to be optimized, and in turn this last depend on the choice of Ψ. The objective of [1] is to properly modify the value of Ψ during the run of the algorithm in order to lower the contribution of the norms, and so lower the regret bound. This is achieved by keeping second order information about the sequence of iterates λ_t and allow Ψ to vary on each round of the algorithm. To achieve this, we must assume that Ψ_t is monotically non-decreasing, 1-strongly convex with respect to a time-dependent semi-norm $\|\cdot\|_{\Psi_t}$. Formally, given two generic points x and y, Ψ is 1-strongly convex with respect to $\|\cdot\|_\Psi$ if

$$\Psi(y) \geq \Psi(x) + \langle \nabla\Psi(x), y - x \rangle + \frac{1}{2}\|x - y\|_\Psi^2$$

As a consequence, strong convexity is guaranteed if and only of $B_{\Psi_t}(x, y) \geq \frac{1}{2}\|x - y\|_{\Psi_t}^2$. The following bound holds then on proximal term, respectively for (2) and (3). Proofs can be found in Appendix F of [1].

Proposition 2.1. *Let the sequence $\{\lambda_t\}$ be defined by the update (2). For any $\lambda^* \in \mathcal{X}$*

$$\sum_{t=1}^{T} \psi(\lambda_t) - \psi(\lambda^*) \leq \frac{1}{\eta}\Psi_T(\lambda^*) + \frac{\eta}{2}\sum_{t=1}^{T} \|\psi'(\lambda_t)\|_{\Psi_{t-1}^*}^2$$

Proposition 2.2. *Let the sequence $\{\lambda_t\}$ be defined by the update (3). For any $\lambda^* \in \mathcal{X}$*

$$\sum_{t=1}^{T} \psi(\lambda_t) - \psi(\lambda^*) \leq \frac{1}{\eta}B_{\Psi_1}(\lambda^*, \lambda_1) + \frac{1}{\eta}\sum_{t=1}^{T-1}\left[B_{\Psi_{t+1}}(\lambda^*, \lambda_{t+1}) - B_{\Psi_t}(\lambda^*, \lambda_{t+1})\right] + \frac{\eta}{2}\sum_{t=1}^{T} \|\psi'(\lambda_t)\|_{\Psi_t^*}^2$$

The proximal term and the bregman divergence present in (2) and (3) will be computed according to the subgradient matrix, in order to iteratively modify the update and thus adapting to the geometry of the space.

2.4 Convergence Analysis

First of all, let us state that the convergence rate of `ADAGRAD` is the same of the Stochastic Gradient Descent (`SGD`), hence $O(1/\sqrt{T})$ but with a lower constant due to the use of G matrix.
The algorithm 1 reported in [1] will be applied to our problem and is reported here:

Algorithm 1 `ADAGRAD` for diagonal matrices

function `ADAGRAD`(η,δ)
  $x_1 \leftarrow 0$
  $\lambda_1 \leftarrow 0$
  $g_{1:0} \leftarrow [\,]$
  for $t \leftarrow 1$ to T **do**
    $Loss = f_t(x_t)$
    $g_t \leftarrow \partial\psi(\lambda_{t-1})$ of ψ at λ_{t-1} ▷ Compute subgradient at λ_{t-1}
    $g_{1:t} \leftarrow [g_{1:t-1}\ g_t]$ ▷ Store subgradient
    $s_{t,i} \leftarrow \|g_{1:t,i}\|_2$ ▷ Compute the optimal $s_{t,i}$
    $H_t \leftarrow \delta I + diag(s_t)$
    $\Psi_t(\lambda) \leftarrow \frac{1}{2}\langle \lambda, H_t\lambda\rangle$
    Either compute (2) or (3)
  end for
end function

The general convergence result of this algorithm for both the updates is reported in theorem 5 of the original paper. We will now report the theoretical analysis behind procedure 1 and lastly modify the algorithm in order to match our problem settings.

Theorem 2.1. *Let the sequence $\{\lambda_t\}$ be defined by algorithm 1. For λ_t generated using the update* (2) *with $\delta \geq \max_t \|g_t\|_\infty$, for any $\lambda^* \in \mathcal{X}$*

$$R_\phi(T) \leq \underbrace{\frac{\delta}{\eta}\|\lambda^*\|_2^2 + \frac{1}{\eta}\|\lambda^*\|_\infty^2 \sum_{i=1}^{d}\|g_{1:T,i}\|_2}_{①} + \underbrace{\eta\sum_{i=1}^{d}\|g_{1:T,i}\|_2}_{②} \tag{5}$$

For λ_t generated using the update (3)*, for any $\lambda^* \in \mathcal{X}$*

$$R_\phi(T) \leq \underbrace{\frac{1}{2\eta}\max_{t\leq T}\|\lambda^* - \lambda_t\|_\infty^2 \sum_{i=1}^{d}\|g_{1:T,i}\|_2}_{③} + \underbrace{\eta\sum_{i=1}^{d}\|g_{1:T,i}\|_2}_{②} \tag{6}$$

We will now delve into analyzing the different $\textcircled{\cdot}$ parts.
First of all, we should recall some intuition in order to understand better the regret bound provided above.
Focusin on the algorithm 1, the chosen value of $s_{t,i}$ explain us why the particular choice of the proximal function $\Psi_t(x) = \frac{1}{2}\langle x, H_t x\rangle$ is so important. $s_{t,i}$ comes from the solution of the problem

$$\min_s \sum_{t=1}^{T}\sum_{i=1}^{d}\frac{g_{t,i}^2}{s_i} \text{ s.t. } s \succeq 0, \langle 1, s\rangle \leq c$$

solved by optimizing the Lagrangian

$$\mathcal{L}(s,\lambda,\theta) = \sum_{i=1}^{d}\frac{\|g_{1:T,i}\|_2^2}{s_i} - \langle\lambda, s\rangle + \theta(\langle 1, s\rangle - c)$$

Taking the partial derivatives to find the infimum of $\mathcal{L}$, and using the complementary slackness condition on $\lambda_i s_i$ imply that $\lambda_i = 0$. As a consequence, we obtain $s_i = c\|g_{1:T,i}\|_2 / \sum_{j=1}^{d}\|g_{1:T,j}\|_2$. Plugging this into the previous objective function, we get

$$\inf_s\left\{\sum_{t=1}^{T}\sum_{i=1}^{d}\frac{g_{t,i}^2}{s_i} : s \succeq 0, \langle 1, s\rangle \leq c\right\} = \frac{1}{c}\left(\sum_{i=1}^{d}\|g_{1:T,i}\|_2\right)^2$$

Now it is natural to suspect that for s achieving the infimum in this latter equation, using a proximal function similar to $\Psi(\lambda) = \langle\lambda, diag(s)\lambda\rangle$ with associated squared dual norm

$\|\lambda\|^2_{\Psi^*} = \langle \lambda, diag(s)^{-1}\lambda \rangle$ we should lower the gradient terms both in (5) and (6).
The upper bound on the gradient term for both the updates is taken from the `LEMMA 4` of [1], stating:

Lemma 2.1. *Let $g_t = \psi'(\lambda_t)$ and $g_{1:t}$ and s_t be defined as in algorithm 1. Then*

$$\sum_{t=1}^{T} \langle g_t, diag(s_t)^{-1} g_t \rangle \leq 2 \sum_{i=1}^{d} \|g_{1:T,i}\|_2$$

