Project Non-ML 5

Matteo De Francesco

Contents

1 Introduction

In this report we will analize the convex quadratic problem

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

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:

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

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

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

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:

$$R_\phi(T) \triangleq \sum_{t=1}^{T} \psi(\lambda_t) - \psi(\lambda^*) \tag{2}$$

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

$$\lambda_{t+1} = \arg\min_{\lambda \in \mathcal{X}} \{\eta \langle \overline{g}_t, \lambda \rangle + \frac{1}{t} \Psi_t(\lambda)\} \tag{3}$$

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

$$\lambda_{t+1} = \arg\min_{\lambda \in \mathcal{X}} \{\eta \langle g_t, \lambda \rangle + B_{\Psi_t}(\lambda, \lambda_t)\} \tag{4}$$

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:

$$\lambda_{t+1} = P_{\mathcal{X}}\{\lambda_t - \eta \, diag(G_t)^{-1/2} g_t\} \tag{5}$$

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 (3) and (4). Proofs can be found in Appendix F of [1].

Proposition 2.1. *Let the sequence $\{\lambda_t\}$ be defined by the update (3). For any $\lambda^* \in \mathcal{X}$*

$$\sum_{t=1}^{T} \psi(\lambda_t) - \psi(\lambda^*) \leq \frac{1}{\eta}\Psi_T(\lambda^*) + \frac{\eta}{2}\sum_{t=1}^{T} \|\psi'(\lambda_t)\|_{\Psi_{t-1}^*}^2 \tag{6}$$

Proposition 2.2. *Let the sequence $\{\lambda_t\}$ be defined by the update (4). 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 (3) and (4) 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 (3) or (4)
  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* (3) *with $\delta \geq \max_t \|g_t\|_\infty$, for any $\lambda^* \in \mathcal{X}$*

$$R_\phi(T) \leq \underbrace{\frac{\delta}{\eta}\|\lambda^*\|_2^2 + \frac{1}{\eta}\|\lambda^*\|_\infty^2 \sum_{i=1}^{d}\|g_{1:T,i}\|_2}_{①} + \underbrace{\eta\sum_{i=1}^{d}\|g_{1:T,i}\|_2}_{②} \tag{7}$$

For λ_t generated using the update (4)*, for any $\lambda^* \in \mathcal{X}$*

$$R_\phi(T) \leq \underbrace{\frac{1}{2\eta}\max_{t\leq T}\|\lambda^* - \lambda_t\|_\infty^2 \sum_{i=1}^{d}\|g_{1:T,i}\|_2}_{③} + \underbrace{\eta\sum_{i=1}^{d}\|g_{1:T,i}\|_2}_{②} \tag{8}$$

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 (7) and (8).
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 (7) and (8), 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 (3) 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 (4) *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 (3) and (4) 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), (3) and (4):

$$\begin{aligned} \lambda_{t+1} &= P_{\mathcal{X}}\{\lambda_t + \eta \, diag(G_t)^{-1/2} g_t\} \\ \lambda_{t+1} &= \arg\max_{\lambda \in \mathcal{X}} \{\eta \langle \overline{g}_t, \lambda \rangle + \frac{1}{t} \Psi_t(\lambda)\} \\ \lambda_{t+1} &= \arg\max_{\lambda \in \mathcal{X}} \{\eta \langle g_t, \lambda \rangle + B_{\Psi_t}(\lambda, \lambda_t)\} \end{aligned} \tag{9}$$

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.

$$\hat{\lambda}_{t+1} = \lambda_t + \eta diag(G_t)^{-1/2} g_t \qquad \text{(Update (5))}$$

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

$$\hat{\lambda}_{t+1} = \lambda_t - \eta \, H_t^{-1} \, g_t \qquad \text{(Update (4))}$$

For the stepsize η, this can be fixed apriori, by using the classical DSS or Polyak stepsize.
For the dual variables, after we obtain the $\hat{\lambda}_{t+1}$, 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 [7].
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

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

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:

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

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

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 factorization:

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

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

which can be solved efficiently through forward substitution (for the matrix L) and back substitution (for the matrix U) with a complexity of $O((n+K)^2)$.
In the most general case the pivoted LU factorization is preferred, but we will see in the next chapter that the implementation language automatically identify the best factorization to use (for example if the full matrix is positive definite, *Cholesky* is better, as well as if the matrix is symmetric, the *LDL* or *Bunch-Kaufman* factorization are better).
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

$$\|\lambda_t - \lambda_{t-1}\|_2 \leq \varepsilon \tag{11}$$

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

$$f(x^*) - \psi(\lambda_i) \leq \tau$$

for each λ_i computed during iterations.
Lastly, an optional parameter is given as input to the algorithm. The parameter $defl$ is a boolean value indicating the use of deflection or not, whose derivation is described in the Appendix C.
Putting everything together, we obtain a slightly different algorithm than 1:

Algorithm 3 `ADAGRAD` on our dual problem

function `ADAGRAD`(η, δ, ε, max_iter, $defl$)
  $\lambda_0 \leftarrow 0$
  $g_{1:0} \leftarrow [\,]$
  $x_0 \leftarrow 0$
  for $t \leftarrow 1$ to T **do**
    $Loss \leftarrow f(x_{t-1})$
    $g_t \leftarrow \dfrac{\partial \psi_\lambda(\lambda)}{\partial \lambda}$ ▷ Compute subgradient B
    if $defl$ **then** ▷ Compute deflection
      Compute γ using C
      Update $g_t = \gamma g_t + (1-\gamma) g_{t-1}$
    end if
    $g_{1:t} \leftarrow [g_{1:t-1}\ g_t]$ ▷ Store subgradient
    $s_{t,i} \leftarrow \|g_{1:t,i}\|_2$ ▷ Solution of the problem 2.4
    $H_t =\leftarrow \delta I + diag(s_t)$
    $x_t = \arg\min_{x \in \mathcal{Y}} \left\{ x^T Q x + qx - \langle \lambda_{t-1}, x \rangle \right\}$ ▷ Lagrangian
    $\hat{\lambda}_t =$ one among (9)
    $\lambda_t = P_{\mathcal{X}}(\hat{\lambda}_t)$
    Check stopping conditions (11) or max_iter reached
  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 dimensions of the problem n and the number of simplices K for each different n we want to test.
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 resulting solutions of the three different update rules are saved and compared with the optimal primal solution $f(x^*)$ found using the off-the-shelf solver *JuMP.jl* [8].
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 x, λ, each λ-norm and x-norm residual, the number of iterations, the value of the Lagrangian function and the gap at the current iteration are displayed step by step and stored in a *csv* file. At the end of the algorithm runs, the collected values are used to display a convergence plot (*iterations* vs $\phi(\lambda_t)$), a residual visualization plot of the λ-norm and x-norm (*iterations* vs λ/*x-norms*) and finally a gap residual plot (*iterations* vs $f(x^*) - \phi(\lambda_t)$). All the duality gaps and the resulting value of the Lagrangian functions are saved too in a dictionary and printed at the end for the sake of identify the best duality gap (among the different update rules) and the function value.
During the execution of the algorithm, the initial parameters are showed, as well as the result of *JuMP.jl*. Then, the customized `ADAGRAD` starts with the three different variations, showing at each iteration the values described before (the time elapsed, the $\mathcal{L}$ value, both the λ norm residual and the x one, and the dual gap).