To obtain a bound, we need to consider the terms consisting of the dual-norm of the subgradient in the bounds (5) and (6), which is $\|\psi'(\lambda_t)\|^2_{\Psi_t^*}$. When we choose $\Psi_t(\lambda) = \langle \lambda, (\delta I + diag(s_t))\lambda \rangle$, the associated dual norm is

$$\|g\|^2_{\Psi_t^*} = \langle g, (\delta I + diag(s_t))^{-1} g \rangle$$

Following from the definition of s_t in 1, we have $\|\psi'(\lambda_t)\|^2_{\Psi_t^*} \leq \langle g_t, diag(s_t)^{-1} g_t \rangle$. Thus we have the following implication

$$\sum_{t=1}^{T} \|\psi'(\lambda_t)\|^2_{\Psi_t^*} \leq \sum_{i=1}^{d} \|g_{1:T,i}\|_2$$

which prove the regret term (2).
Consequently, it remains to prove the bound over the Bregman divergence and the term $\Psi_T(\lambda^*)$ of proposition 2.1 and 2.2. Focusing on the composite mirror descent, we have:

$$\begin{aligned} B_{\Psi_{t+1}}(\lambda^*, \lambda_{t+1}) - B_{\Psi_t}(\lambda^*, \lambda_{t+1}) &= \frac{1}{2} \langle \lambda^* - \lambda_{t+1}, diag(s_{t+1} - s_t)(\lambda^* - \lambda_{t+1}) \rangle \\ &\leq \frac{1}{2} \max_i (\lambda_i^* - \lambda_{t+1,i})^2 \|s_{t+1} - s_t\|_1 \end{aligned}$$

Since $\|s_{t+1} - s_t\|_1 = \langle s_{t+1} - s_t, 1 \rangle$ and $\langle s_T, 1 \rangle = \sum_{i=1}^{d} \|g_{1:T,i}\|_2$ we have

$$\begin{aligned} \sum_{t=1}^{T-1} B_{\Psi_{t+1}}(\lambda^*, \lambda_{t+1}) - B_{\Psi_t}(\lambda^*, \lambda_{t+1}) &\leq \frac{1}{2} \sum_{t=1}^{T-1} \|\lambda^* - \lambda_{t+1}\|^2_\infty \langle s_{t+1} - s_t, 1 \rangle \\ &\leq \frac{1}{2} \max_{t \leq T} \|\lambda^* - \lambda_t\|^2_\infty \sum_{i=1}^{d} \|g_{1:T,i}\|_2 - \frac{1}{2} \|\lambda^* - \lambda_1\|^2_\infty \langle s_1, 1 \rangle \end{aligned}$$

which prove us the term (3).
Finally, we also have that

$$\Psi_T(\lambda^*) = \delta \|\lambda^*\|_2^2 + \langle \lambda^*, diag(s_T)\lambda^* \rangle \leq \delta \|\lambda^*\|_2^2 + \|\lambda^*\|^2_\infty \sum_{i=1}^{d} \|g_{1:T,i}\|_2$$

which give us the bound (1).
Performing a few algebraic simplification lead us to a corollary which give us a more intuitive form of the regret bound. Assume that $\mathcal{X}$ is a compact set and set $D_\infty = \sup_{\lambda \in \mathcal{X}} \|\lambda - \lambda^*\|_\infty$, and define

$$\gamma_T \triangleq \sum_{i=1}^{d} \|g_{1:T,i}\|_2 = \inf_s \left\{ \sum_{t=1}^{T} \langle g_t, diag(s)^{-1} g_t \rangle : \langle 1, s \rangle \leq \sum_{i=1}^{d} \|g_{1:T,i}\|_2, s \succeq 0 \right\}$$

Corollary 2.1. *Assume that D_∞ and γ_T are defined as above. For $\{\lambda_t\}$ generated using the (2) with $\eta = \|\lambda^*\|_\infty$, for any $\lambda^* \in \mathcal{X}$ we have*

$$R_\phi(T) \leq 2\|\lambda^*\|_\infty \gamma_T + \delta \frac{\|\lambda^*\|_2^2}{\|\lambda^*\|_\infty} \leq 2\|\lambda^*\|_\infty \gamma_T + \delta \|\lambda^*\|_1$$

Using update (3) *to generate* $\{\lambda_t\}$ *and setting* $\eta = D_\infty/\sqrt{2}$*, we have*

$$R_\phi(T) \leq \sqrt{2} D_\infty \sum_{i=1}^{d} \|g_{1:T,i}\|_2 = \sqrt{2} D_\infty \gamma_T$$

All these presented results are respected by our problem. Indeed, as we stated before, our original problem is for sure a convex problem (a quadratic function with positive semidefinite Q) and also the constraints are convex (a set of disjoint unitary simplices) implying strong duality. As we know from theory, the dual problem is for sure convex, even if the original one it's not. What we should argue is the presence of the constraints on the dual variables. Being `ADAGRAD` applicable to any convex set $\mathcal{X} \subseteq \mathbb{R}^n$, this is suitable for our problem, since $\mathcal{X} = \{\lambda \geq 0\}$. We will use one of the just explained update, (preferably the *primal-dual* one, since we have a general proof of convergence over exactly $\mathbb{R}^n_+$, in `LEMMA 2` of Appendix A of [4]), compute the update and then project it on the λ's constraint set.
We expect to achieve an asymptotically sub-linear regret, as the authors showed in their work.

2.5 General application to our case

In our setting, we aim to solve problem (P) using (D). We will use $\mathcal{X} = \{\lambda \geq 0\}$ as the constraint set of the lagrangian multipliers λ's, and $\mathcal{Y} = \{\sum_{i \in I^k} x_i = 1 \ \forall k \in K\}$ as the constraint set of the primal variables.
To solve the problem (2) and (3) in our problem settings, we consider the update of the dual variable λ. Referring to the problem (D), we can freely choose among three different update rules, in order (1), (2) and (3):

$$\begin{aligned} \lambda_{t+1} &= P_{\mathcal{X}}\{\lambda_t + \eta \, diag(G_t)^{-1/2} g_t\} \\ \lambda_{t+1} &= \arg\max_{\lambda \in \mathcal{X}} \{\eta \langle \overline{g}_t, \lambda \rangle + \frac{1}{t} \Psi_t(\lambda)\} \\ \lambda_{t+1} &= \arg\max_{\lambda \in \mathcal{X}} \{\eta \langle g_t, \lambda \rangle + B_{\Psi_t}(\lambda, \lambda_t)\} \end{aligned} \tag{7}$$

We need to find a maximum for the considered update function and then project it onto the constraint set $\mathcal{X}$.
About the terms Ψ and B_Ψ, these will be replaced, according to what we have said before, respectively by:

$$\begin{aligned} \Psi_t(\lambda) &= \frac{1}{2} \langle \lambda, H_t \lambda \rangle \\ B_{\Psi_t}(\lambda, \lambda_t) &= \Psi_t(\lambda) - \Psi_t(\lambda_t) - \langle \nabla \Psi_t(\lambda_t), \lambda - \lambda_t \rangle \end{aligned}$$

Regarding the maximum, this can be simply solved by differentiating the function with respect to the variable to be maximized, and setting it equal to 0. The detailed derivation of each update can be found in the appendix A.

$$\hat{\lambda}_{t+1} = \lambda_t + \eta diag(G_t)^{-1/2} g_t \qquad \text{(Update (4))}$$

$$\hat{\lambda}_{t+1} = -H_t^{-1} t \, \eta \, \overline{g}_t \qquad \text{(Update (2))}$$

$$\hat{\lambda}_{t+1} = \lambda_t + \eta \, H_t^{-1} \, g_t \qquad \text{(Update (3))}$$

For the stepsize η, this can be fixed apriori and we will see being it crucial for the convergence of the algorithm. Different stepsize rules can be employed [7]

$$\eta = h \qquad \text{Constant step size rule, with } h > 0 \tag{9}$$

$$\eta = \frac{h}{\|g^t\|} \qquad \text{Constant step length} \tag{10}$$

$$\eta = \frac{\alpha}{\beta + t} \qquad \text{Square summable but not summable, with } \alpha > 0 \text{ and } \beta \geq 0 \tag{11}$$

$$\eta = \frac{\alpha}{\sqrt{t}} \qquad \text{Nonsummable diminishing} \tag{12}$$

$$\eta = \frac{f(x^*) - \phi(\lambda_t)}{\|g^t\|^2} \qquad \text{Polyak stepsize} \tag{13}$$

For the dual variables, after we obtain $\hat{\lambda}_{t+1}$, we use the projection over the nonnegative orthant to get λ_{t+1}, formally

$$\lambda_{t+1} = P_{\mathcal{X}}(\hat{\lambda}_{t+1}) = \max\{0, \hat{\lambda}_{t+1}\}$$

which is a trivial problem tackled many times in the literature [8].
We need also to derive an update for the primal variables, which are needed at each iteration in order to compute the dual function $\psi(\lambda)$ and the subgradient $\nabla_\lambda \psi(\lambda)$. Given the previous multiplier value λ_t

$$x_{t+1} = \arg\min_{x \in \mathcal{Y}} \{x^T Q x + q^T x - \langle \lambda_t, x \rangle\}$$

We need to solve the lagrangian relaxation of a convex quadratic problem, depending on equality constraints made up of disjoint simplices. The constraint set $\mathcal{Y}$ can be rewritten in a matrix form $Ax = b$, where the vector b is a $k \times 1$ vector of all ones, and the matrix A can be derived with the following simple algorithm:

Algorithm 2 Construct matrix A

```
procedure CONSTRUCT_A(K, [I^k])
    A ← [ ]                                  ▷ Initialize k × n empty matrix
    for k ← 1 to K do
        a_k ← zeros(n, 1)                    ▷ n × 1 empty row vector
        for i ← 1 to n do
            if i ∈ I^k then                  ▷ Check if index i is in the set I^k
                a_k[i] ← 1
            end if
        end for
        A[k, :] ← a_k
    end for
end procedure
```

Using the **KKT** (Karush-Kuhn Tucker) conditions, we can solve the problem directly through linear algebra by constructing the following linear system:

$$\begin{cases} Qx + q - \lambda_t + \mu A = 0 \\ Ax - b = 0 \end{cases} \implies \begin{cases} Qx + \mu A = \lambda_t - q \\ Ax \qquad\quad = b \end{cases} \implies$$

$$\begin{bmatrix} Q & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x \\ \mu \end{bmatrix} = \begin{bmatrix} \lambda_t - q \\ b \end{bmatrix}$$

a linear system that can be solved directly and efficiently by computing the pivoted LU factorization of the matrix and save it for the next iterations. Hence, using the LU factor-

ization:

$$\begin{bmatrix} Q & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x \\ \mu \end{bmatrix} = \begin{bmatrix} \lambda_t - q \\ b \end{bmatrix} \implies \begin{bmatrix} x \\ \mu \end{bmatrix} = \begin{bmatrix} Q & A^T \\ A & 0 \end{bmatrix}^{-1} \begin{bmatrix} \lambda_t - q \\ b \end{bmatrix} \implies$$

$$\begin{bmatrix} x \\ \mu \end{bmatrix} = U^{-1} * \left(L^{-1} * \left(P * \begin{bmatrix} \lambda_t - q \\ b \end{bmatrix} \right) \right)$$

which can be solved efficiently through forward substitution (for the matrix L) and back substitution (for the matrix U) with a complexity of $O((n+K)^2)$.
Lastly, regarding the stopping condition for the algorithm termination, we can check the dual variables residual. Follows that, given a certain value of tolerance ε, we have

$$\|\lambda_t - \lambda_{t-1}\|_2 \leq \varepsilon \tag{14}$$

In addition to this, what we can check too is the duality gap between the original function and the dual function (provided that we know the optimal value), so

$$f(x) - \psi(\lambda)$$
$$f(x) - f(x^*) \leq f(x) - \psi(\lambda)$$

From this we understand that having a zero duality gap implies optimality of the solution, and so fixing a certain value of tolerance τ we have a second stopping condition

$$f(x^*) - \psi(\lambda_i) \leq \tau \tag{15}$$

for each λ_i computed during iterations.
The starting value of the λ's iterates is a vector of ones and the starting x is computed immediately using the starting λ and the factorization explained before.
Putting everything together, we obtain a slightly different algorithm than 1:

Algorithm 3 `ADAGRAD` on our dual problem

function `ADAGRAD`(η, δ, ε, τ, max_iter)
  $\lambda_0 \leftarrow 1$
  $g_{1:0} \leftarrow [\,]$
  $x_0 \leftarrow$ system solution
  $\psi = \psi(\lambda_0)$
  for $t \leftarrow 1$ to max_iter **do**
    $g_t \leftarrow \dfrac{\partial \psi_\lambda(\lambda)}{\partial \lambda}$ ▷ Compute subgradient B
    $g_{1:t} \leftarrow [g_{1:t-1}\ g_t]$ ▷ Store subgradient
    $s_{t,i} \leftarrow \|g_{1:t,i}\|_2$ ▷ Solution of the problem 2.4
    $H_t =\leftarrow \delta I + diag(s_t)$
    $x_t = \arg\min_{x \in \mathcal{Y}} \left\{ x^T Q x + q^T x - \langle \lambda_{t-1}, x \rangle \right\}$ ▷ Lagrangian
    $\hat{\lambda}_t =$ one among (7)
    $\lambda_t = P_{\mathcal{X}}(\hat{\lambda}_t)$
    if $\psi(\lambda_t) \geq \psi$ **then**
      $\psi = \psi(\lambda_t)$
    end if
    if (14) `OR` (15) is *true* **then**
      return ψ
    end if
  end for
end function

3 Implementation

The ideas above described have been implemented using `Julia`. The provided package give us the ability to provide the dimension of the problem n and the number of simplices K.

Subsequently, the code is all automated and provide the creation of the matrix $Q \succeq 0$, A, the random sets I^k, the vector q and all the required parameters. The update rules to test can be chosen as well as the different stepsize rules that you want to employ.
The structures created are stored in a *.mat* file and loaded each time to avoid time consumption in the creation of the different objects and also for the derivation of the optimal primal solution. This latter is found using the off-the-shelf solver function *quadprog()*, available through `MATLAB` with the optional *Optimization toolbox*. The generated *.mat* file is stored and a simple script is used to derive the primal solution.
Also, there are some hyperparameters which can be modified by the user inside *main.jl*, like the maximum number of iterations and the ε value to check the λ residual.
All the values assumed by each λ-norm and x-norm residual, the number of iterations, the value of the dual function and the gap at the current iteration are stored in a *.csv* file. The last one is used then to generate some plots showing the convergence towards the optimal solution.
The customized `ADAGRAD` display at every iteration the time elapsed, the ψ value, both the λ norm residual and the x one, and the dual gap $f(x^*) - \psi(\lambda)$.
In order to test the program on different problem dimensions, we have chosen different sizes to perform our experiments, in order to have a "grid search" of different experimentations:

n	**K**	$\boldsymbol{\varepsilon}$	$\boldsymbol{\tau}$	$\boldsymbol{\delta}$	**max_iter**
1000	$[20, 100, 500]$	10^{-14}	10^{-7}	10^{-16}	$[5000, 7000, 10000]$
5000	$[10, 1000, 2500]$		10^{-6}		
10000	$[10, 2500]$		10^{-6}		