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**	ε	**max_iter**
2	$[1, 2]$	10^{-6}	100000
5	$[2, 3, 4]$	10^{-6}	100000
10	$[3, 5, 7]$	10^{-6}	100000
20	$[5, 10, 15]$	10^{-6}	100000
25	$[7, 13, 19]$	10^{-6}	100000
40	$[10, 20, 30]$	10^{-6}	100000
50	$[10, 20, 30, 40]$	10^{-6}	100000
100	$[20, 33, 50, 66, 80]$	10^{-6}	100000

Note that in each execution also the value of $defl$ was set to true.
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, then the next executions are much much faster. We suggest to do first executions with low magnitude of dimensions ($n = 2$ and $K = [1, 2]$) and ignore the results, and then start the experiments with whatever value you choose.
The creation of the matrix Q and the starting value of the iterates x and λ are a key point for the algorithm. The matrix Q is created using a support matrix A_help and a vector $vect$. Q is constructed using the eigenvalue decomposition $A\_help * vect * A\_help^{-1}$, by choosing an appropriate non singular matrix A_help and a vector of eigenvalues $vect$ which is filled with positive numbers, and an arbitrary number of zeros (one fifth of the dimension n) based on a certain probability.
What has been observed is that having a very bad conditioned matrix Q (with many 0 eigenvalues) results in an unstable optimization, with the value of the Lagrangian and the gap oscillating between big positive numbers and big negative ones.
In alternative, Q can be created as a covariance matrix, computing $A\_help^T * A\_help$ obtaining a surely positive semidefinite matrix which is also symmetric.
When instead Q has a lower magnitude of 0 eigenvalues or only positive eigenvalues, the problem is well conditioned and the values returned at each iteration are consistent.
Another observation regard the subgradient. Being the ϕ non differentiable in the general case, the direction is not guaranteed to be ascent/descent. So we keep track of the best function value, best iteration, best x and λ to get the suboptimal solution found. This sometimes result in the Lagrangian function being minimized instead of maximized. This is due to the fact that there is no guarantee that the direction is an ascent one.
Lastly, we can observe that using the deflection result in a more "stable" optimization, with the value of $\mathcal{L}$ stabilizing in a certain interval. In addition, the update rule 2 (which does not depend on the previous λ) seems to be the most effective in general.

We can report now some plots comparing the results of the executions obtained together with the total time spent in the algorithm execution, with fixed value of $n = 50$, number of simplices set $K = 40$, first using a positive definite Q and then a positive semidefinite Q.
Following, a table will sum up all the results obtained.
Further, we show also a sum up table of the execution with other parameters n and K.

Positive definite Q

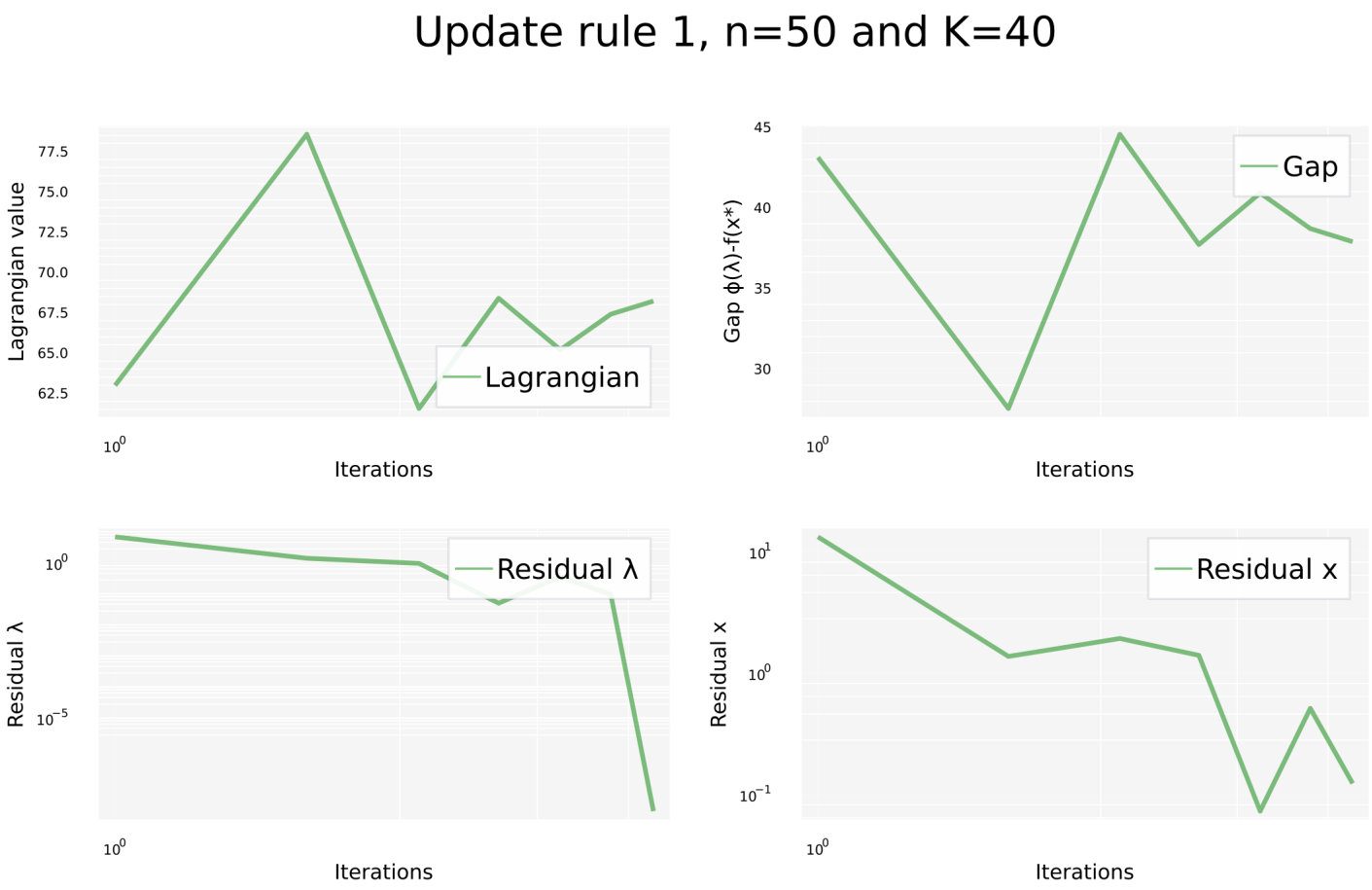

Figure 1: Update rule 1 without deflection

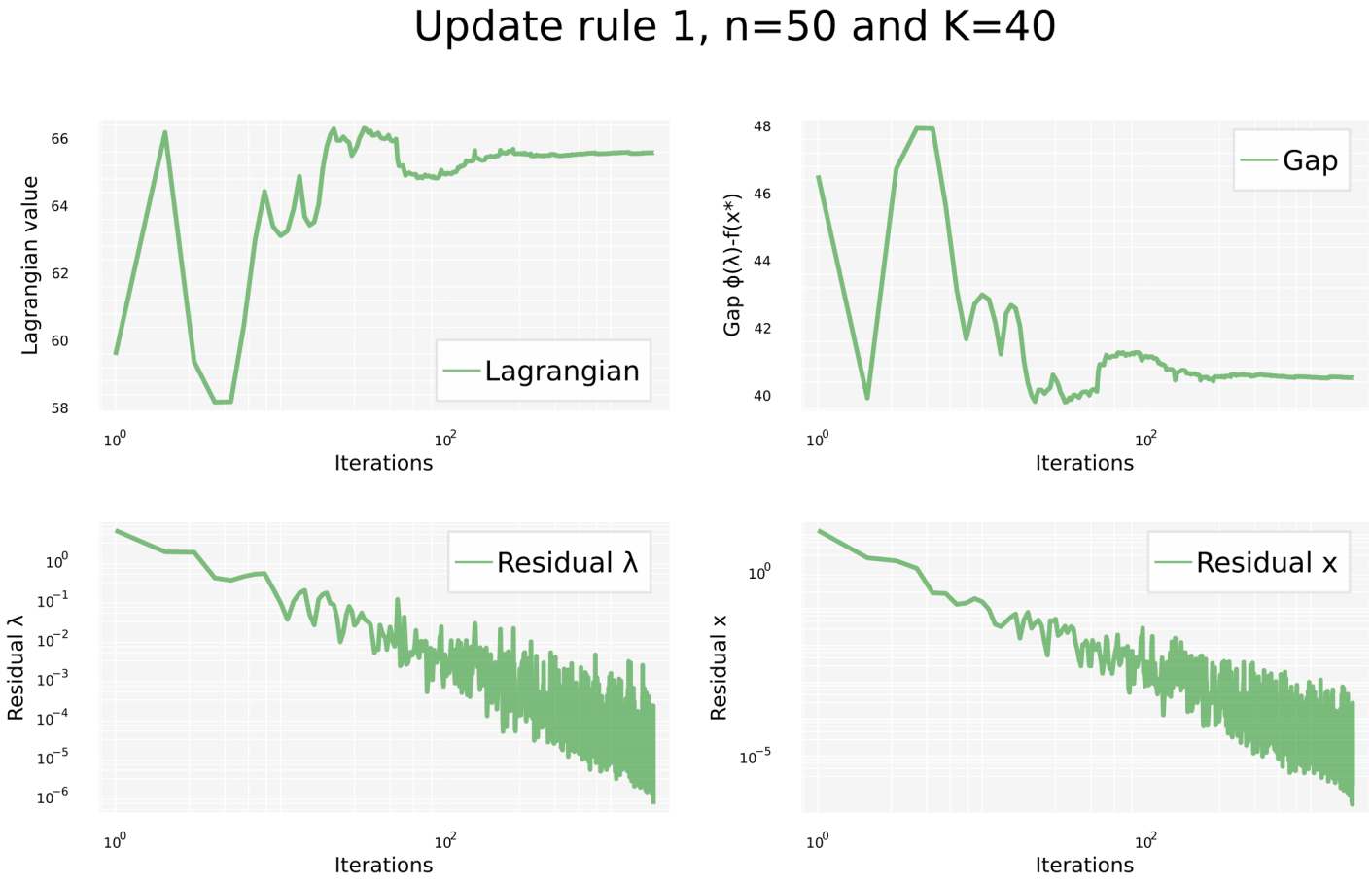

Figure 2: Update rule 1 with deflection

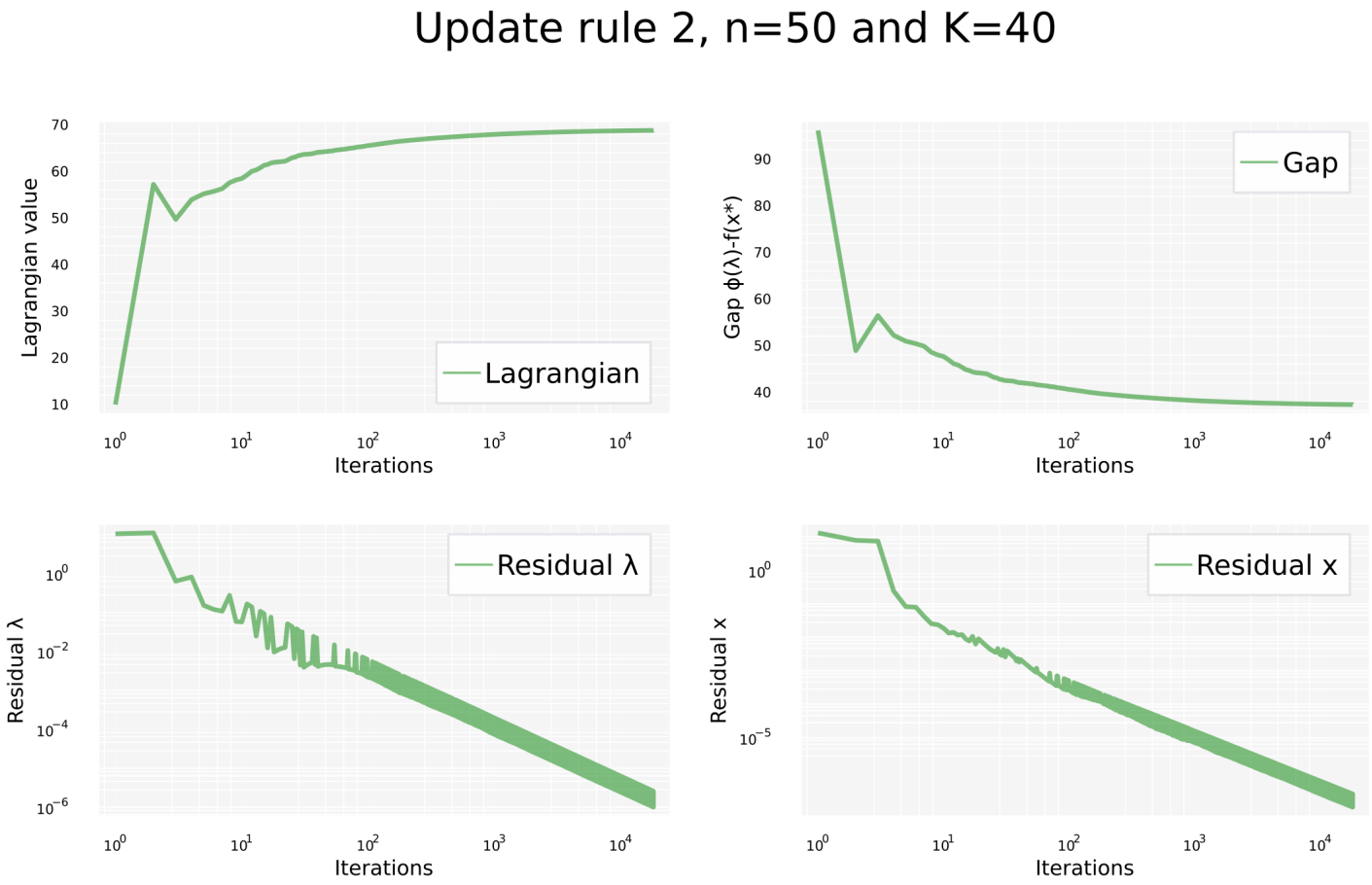

Figure 3: Update rule 2 without deflection

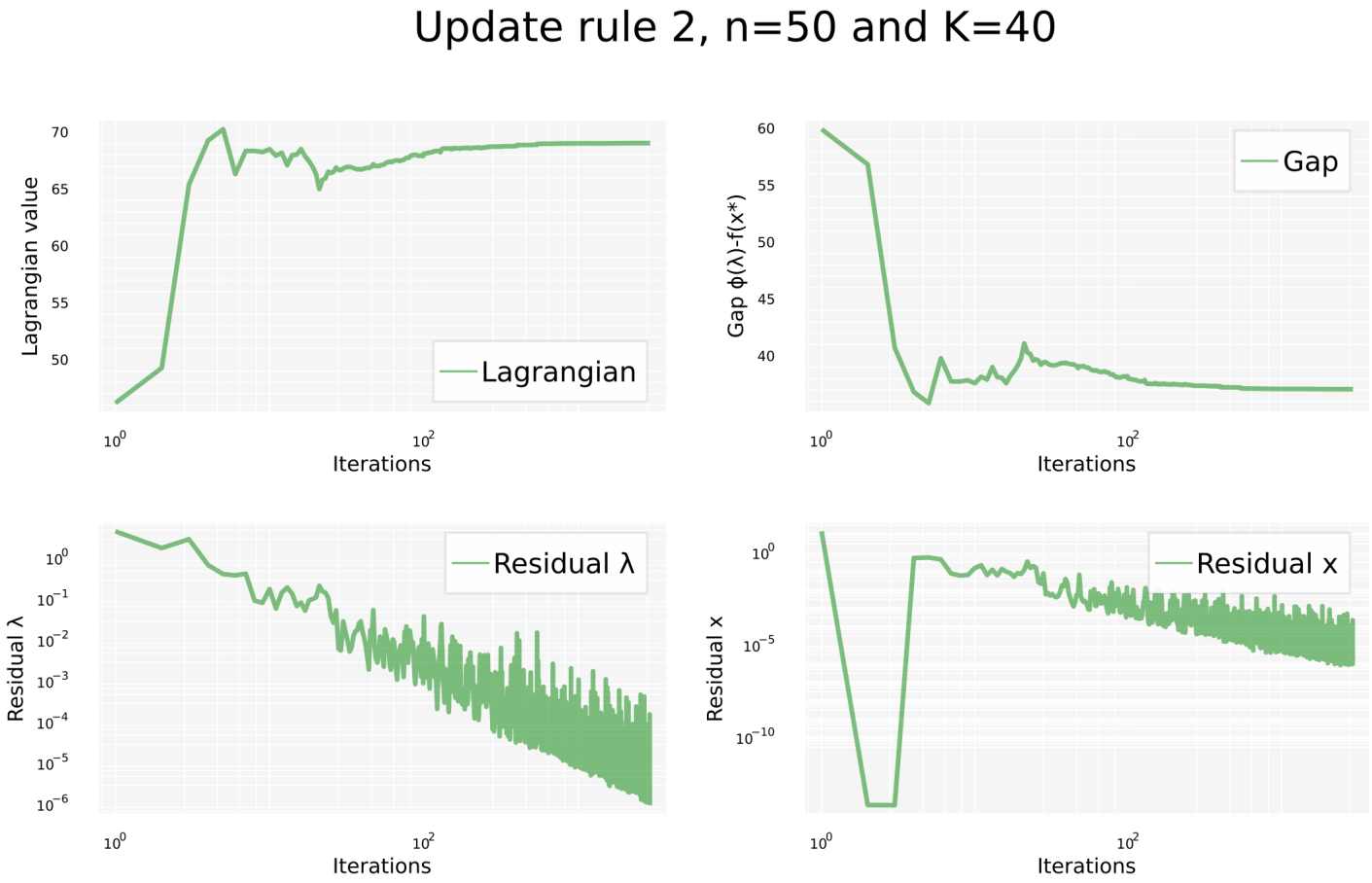

Figure 4: Update rule 2 with deflection