The code was executed using `Julia REPL` tool. It has a slowish startup time, so the first execution are "warm-up" execution, but as soon as the entire code is compiled, all modules are loaded and the needed memory is computed. Q is the resulting covariance matrix of a random generated matrix A_h, in formula:

$$Q = A_h^T * A_h$$

which we know being symmetric and surely positive semidefinite.
Regarding the factorization, the entire `KKT` matrix is factorized using the `Julia` method `lu!()`.
Another observation regard the subgradient: being the ψ non differentiable in the general case, the direction is not guaranteed to be ascent/descent. So we keep track of the best $\psi(\lambda)$, best iteration, best x and λ to get the suboptimal solution found. This sometimes result in the dual function being minimized instead of maximized. In certain cases we observed also that when we reach a negative dual gap (due to big steps and not feasibility of the actual iterates) changing the direction of the subgradient brings improvement in reaching the solution.
Last but not least, we noticed in the experiments that the stepsize selection is crucial for the convergence of the algorithm. In particular, the value of the hyperparameters α, β and δ should be chosen ad-hoc to guarantee that we have a "smooth" optimization. The mentioned hyperparameters are fundamental, the latter for the update rules (2) and (3) while the first two for the standard (4).
In the first two mentioned updates, the value of H_t^{-1} acts as a sort of "regularizer", scaling in both cases the value of the gradient. Modifying δ impacts primarly on the steps taken, with smaller value of it resulting in bigger steps while a low value of it result in a very low convergence (even too much).
Referring always to these two updates, the only stepsize rule that give a "fast" convergence with big steps is the first one (9), with the variable h. We change the value of h during iteration making it smaller as soon as we reach the optimal gap.
The rule (2) shows no improvement even adjusting these parameters, while rule (3) converges slowly with respect to the standard one.
Indeed it is rule (4) that give us the best results: we are capable of reaching the desired dual gap in fewer iterations, adjusting the value of the hyperparameters, switching between the different stepsize rules at our disposal.

We can report now the convergence plots of some cases highlighted in table 3, together with the best result obtained, the time required by the off-the-shelf solver, a summing up table showing the comparison with other hyperparameters' values and a comment to better understand what we obtained.
Also, we will take a look at how the solver scales changing the value of K by means of timing, dimensions and iterations.

4 Results

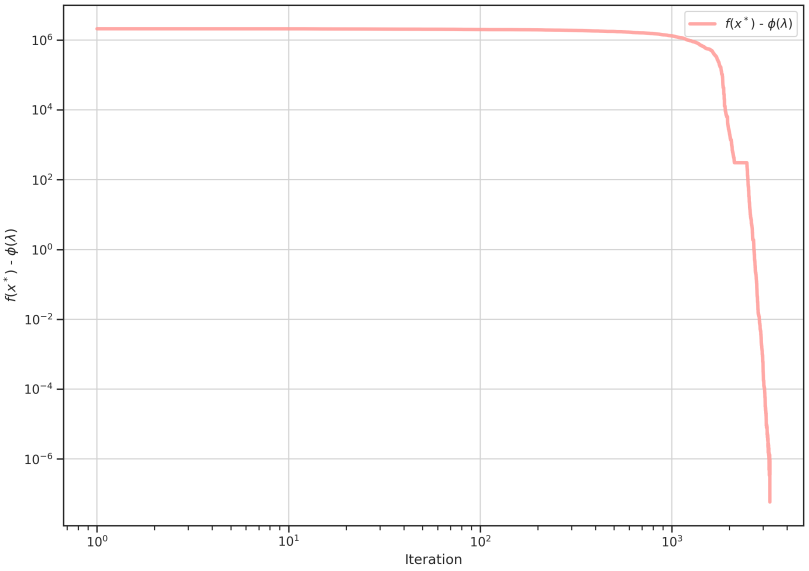


Figure 1: Dual gap with update rule 1, $n = 1000$ and $K = 100$

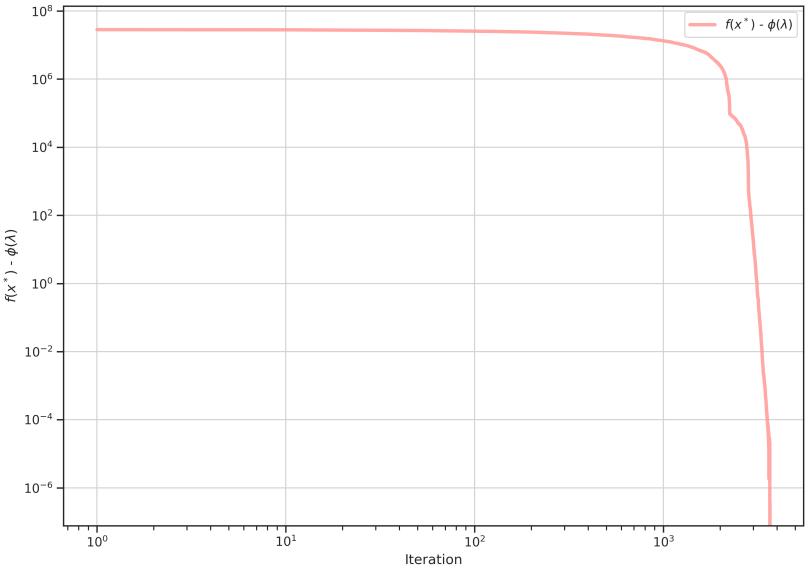
Stepsize used	(12)(13)
Optimal α *found*	0.8
Total iterations	3247
total time (sec)	170.4 *s*
Best $f(x^*) - \psi(\lambda)$	5.867*e*−08
Best iteration	3247
Best $\psi(\lambda)$	2.365*e*+06
Best $\|\lambda_t - \lambda_{t-1}\|$	2.384*e*−12
Best $\|x_t - x_{t-1}\|$	2.839*e*−11
`quadprog` time	1.2 *s*
`quadprog` iterations	17

Table 1: Sum up table for $n = 1000$, $K = 100$ and rule 1

	$\alpha = 0.1$	$\alpha = 5$	$\alpha = 10$
Stepsize used	(11)	(11)	(11)
Total iterations	5000	5000	5000
total time (sec)	361 *s*	312.7 *s*	298.4 *s*
Best $f(x^*) - \psi(\lambda)$	2.11*e*+06	1.60*e*+06	4.068*e*+00
Best iteration	5000	5000	1872
Best $\psi(\lambda)$	2.55*e*+05	7.62*e*+05	2.365*e*+06
Best $\|\lambda_t - \lambda_{t-1}\|$	6.291*e*−04	3.157*e*−02	0.309*e*+00
Best $\|x_t - x_{t-1}\|$	1.758*e*−10	1.139*e*−01	8.180*e*−11
`quadprog` time	1.2 *s*	1.2 *s*	1.2 *s*
`quadprog` iterations	17	17	17

Table 2: Sum up table for $n = 1000$, $K = 100$ and rule 1

In this first case we find the optimal $\alpha = 0.8$ using different rules to have a faster and smooth convergence. We compared this best result with the standard DSS rule and different values of the hyperparameter α.
Looking at the other tries, we notice how using $\alpha = 10$, we found a gap in the order $O(10^0)$, which is not the desired one but near to the optimal. In particular this is reached at the iteration 1872, despite we reach the maximum 5000 iterations. Indeed in this case the value of α is quite big, and so the value of dual function inevitably jumps around the optimal, with values $\gtrless f^*$.



Stepsize used	(12)(11)(13)
Optimal α *found*	10
Total iterations	3665
total time (sec)	76.9 s
Best $f(x^*) - \psi(\lambda)$	3.799e−07
Best iteration	3665
Best $\psi(\lambda)$	6.158e+07
Best $\|\lambda_t - \lambda_{t-1}\|$	3.551e−15
Best $\|x_t - x_{t-1}\|$	7.712e−09
`quadprog` time	0.9 s
`quadprog` iterations	17

Table 3: Sum up table for $n = 1000$, $K = 500$ and rule 1

Figure 2: Dual gap with update rule 1, $n = 1000$ and $K = 500$

As we can see, the time scaling is consistent, while iterations stays the same. When we add more constraints, the problem gets easier, since fixing the value of x is more easier given that the dimension of the sets I^k is smaller (on average). Indeed, in the same order of iterations ($O(3000)$), the time is halved for our solver.
Instead for the iterations, we can notice how this increase. This is due to the more timing required to solve the `KKT` system and also to the more iterations needed, given that the problem objective increase.
Of course everything is also dependent on the stepsize rules used. In this case to get a faster and smooth convergence we used three different stepsize rules. Let's look now at a table summing up the comparison with other value of the hyperparameter α and the stepsize rule (11):

	$\alpha = 5$
Stepsize used	(11)
Total iterations	10000
total time (sec)	753.8 s
Best $f(x^*) - \psi(\lambda)$	5.6e+05
Best iteration	10000
Best $\psi(\lambda)$	6.1e+07
Best $\|\lambda_t - \lambda_{t-1}\|$	1.726e−00
Best $\|x_t - x_{t-1}\|$	0.151e−00
`quadprog` time	0.9 s
`quadprog` iterations	17

Table 4: Sum up table for $n = 1000$, $K = 500$ and rule 1

Using a different α and the standard DSS rule in this case, we still stay very far from the optimal gap. That's why we combined the different step rules, to reach the optimal gap in a faster way and always in a smooth way. Using the standard value and fixed value of α (like in the last table) we know from theory that convergence is still guaranteed but in particular with the subgradient method is slower than standard gradient.

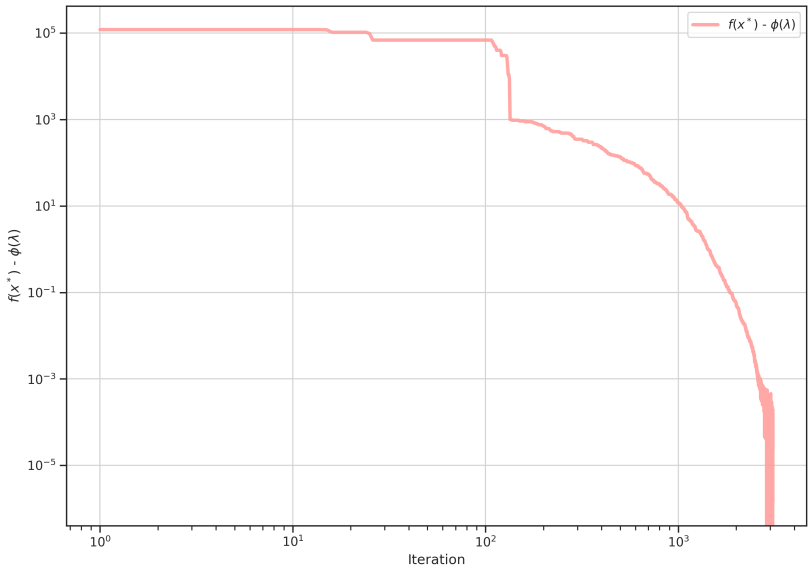


Figure 3: Dual gap with update rule 1, $n = 5000$ and $K = 10$

Stepsize used	(12)(13)
Optimal α	0.3
Total iterations	3092
total time (sec)	665.3 *s*
Best $f(x^*) - \psi(\lambda)$	1.414*e*−06
Best iteration	3092
Best $\psi(\lambda)$	1.202*e*+05
Best $\|\lambda_t - \lambda_{t-1}\|$	2.586*e*−08
Best $\|x_t - x_{t-1}\|$	5.190*e*−11
`quadprog` time	41.4 *s*
`quadprog` iterations	23

Table 5: Sum up table for $n = 5000$, $K = 10$ and rule 1

	$\alpha = 1e{-}1$	$\alpha = 1$	$\alpha = 5$	$\alpha = 10$
Stepsize used	(11)	(11)	(11)	(11)
Total iterations	6805	5000	5000	5000
total time (sec)	1867.5 *s*	1175.5 *s*	1247.1 *s*	1341.5 *s*
Best $f(x^*) - \psi(\lambda)$	4.031*e*−05	2.939*e*−02	1.085*e*+02	2.032*e*+01
Best iteration	4740	4531	5000	3641
Best $\psi(\lambda)$	1.202*e*+05	1.202*e*+05	1.201*e*+05	1.202*e*+05
Best $\|\lambda_t - \lambda_{t-1}\|$	0.001*e*+00	1.484*e*−02	6.725*e*−02	2.852*e*−01
Best $\|x_t - x_{t-1}\|$	5.98*e*−11	1.018*e*−10	4.598*e*−11	1.166*e*−10
`quadprog` time	41.4 *s*	41.4 *s*	41.4 *s*	41.4 *s*
`quadprog` iterations	23	23	23	23

Table 6: Sum up table for $n = 5000$, $K = 10$ and rule 1

Experimentations in this case confirm us the general theory, hence how it is possible to reach the desired dual gap using any possible α and the classical (11) stepsize rule. Indeed we can see how the tested values go almost near to the desire dual gap, even achieving almost the optimal one. Giving more iterations to our solver should reflect the standard subgradient convergence theory [9].

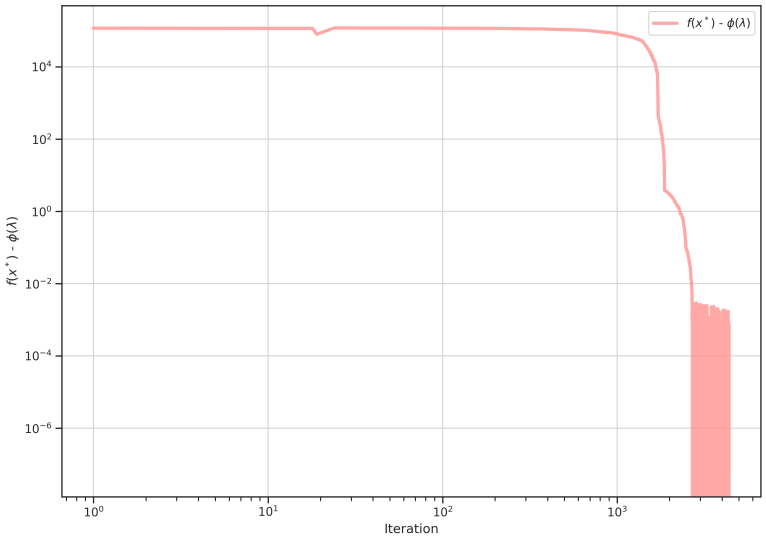


Figure 4: Dual gap with update rule 3, $n = 5000$ and $K = 10$

Stepsize used	(9)
Optimal h's	[200, 25, 5, 2, 1e−2, 1e-3]
Total iterations	4379
total time (sec)	952.5 *s*
Best $f(x^*) - \psi(\lambda)$	5.243e−08
Best iteration	4379
Best $\psi(\lambda)$	1.202e+05
Best $\|\lambda_t - \lambda_{t-1}\|$	1.471e−09
Best $\|x_t - x_{t-1}\|$	4.153e−05
`quadprog` time	41.4 *s*
`quadprog` iterations	23

Table 7: Sum up table for $n = 5000$, $K = 10$ and rule 3

We show an example using the update rule (3). As we can see, the optimization in this case required a more ad-hoc setting of the parameter. Using the other stepsize rules we did not get any improvement during the optimization, while the rule (10) is able to perform steps and reach the desired dual gap. However the value of h must be modified in the meantime based on the remaining dual gap, otherwise we perform too big steps and we get away from the optimum value.
We decided to focus on the standard update rule (4), being the most efficient and easier for hyperparameters optimization.