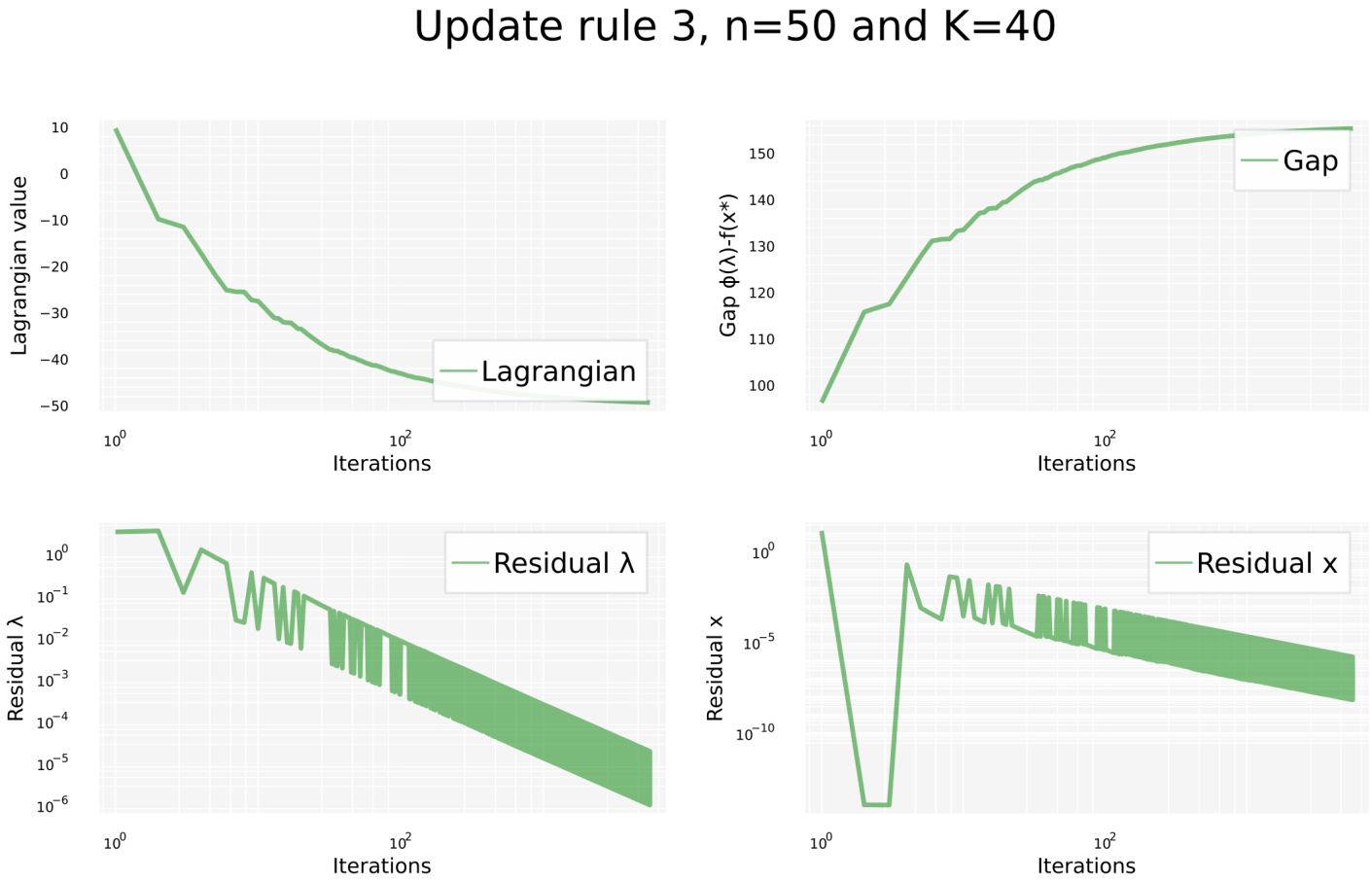

Figure 5: Update rule 3 without deflection

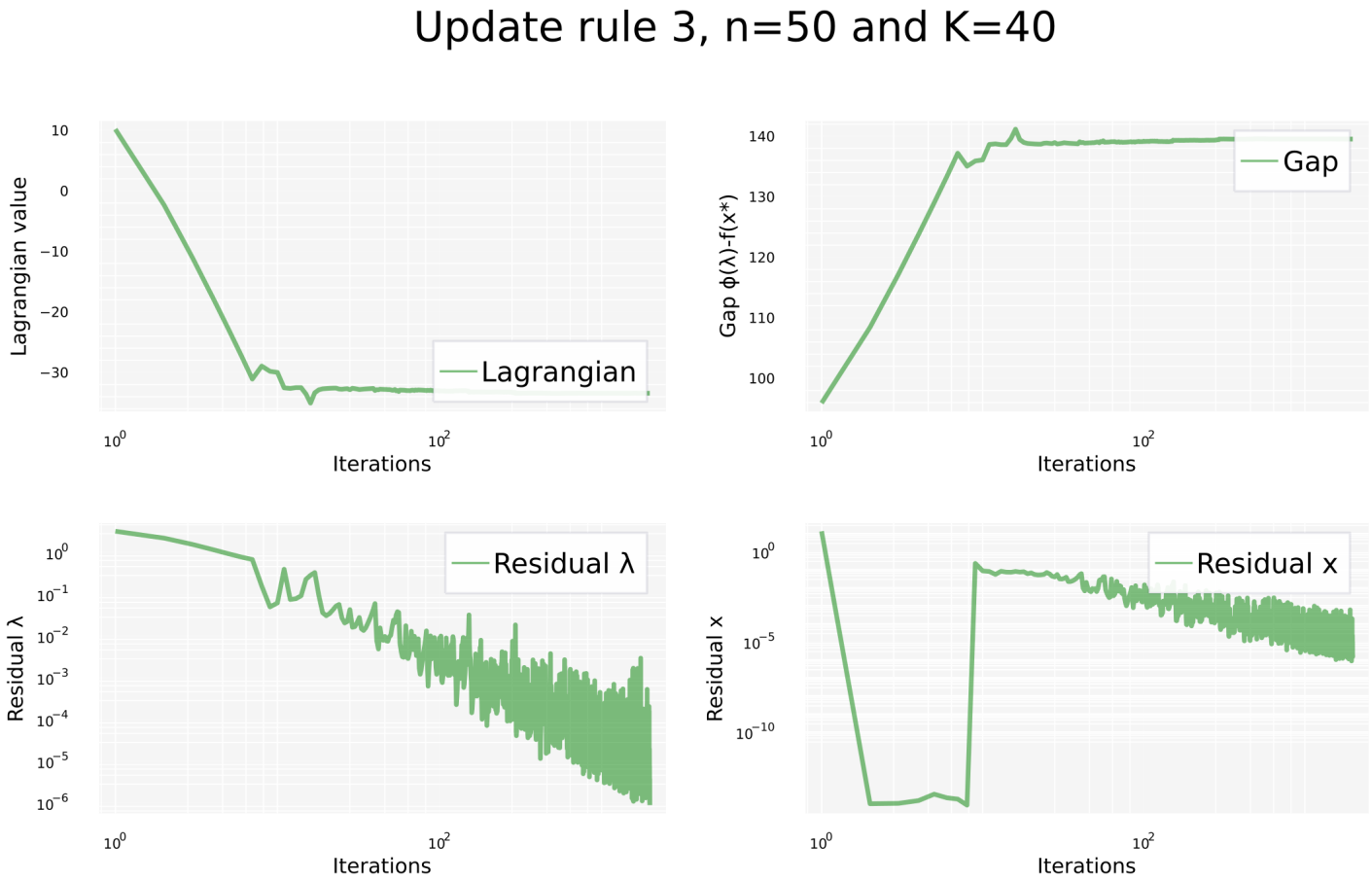

Figure 6: Update rule 3 with deflection

Positive semidefinite Q

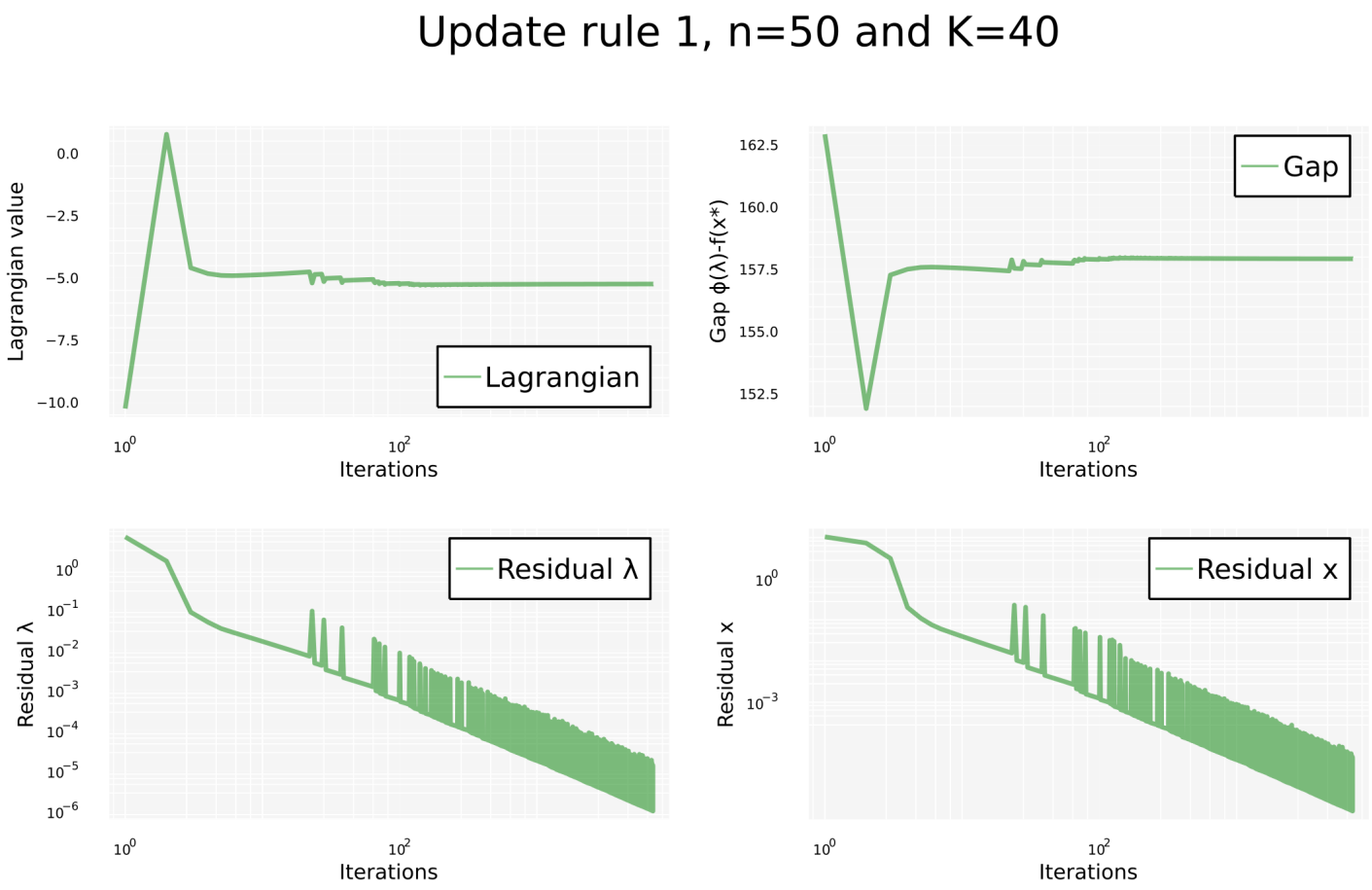

Figure 7: Update rule 1 without deflection

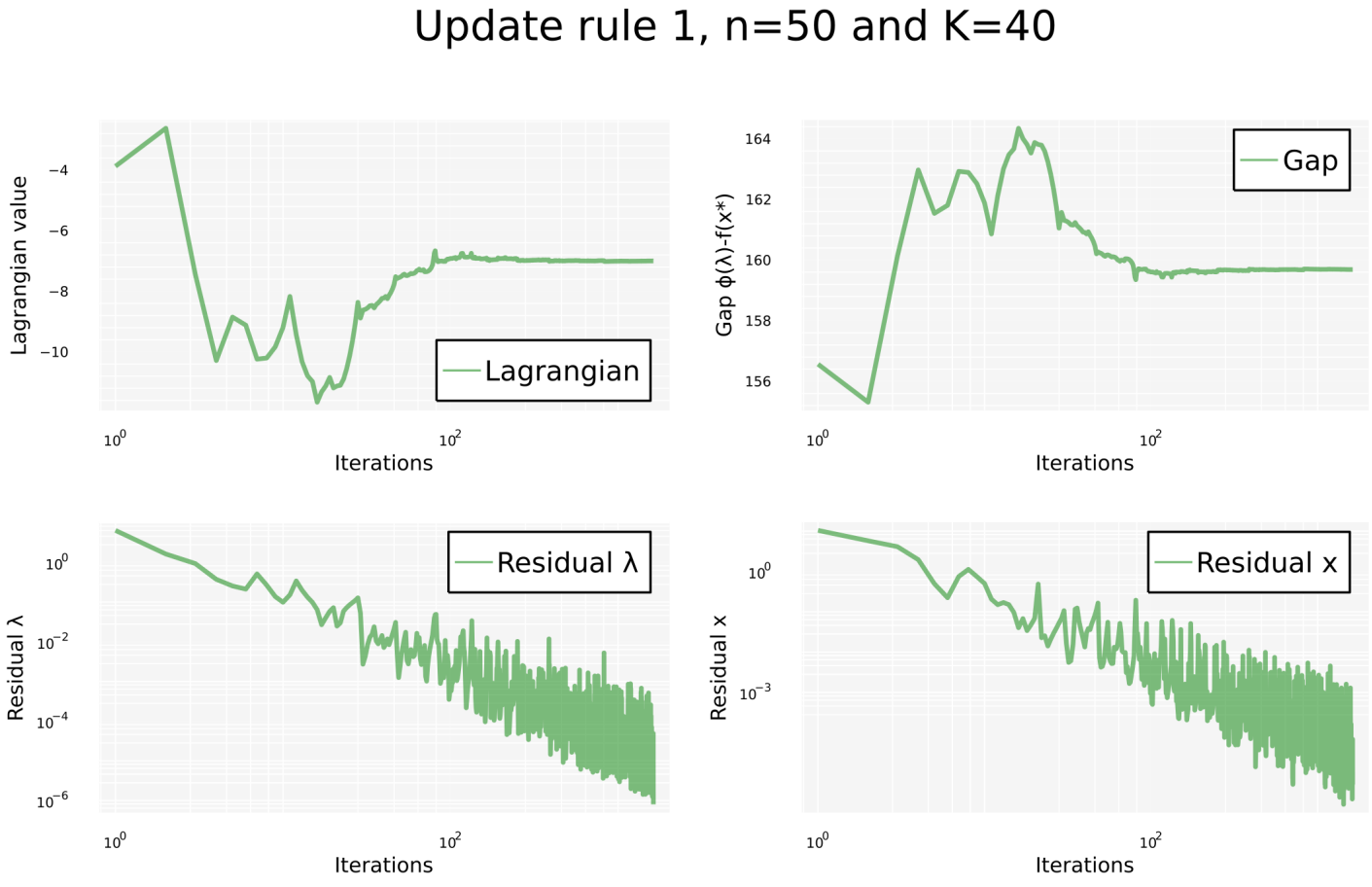