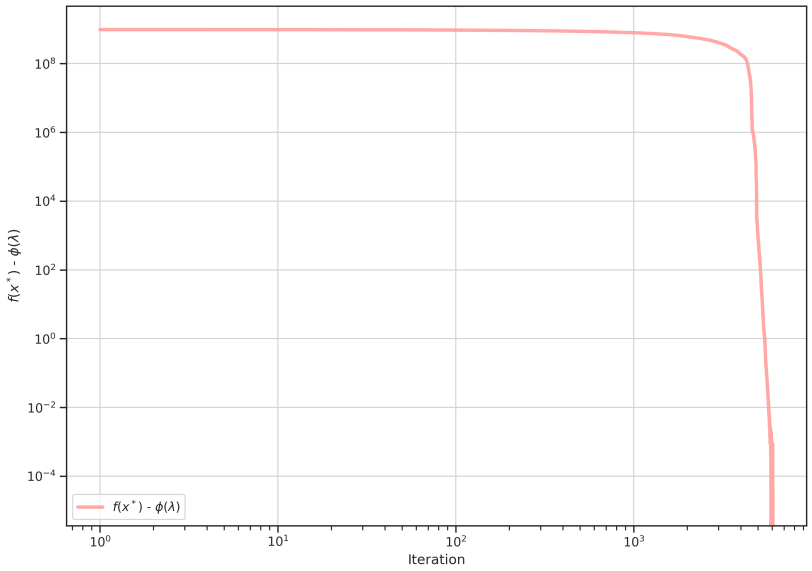


Figure 5: Dual gap with update rule 1, $n = 5000$ and $K = 1000$

Stepsize used	(12)(11)(13)
Optimal α	$[20, 10, 5]$
Total iterations	6053
total time (sec)	1710.1 s
Best $f(x^*) - \psi(\lambda)$	1.764e−05
Best iteration	6027
Best $\psi(\lambda)$	1.228e+09
Best $\|\lambda_t - \lambda_{t-1}\|$	3.025e−13
Best $\|x_t - x_{t-1}\|$	5.908e−08
`quadprog` time	76.3 s
`quadprog` iterations	26

Table 8: Sum up table for $n = 5000$, $K = 1000$ and rule 1

	$\alpha = 50$	$\alpha = 100$	$\alpha = 100$
Stepsize used	(11)	(11)	(12)(11)
Total iterations	7000	7000	7000
total time (sec)	2024.2 s	2016 s	2018.1 s
Best $f(x^*) - \psi(\lambda)$	9.429e+08	8.851e+08	8.851e+08
Best iteration	7000	7000	7000
Best $\psi(\lambda)$	2.855e+08	3.434e+08	3.434e+08
Best $\|\lambda_t - \lambda_{t-1}\|$	5.047e−01	1.009e−00	1.009e−00
Best $\|x_t - x_{t-1}\|$	2.209e−07	8.827e−02	8.827e−02
`quadprog` time	76.3 s	76.3 s	76.3 s
`quadprog` iterations	26	26	26

Table 9: Sum up table for $n = 5000$, $K = 1000$ and rule 1

Also in this latter case, we can see how adapting the stepsize led us to an improvement in the solution, reaching the dual gap in a faster way. Employing different stepsize rules and trying different hyperparameter values allow us to find an optimal setting of them which are suitable for reaching the desired gap.
Surely other settings of hyperparameters exists that would allow us to reach the desired objective, faster of slower than this. We can see an example in 6, where with a fixed $\alpha = 0.1$ and a fixed rule (11) we reach the same the desired dual gap with more iterations than the optimal settings we found.

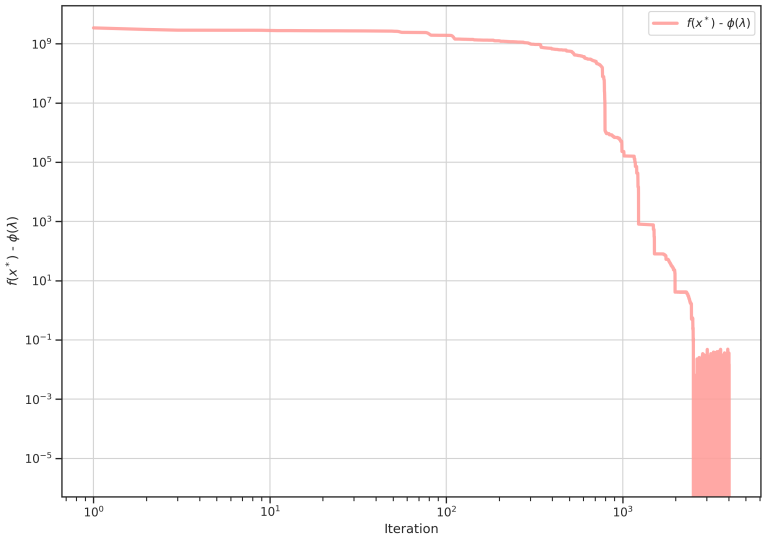


Figure 6: Dual gap with update rule 1, $n = 5000$ and $K = 2500$

Stepsize used	(12)(11)
Optimal α	$[40, 20, 10,$ $5, 2, 1, 1e{-}1,$ $1e{-}3, 1e{-}4]$
Total iterations	4006
total time (sec)	1163.5 s
Best $f(x^*) - \psi(\lambda)$	2.861e−06
Best iteration	3758
Best $\psi(\lambda)$	7.759e+09
Best $\|\lambda_t - \lambda_{t-1}\|$	1.881e−06
Best $\|x_t - x_{t-1}\|$	1.534e−06
`quadprog` time	69.9 s
`quadprog` iterations	28

Table 10: Sum up table for $n = 5000$, $K = 2500$ and rule 1

This last example show the scaling of K using the same value of n. As we can see, the same holds here, like what happened for $n = 1000$. Increasing the number of constraints take less time to optimize the function, since many of disjoint sets now are composed by less elements, and so it is more easier to fix the value of the iterates, both for our solver and for *quadprog*. Also here we performed some other experiments using different rules and hyperparameters' values.

	$\alpha = 100$	$\alpha = [100, 50, 10, 5]$
Stepsize used	(11)	(12)(11)(13)
Total iterations	7000	7000
total time (sec)	2224.4 s	2048.9 s
Best $f(x^*) - \psi(\lambda)$	9.032e+08	4.441e+03
Best iteration	7000	6976
Best $\psi(\lambda)$	6.855e+09	7.759e+09
Best $\|\lambda_t - \lambda_{t-1}\|$	1.01e−00	1.004e−05
Best $\|x_t - x_{t-1}\|$	3.366e−02	2.369e−06
`quadprog` time	69.9 s	69.9 s
`quadprog` iterations	28	28

Table 11: Sum up table for $n = 5000$, $K = 2500$ and rule 1

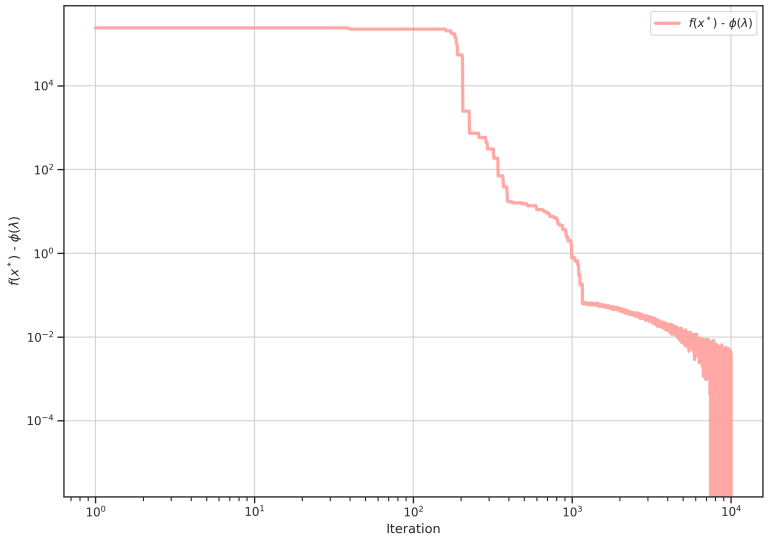


Figure 7: Dual gap with update rule 1, $n = 10000$ and $K = 10$

Stepsize used	(12)(11)(13)
Optimal α	$[0.2, 1{-}1, 1e{-}2,$ $1e{-}3, 1e{-}4, 1e{-}6]$
Total iterations	10000
total time (sec)	7446.3 s
Best $f(x^*) - \psi(\lambda)$	5.185e−06
Best iteration	9699
Best $\psi(\lambda)$	2.428e+05
Best $\|\lambda_t - \lambda_{t-1}\|$	3.153e−07
Best $\|x_t - x_{t-1}\|$	1.263e−10
`quadprog` time	401.6 s
`quadprog` iterations	28

Table 12: Sum up table for $n = 10000$, $K = 10$ and rule 1

Finally, we were able to scale n also to 10000, our machine supported the magnitude of the problem and was capable of computing the solution.
Also here we show a comparison using different hyperparameters:

	$\alpha = 0.2$	$\alpha = 0.3$
Stepsize used	(11)	(11)
Total iterations	5000	5000
total time (sec)	2678.1 s	3020.3 s
Best $f(x^*) - \psi(\lambda)$	2.022e+05	2.361e+05
Best iteration	5000	5000
Best $\psi(\lambda)$	4.055e+04	6.689e+03
Best $\|\lambda_t - \lambda_{t-1}\|$	3.804e−03	5.706e−03
Best $\|x_t - x_{t-1}\|$	7.211e−11	1.162e−11
`quadprog` time	401.6 s	401.6 s
`quadprog` iterations	28	28

Table 13: Sum up table for $n = 10000$, $K = 10$ and rule 1

Also here an ad-hoc selection of the values of α was needed.

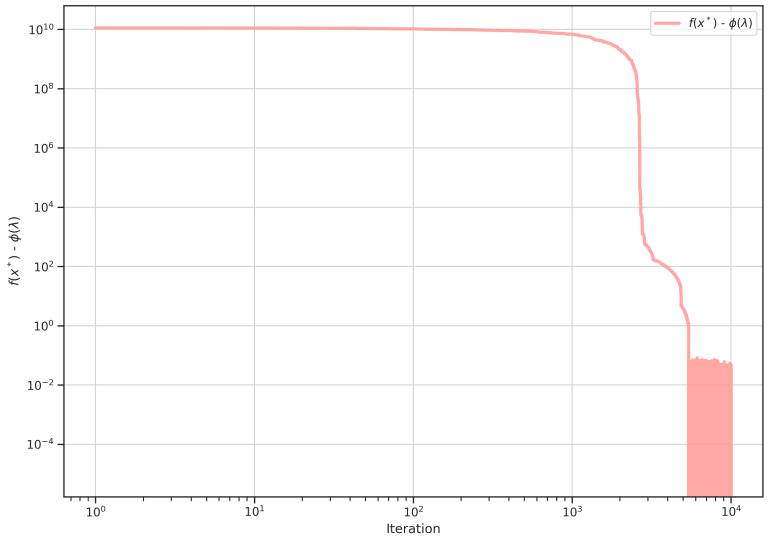


Figure 8: Dual gap with update rule 1, $n = 10000$ and $K = 2500$

Stepsize used	(12)(11)
Optimal α	$[50, 20, 5, 1,$ $1e{-}1, 1e{-}2, 1e{-}3,$ $1e{-}4, 5e{-}4, 1e{-}5,$ $5e{-}5, 1e{-}6, 1e{-}7,$ $1e{-}8, 1e{-}9, 1e{-}10]$
Total iterations	10000
total time (sec)	7272.9 *s*
Best $f(x^*) - \psi(\lambda)$	9.536e−06
Best iteration	6140
Best $\psi(\lambda)$	1.544e+10
Best $\|\lambda_t - \lambda_{t-1}\|$	1.634e−09
Best $\|x_t - x_{t-1}\|$	1.481e−06
`quadprog` time	401.6 *s*
`quadprog` iterations	28

Table 14: Sum up table for $n = 10000$, $K = 2500$ and rule 1

Lastly, we show the scaling of K also over the big value of $n = 10000$. In this case, we notice how the same timing is required in both $K = 10$ and $K = 2500$. We can assert from this that time scaling is consistent (hence we do not increase time by adding many constraints) but at the same time we need more attention when we fix the hyperparameters.
As we can see, in particular here, a lot of α values need to be changed during the optimization.

	$\alpha = [50, 20, 5, 1, 1e{-}1, 1e{-}2,$ $1e{-}3, 1e{-}4, 5e{-}4, 1e{-}5]$
Stepsize used	(12)(11)(13)
Total iterations	10000
total time (sec)	7582 *s*
Best $f(x^*) - \psi(\lambda)$	6.524e−01
Best iteration	9484
Best $\psi(\lambda)$	1.544e+10
Best $\|\lambda_t - \lambda_{t-1}\|$	1.144e−09
Best $\|x_t - x_{t-1}\|$	1.537e−06
`quadprog` time	401.6 *s*
`quadprog` iterations	28

Table 15: Sum up table for $n = 10000$, $K = 2500$ and rule 1

To conclude, we can see overall that in almost all cases we are still quite far from the time took by `quadprog`.
Our explanation about this resides first of all in the machine used, which is not a very powerful machine (8 `GB` of `RAM` and a quadcore processor). Then, we also have a lot of vectorizable operations and the solution of the `KKT` linear system. Indeed, we used the underlying `BLAS` operations provided by `Julia` to speed up the different computations and saving memory by overwriting the non fixed objects, avoid using matrices and efficiently compute the `KKT` linear system solution. Using this tweak speed up the timing but sadly not better than *quadprog*.
Surely adapting the hyperparameters differently and changing the stepsize in a different manner could improve the time required by our solver to reach the solution, performing big steps and reaching the optimal gap in a faster way, maybe approaching the same order of time of the off-the-shelf solver.

5 Code description

The code is provided in `Julia` through package. To try it, simply open the `Julia REPL` inside the current folder, and launch `julia` specifying the option

```
--project=.
```

then enter in the `pkg` mode using `]` and launch

```
instantiate
```

to download the required modules. Then go back to the command line tool (simply backspace) and run

```
include(src/main.jl)
```

At this point you will be asked for a value of n and K.
All the code is contained inside the `src` folder:

- *Utils.jl*: module containing some useful functions:
  - `construct_full_matrix(`Q`, `A`, `K`)`: return the entire `KKT` matrix;
  - `construct_A(`K`, `n`, `I_K`)`: return the constraint matrix A as described in the report;
- *ADAGRAD_Solver.jl*: module containing:
  - a struct `Solver`, containing all the parameters and return value needed for the problem;
  - functions to compute the λ/x-norm, the subgradient, the γ value and the update rules;
  - `my_ADAGRAD`, which implement the algorithm derived in the report;
- *main.jl*: used to test all the code;
- *solve_prob.m*: `MATLAB` script to compute optimal primal solution using `quadprog`;

In addition, the entire project folder contains:

- *papers*: folder containing all the literature referenced in the bibliography;
- *results*: folder containing all the plots and logs obtained with the experimentations values of table 3;
- *util.sh*: a clean-up routine for the *logs* and *plots* produced;
- *analyzer.py*: used to inspect the collected *.csv* files and get some stats;
- *plotter.py*: used to generate plots from the *.csv* files;

References

[1] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(61):2121–2159, 2011.

[2] Lin Xiao. Dual averaging method for regularized stochastic learning and online optimization. In Y. Bengio, D. Schuurmans, J. Lafferty, C. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, volume 22. Curran Associates, Inc., 2009.