Figure 8: Update rule 1 with deflection

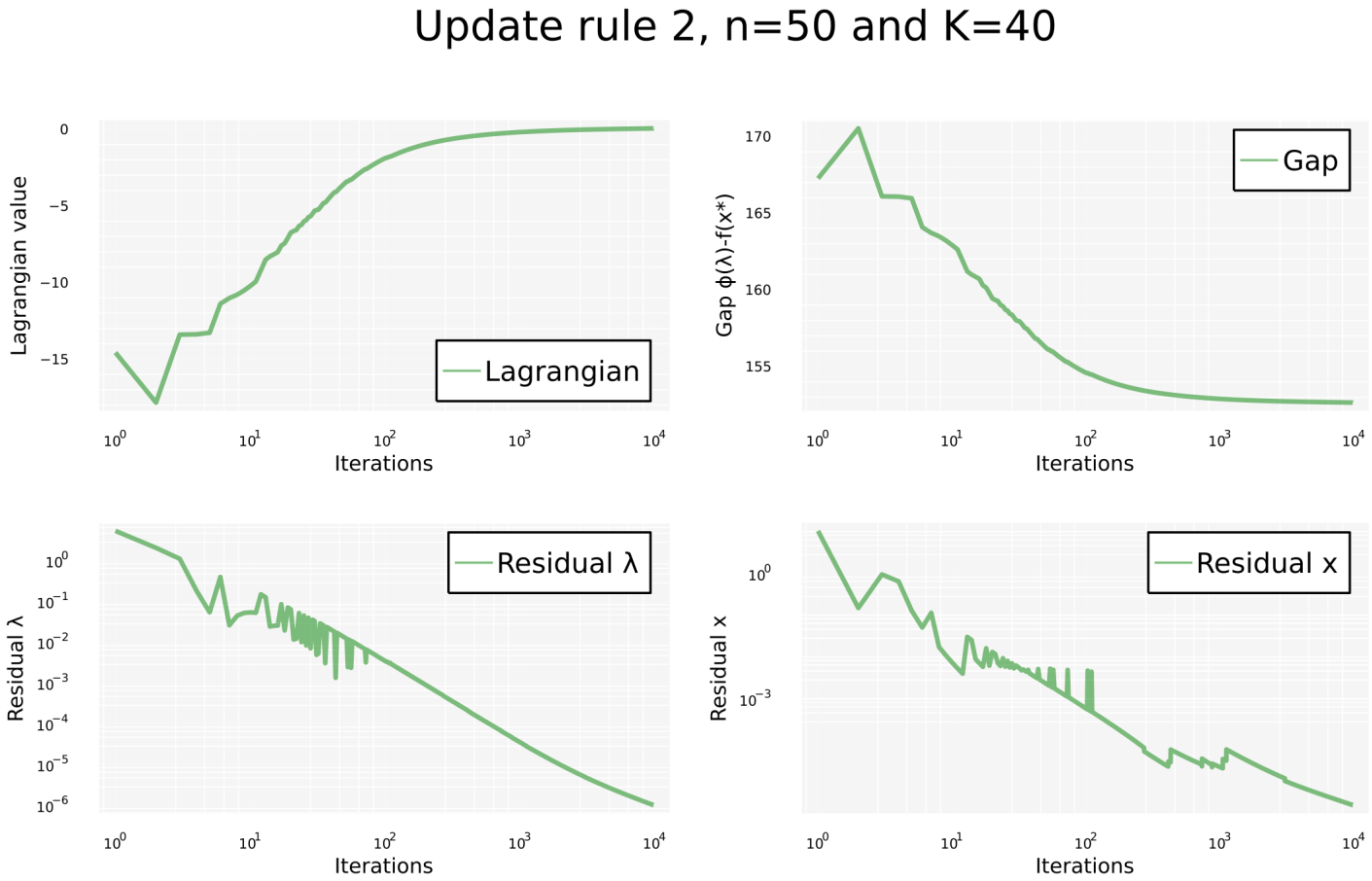

Figure 9: Update rule 2 without deflection

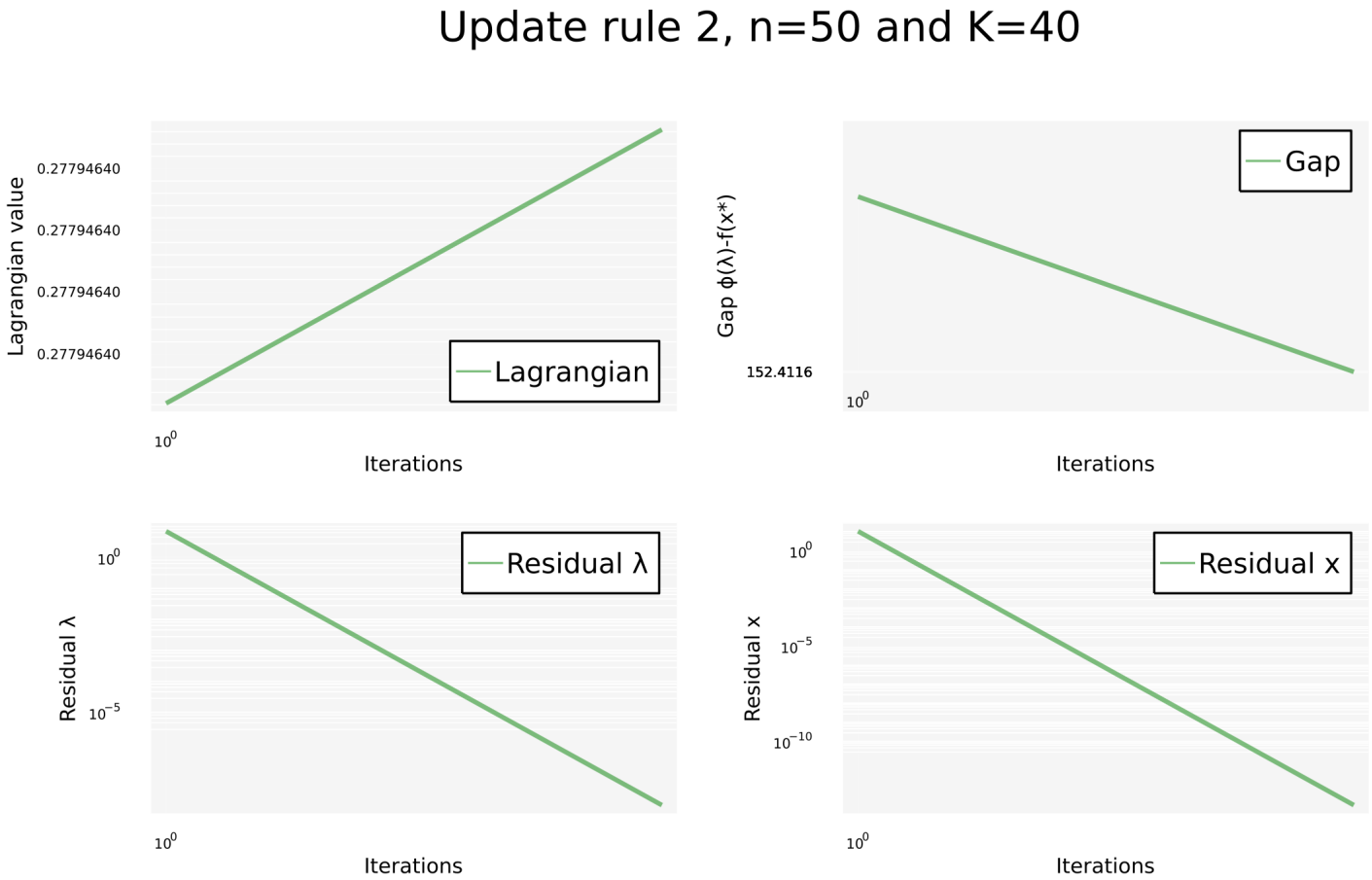

Figure 10: Update rule 2 with deflection

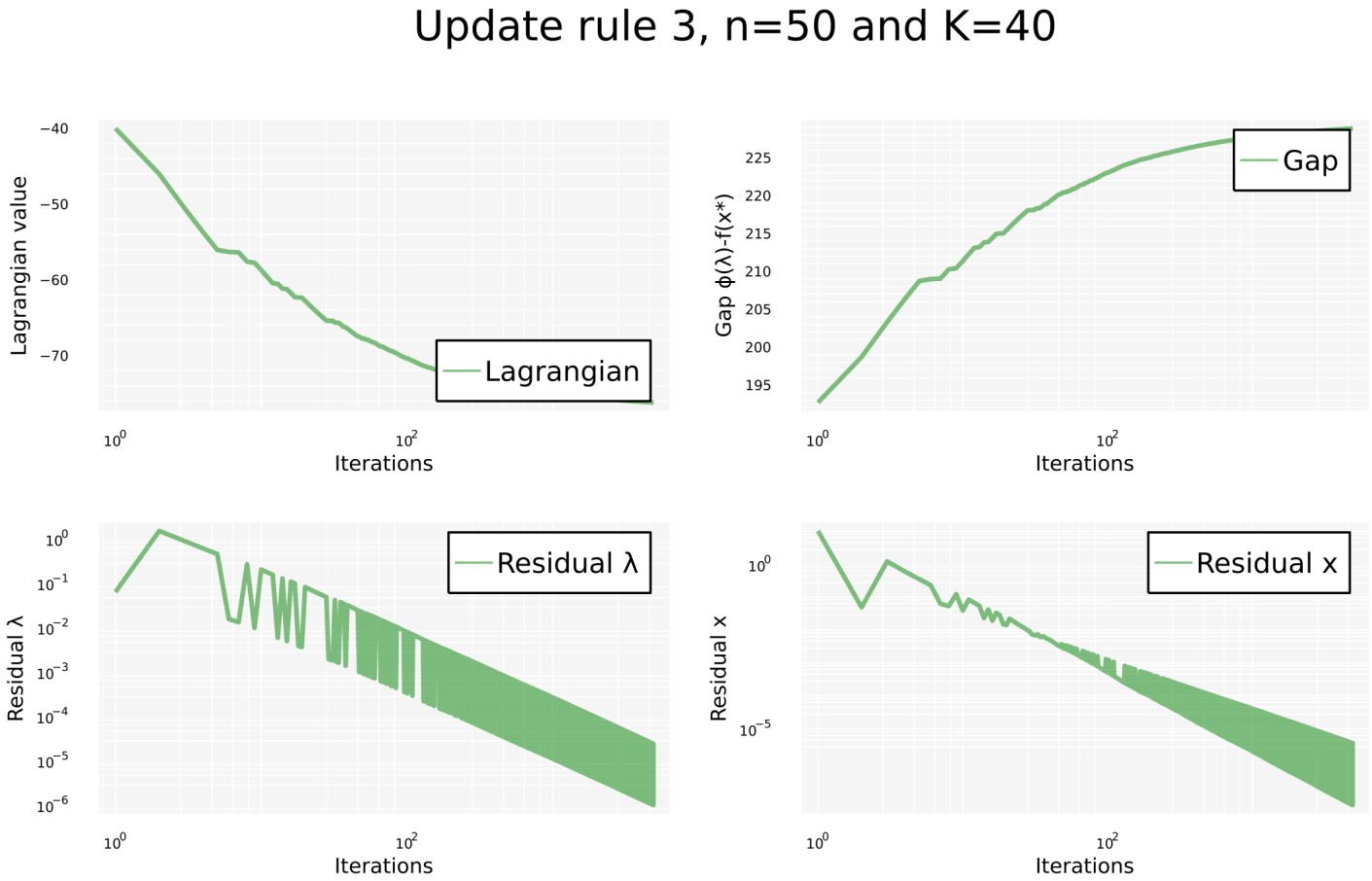

Figure 11: Update rule 3 without deflection

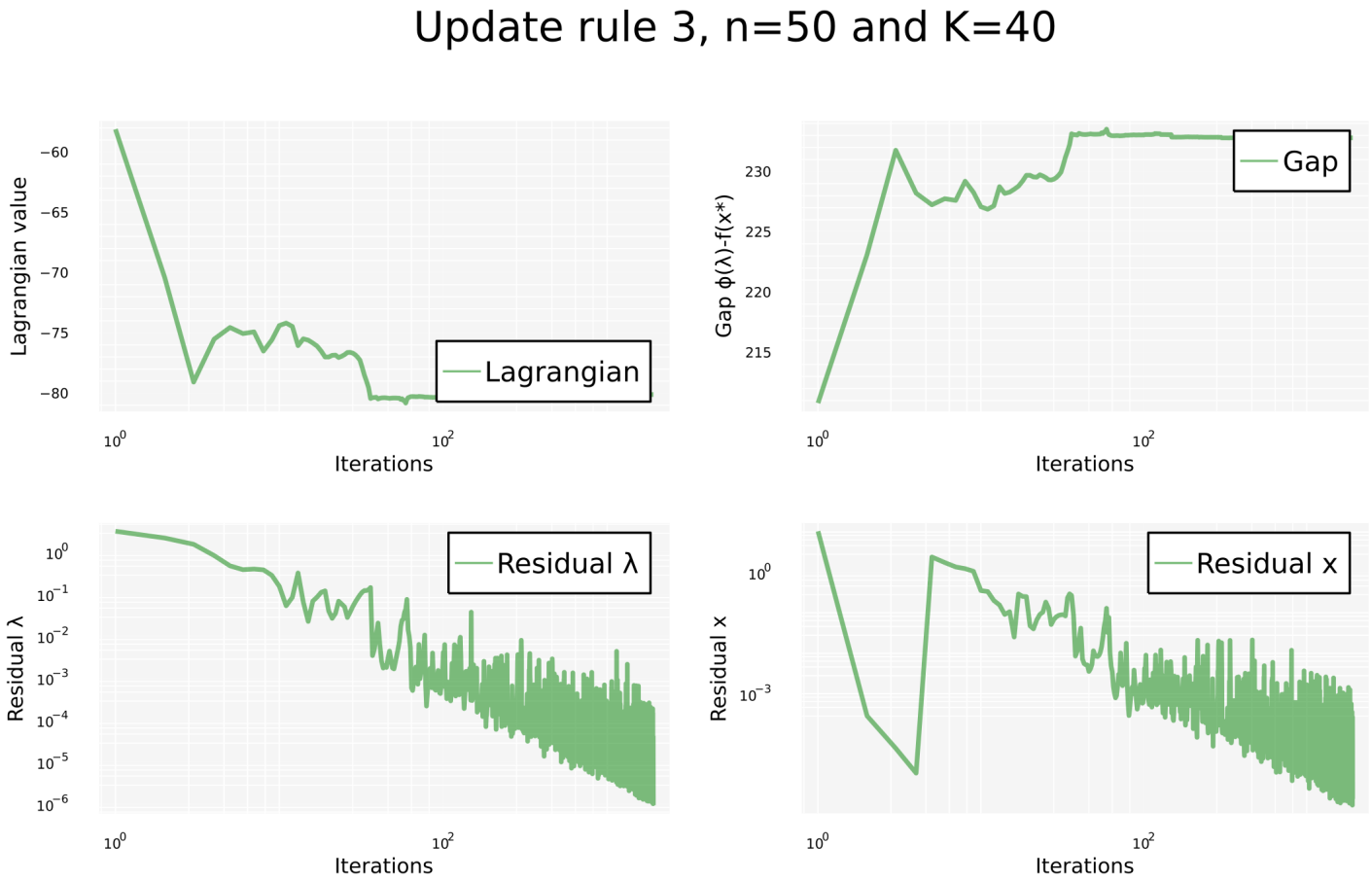

Figure 12: Update rule 3 with deflection

Q	*Update rule*	*defl*	*total iter*	*total time (sec)*	*Optimal*	*Best* $\phi(\lambda) - f(x^*)$	*Best iteration*	*Best* $\psi(\lambda)$
P.D.	1	*false*	7	0.0016	λ	27.54401	2	78.56617
		true	1807	0.7827	λ	39.80405	32	66.30613
	2	*false*	18057	54.86	λ	37.33086	18057	68.77932
		true	2915	1.8307	λ	35.85367	5	70.25651
	3	*false*	5560	6.3576	λ	96.32328	1	9.78689
		true	1983	0.827	λ	95.91524	1	10.19493
P.SD.	1	*false*	7024	10.1979	λ	151.9145	2	0.77504
		true	1624	1.0509	λ	155.3109	2	−2.62131
	2	*false*	9773	17.715	λ	152.63745	9773	0.05209
		true	2	0.0005	λ	152.4116	2	0.27795
	3	*false*	4908	4.7208	λ	192.6884	1	−39.99881
		true	1912	0.6989	λ	210.7933	1	−58.1038