[3] John Duchi, Shai Shalev-Shwartz, Yoram Singer, and Ambuj Tewari. Composite objective mirror descent. In *Composite Objective Mirror Descent*, pages 14–26, 12 2010.

[4] Antonio Frangioni, Bernard Gendron, and Enrico Gorgone. On the computational efficiency of subgradient methods: a case study with lagrangian bounds. *Mathematical Programming Computation*, 9(4):573–604, Dec 2017.

[5] Neal Parikh and Stephen Boyd. Proximal algorithms. *Foundations and Trends® in Optimization*, 1(3):127–239, 2014.

[6] Yurii Nesterov. Primal-dual subgradient methods for convex problems. *Mathematical Programming*, 120(1):221–259, Aug 2009.

[7] Marina A. Epelman. Ioe 511/math 652: Continuous optimization methods, section 1, 2007. Lecture notes `http://www-personal.umich.edu/~mepelman/teaching/511notesFA07.pdf`.

[8] Ang Andersen. Projection onto nonnegative orthant, rectangular box and polyhedron, 2020. First draft: March 19, 2020; Last update: December 23, 2020. Université de Mons, `https://angms.science/doc/CVX/Proj_nonnegBoxpoly.pdf`.

[9] G. d'Antonio and A. Frangioni. Convergence analysis of deflected conditional approximate subgradient methods. *SIAM J. Optim.*, 20:357–386, 2009.

A Appendix A: Updates derivations

Given a matrix A of size $n \times n$, the proximal term expression of a vector x of size $n \times 1$ at step t is

$$\Psi_t(x) = \frac{1}{2} \langle x, Ax \rangle$$

A.1 Differentiating proximal term

Derivation of proximal term arises from the need of obtaining the $\arg\max x$ of a given update function. Starting by the definition

$$\frac{\partial \Psi_t(x)}{\partial x} = \frac{1}{2} \frac{\partial}{\partial x} \langle x, Ax \rangle$$

which can be rewritten in a more classical form, and the derivative is immediate

$$\frac{1}{2} \frac{\partial}{\partial x} x^T Ax = Ax$$

A.2 Derivation of primal-dual update

We can now look into the detailed derivation of the update (2). Starting from the equation

$$\lambda_{t+1} = \arg\max_{\lambda \in \mathcal{X}} \{\eta \langle \overline{g}_t, \lambda \rangle + \frac{1}{t} \Psi_t(\lambda)\}$$

we want to achieve the $\arg\max$ over the set $\mathcal{X}$. Focusing on the maximum problem, we want to get the maximum λ, hence:

$$\begin{aligned} \frac{\partial}{\partial \lambda} \eta \langle \overline{g}_t, \lambda \rangle + \frac{\partial}{\partial \lambda} \frac{1}{t} \Psi_t(\lambda) = 0 \\ \eta \, \overline{g}_t + \frac{1}{2t} \frac{\partial}{\partial \lambda} \langle \lambda, H_t \lambda \rangle = 0 \end{aligned}$$

Knowing that H_t is a diagonal matrix and using the previous derivation:

$$\frac{\partial}{\partial \lambda} \langle \lambda, H_t \lambda \rangle = 2 \, H_t \, \lambda$$

Substituting this into the above derivation, we get the maximum λ:

$$\begin{aligned} \eta \, \overline{g}_t + \frac{1}{t} H_t \, \lambda = 0 \\ \lambda = -H_t^{-1} \, t \, \eta \, \overline{g}_t \end{aligned}$$

A.3 Derivation of composite-mirror update

We follow the same approach also for the update (3). Starting from the definition

$$\lambda_{t+1} = \arg\max_{\lambda \in \mathcal{X}} \{\eta \langle g_t, \lambda \rangle + B_{\Psi_t}(\lambda, \lambda_t)\}$$

where we remark the definition of Bregman divergence

$$B_{\Psi_t}(\lambda, \lambda_t) = \Psi_t(\lambda) - \Psi_t(\lambda_t) - \langle \nabla \Psi_t(\lambda_t), \lambda - \lambda_t \rangle$$

Also in this case, to find the $\arg\max$ we derive the update formula and set it to be zero

$$\begin{aligned} \frac{\partial}{\partial \lambda} \eta \langle g_t, \lambda \rangle + \frac{\partial}{\partial \lambda} B_{\Psi_t}(\lambda, \lambda_t) = 0 \\ \eta \, g_t + \frac{\partial}{\partial \lambda} [\Psi_t(\lambda) - \Psi_t(\lambda_t) - \langle \nabla \Psi_t(\lambda_t), \lambda - \lambda_t \rangle] = 0 \\ \eta \, g_t + H_t \, \lambda - \frac{\partial}{\partial \lambda} \langle \nabla \Psi_t(\lambda_t), \lambda - \lambda_t \rangle = 0 \end{aligned}$$

Considering the first term of the scalar product, we have to evaluate the first order taylor model of the proximal function Ψ_t at the known quantity λ_t. It is easy to see that by using the derivation in the first paragraph and the fact that H_t is diagonal

$$\nabla\Psi_t(\lambda_t) = \nabla\frac{1}{2}\langle\lambda_t, H_t\lambda_t\rangle = H_t\lambda_t$$

Putting this into equation

$$\frac{\partial}{\partial\lambda}\langle H_t\lambda_t, \lambda - \lambda_t\rangle = \frac{\partial}{\partial\lambda}\sum_{i=1}^{n} h_{ii}\lambda_{t,i}(\lambda_i - \lambda_{t,i})$$

And so is easy to see that

$$\frac{\partial}{\partial\lambda}\langle H_t\lambda_t, \lambda - \lambda_t\rangle = \begin{bmatrix} \frac{\partial}{\partial\lambda_1}\langle H_t\lambda_t, \lambda - \lambda_t\rangle \\ \frac{\partial}{\partial\lambda_2}\langle H_t\lambda_t, \lambda - \lambda_t\rangle \\ \cdots \\ \frac{\partial}{\partial\lambda_n}\langle H_t\lambda_t, \lambda - \lambda_t\rangle \end{bmatrix} = \begin{bmatrix} h_{11}\lambda_{t,1} \\ h_{22}\lambda_{t,2} \\ \cdots \\ h_{nn}\lambda_{t,n} \end{bmatrix} = H_t\lambda_t$$

And finally putting this into the main problem, we get:

$$\eta\, g_t + H_t\,\lambda - H_t\lambda_t = 0 \quad \Longrightarrow\; H_t\,\lambda = H_t\lambda_t - \eta\, g_t$$
$$\lambda = H_t^{-1}\left[H_t\lambda_t - \eta\, g_t\right] = \lambda_t + \eta\, H_t^{-1}\, g_t$$

B Appendix B: subgradient computation

Utterly following the definition, we state that s is a subgradient of ψ at x

$$\psi(\lambda) \leq \psi(x) + s^T(\lambda - x) \quad \forall\lambda \in \mathbb{R}^n$$

which reordering the term can be written as

$$s^T(\lambda - x) \geq \psi(\lambda) - \psi(x) \quad \forall\lambda \in \mathbb{R}^n$$

Either we can solve the above complex inequality with unknown variables s and λ, or better we can apply the following reasoning. The subdifferential is the set containing all the numbers in the interval $[a, b]$ such that

$$a = \lim_{\lambda\to\lambda_0^-} \frac{\psi(\lambda) - \psi(\lambda_{t-1})}{\lambda - \lambda_{t-1}}$$
$$b = \lim_{\lambda\to\lambda_0^+} \frac{\psi(\lambda) - \psi(\lambda_{t-1})}{\lambda - \lambda_{t-1}}$$

where the $^+$ and $^-$ portion must have a certain value greater than zero (order of 10^{-2}).