Table 1: Execution with $n = 50$ and $K = 40$

K	*Update rule*	*defl*	*total iter*	*total time (sec)*	*Optimal*	*Best* $\phi(\lambda) - f(x^*)$	*Best iteration*	*Best* $\psi(\lambda)$
20	1	*false*	17407	207.56	λ	Not valid	0	Not valid
		true	3536	13.319	λ	18.93256	2	277.3673
	2	*false*	24302	396.62	λ	Not valid	0	Not valid
		true	3131	10.798	λ	109.6393	2	186.6606
	3	*false*	31914	893.67	λ	171.8724	3	124.4275
		true	3	0.021	λ	144.573	3	151.727
33	1	*false*	18860	265.94	λ	578.480	2	39.12439
		true	3060	11.336	λ	622.0355	11	−4.4311
	2	*false*	26241	373.783	λ	596.7733	4	20.8311
		true	3163	11.733	λ	Not valid	0	Not valid
	3	*false*	19547	204.111	λ	684.7382	1	−67.1338
		true	3	0.028	λ	674.0635	2	−56.4591
50	1	*false*	14139	107.0964	λ	4440.370	2	20.7995
		true	3499	13.169	λ	4458.561	1	2.60828
	2	*false*	17739	238.099	λ	Not valid	0	Not valid
		true	2	0.01	λ	4458.552	1	2.61751
	3	*false*	11662	78.604	λ	4558.096	1	−96.9265
		true	3	0.003	λ	4513.209	2	−52.0396
66	1	*false*	15227	125.117	λ	32272.27	4	20.2569
		true	3093	11.914	λ	32291.0	3	1.53456
	2	*false*	27690	396.278	λ	32293.9	27688	−1.36491
		true	11304	76.041	λ	32265.3	4	27.2090
	3	*false*	16347	129.622	λ	32365.5	1	−72.9762
		true	3307	12.543	λ	32387.6	1	−95.0997
80	1	*false*	4290	18.700	λ	34.4769	2	56.6695
		true	2429	8.105	λ	34.7176	81	56.4288
	2	*false*	2	0.012	λ	35.9508	1	55.1956
		true	3163	11.425	λ	35.9708	3163	55.1756
	3	*false*	4984	19.567	λ	195.0429	1	−103.896
		true	3	0.007	λ	194.9917	1	−103.845

Table 2: Execution with $n = 100$ and positive definite Q

Update rule	*defl*	*total iter*	*total time (sec)*	*Optimal*	*Best* $\phi(\lambda) - f(x^*)$	*Best iteration*	*Best* $\psi(\lambda)$
1	*false*	14077	22.044	λ	Not valid	0	Not valid
	true	2136	0.596	λ	49.0422	10	20.4259
2	*false*	14710	21.287	λ	45.1122	3	24.3559
	true	5981	4.258	λ	53.5208	67	15.9472
3	*false*	24833	55.003	λ	82.5786	1	−13.1105
	true	2177	0.470	λ	75.4582	1	−5.99017

Table 3: Execution with $n = 25$, $K = 13$ and positive semidefinite Q

We can notice from the above plots and tables that the guarantee of having an ascent direction is not ensured. Indeed in some cases we notice how the Lagrangian value trend tends to decrease instead of increasing, and as a consequence the dual gap behavior tends to increase, which is not what we want. That's why we always keep track of the best dual function value. Also, we see how in other cases, in particular when using deflection, the value tends to stabilize, reaching a local maxima. Regarding the λ/x norm, we see how the decreasing fashion towards the imposed threshold of 10^{-6} is not quite smooth in most cases. We noticed also how small values of n and good conditioning of the matrix Q results also in reaching a global maxima (and so having a stopping condition due to dual gap, with tries with $n = 2$ and $n = 5$). When increasing the size of the problem n, the algorithm tends to reach a local maxima and stay quite far from the dual gap (due to big magnitude of the subgradient).
In the vastly number of tries, the λ-norm is the stopping condition which terminates the execution. The threshold set on the dual gap works better on low magnitude experiments (as we just said), while the threshold of *max_iter* is never reached.

4 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 values 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 matrix used to solve the lagrangian relaxation;
  - `construct_A(`K`, `n`, `I_K`)`: return the constraint matrix A as described in the report;
  - `compute_lagrangian(`Q`,`q`,`x`,`λ`)`: compute the lagrangian function $x^TQx+q^Tx-\lambda^Tx$
- *JuMPSolution.jl*: module computing the off-the-shelf primal solution, containing a struct describing the problem parameters and a function to return the optimal value found
- *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;

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;

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] 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`.

[8] Iain Dunning, Joey Huchette, and Miles Lubin. Jump: A modeling language for mathematical optimization. *SIAM Review*, 59(2):295–320, 2017.

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

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

The term $y = Ax$ is given by:

$$y = \begin{bmatrix} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \\ \cdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n \end{bmatrix}$$

and then expanding the scalar product we obtain

$$\langle x, y \rangle = x_1 y_1 + x_2 y_2 + \cdots + x_n y_n$$

Now we can differentiate everything w.r.t. to x, obtaining the closed form

$$\frac{\partial \langle x, y \rangle}{\partial x} = \begin{bmatrix} \frac{\partial \langle x, y \rangle}{\partial x_1} \\ \frac{\partial \langle x, y \rangle}{\partial x_2} \\ \cdots \\ \frac{\partial \langle x, y \rangle}{\partial x_n} \end{bmatrix} = \begin{bmatrix} 2a_{11}x_1 + \sum_{j=1, j\neq 1}^{n} a_{1j}x_j + \sum_{i=1, i\neq 1}^{n} a_{i1}x_i \\ 2a_{22}x_2 + \sum_{j=1, j\neq 2}^{n} a_{2j}x_j + \sum_{i=1, i\neq 2}^{n} a_{i2}x_i \\ \cdots \\ 2a_{nn}x_n + \sum_{j=1, j\neq n}^{n} a_{nj}x_j + \sum_{i=1, i\neq n}^{n} a_{in}x_i \end{bmatrix}$$

A.2 Derivation of primal-dual update

After describing the derivation of the proximal term, we can now look into the detailed derivation of the update (3). 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, the summation terms obtained in the derivation of Ψ vanish:

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

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

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

A.3 Derivation of composite-mirror update

We follow the same approach also for the update (4). 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

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

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}).

C Appendix C: γ computation

Deflection is a way to change the direction of the computed subgradient by using a convex combination

$$d^i = \gamma^i g^i + (1 - \gamma^i) d^{i-1}$$

where g^i is the subgradient at iteration i and d^{i-1} is the direction computed at the previous step.
To find the best γ^i at iteration i, we could solve the problem

$$\gamma^i \in \arg\min \{\|\gamma g^i + (1 - \gamma) d^{i-1}\|^2 : \gamma \in [0, 1]\}$$

Hence we have to solve

$$\frac{\partial}{\partial \gamma} \|\gamma g^i + (1 - \gamma) d^{i-1}\|^2$$

which can be rewritten as

$$\frac{\partial}{\partial \gamma} \sum_{k=1}^{n} (\gamma_k g_k^i + (1 - \gamma_k) d_k^{i-1})^2 = 0$$

which result in the following vector

$$\begin{bmatrix} \frac{\partial}{\partial \gamma_1} \sum_{k=1}^{n} (\gamma_k g_k^i + (1-\gamma_k) d_k^{i-1})^2 \\ \frac{\partial}{\partial \gamma_2} \sum_{k=1}^{n} (\gamma_k g_k^i + (1-\gamma_k) d_k^{i-1})^2 \\ \vdots \\ \frac{\partial}{\partial \gamma_n} \sum_{k=1}^{n} (\gamma_k g_k^i + (1-\gamma_k) d_k^{i-1})^2 \end{bmatrix} = \mathbf{0} \implies \begin{bmatrix} \frac{\partial}{\partial \gamma_1} (\gamma_1 g_1^i + (1-\gamma_1) d_1^{i-1})^2 \\ \frac{\partial}{\partial \gamma_2} (\gamma_2 g_2^i + (1-\gamma_2) d_2^{i-1})^2 \\ \vdots \\ \frac{\partial}{\partial \gamma_n} (\gamma_n g_n^i + (1-\gamma_n) d_n^{i-1})^2 \end{bmatrix} = \mathbf{0} \implies$$

$$\begin{bmatrix} 2(\gamma_1 g_1^i + (1-\gamma_1) d_1^{i-1})(g_1^i - d_1^{i-1}) \\ 2(\gamma_2 g_2^i + (1-\gamma_2) d_2^{i-1})(g_2^i - d_2^{i-1}) \\ \vdots \\ 2(\gamma_n g_n^i + (1-\gamma_n) d_n^{i-1})(g_n^i - d_n^{i-1}) \end{bmatrix} = \mathbf{0}$$

Isolating each γ_i, we obtain the following compact value for γ

$$\gamma = \frac{(d^{i-1})^2 - g^i}{(g^i - d^{i-1})^2}$$