UNIVERSITÀ DI PISA

Optimization for Data Science

Project 41

Professor:
Antonio Frangioni

Alessandro Carella
(673380)

Erica Neri
(682010)

2024-2025

Contents

Chapter 1

Introduction

1.1 Problem description

Let's consider the problem (P) as follow:

$$(P) \quad \min_x \{\, x^T Q x \; + \; qx \; : \; Ex = b \,,\; 0 \leq x \leq u \,\}$$

Is a minimum cost flow problem with a quadratic, convex, separable objective function. The problem's objects are defined as follows:

- $x\,, q \;\in \mathbb{R}^m$ are the decisional variables that contributes linearly to the objective function. The single element $x_{i,j}$ represent the cost that we have by going from the i-th node to the j-th one.

- $b \;\in \mathbb{R}^n$ is a vector that represents the linear equality constraints. It ensures that the linear combination of decision variables x adheres to specific budget or flow relationships within the system.

- $u \;\in \mathbb{R}^m$ represents the upper bounds of the decision variables. This constraint ensures that the decision variables do not exceed specific values, which could be physical or resource constraints in the problem.

- $Q \;\in\; \mathbb{R}^{m \times m}$ is a diagonal matrix that is positive semidefinite, containing only non-negative eigenvalues.

- $E \;\in\; \mathbb{R}^{n \times m}$ is the node-arc incidence matrix of a directed connected graph with n nodes and m arcs. E is not a full-rank matrix, i.e., the rows in E are not linearly independent.

The flow conservation constraints are represented in vector form as $Ex = b$, where the incidence matrix E is constructed as follows:

$$E_{ij} = \begin{cases} -1 & \text{If arc } j \text{ exits node } i \\ 1 & \text{If arc } j \text{ enters node } i \\ 0 & \text{Otherwise} \end{cases}$$

The capacity constraint is represented by the one-box constraint, where x is limited by u (upper bound) and 0 (lower bound).

As we said before, it is presumed that each arc (i, j) of the network is associated with a unit transport cost, denoted x_{ij}. This means that for each unit of flow sent from a source to a destination, a cost is incurred equal to the sum of the unit costs associated with the individual arcs traversed.
The objective is to identify the minimum cost flow, which is defined as the set of arcs traversed with the lowest total cost. We search for a solution that minimizes the total cost while meeting the demand of all nodes in the network.

An application of the *min cost flow problem* can be to determine the most economic way to transport a certain amount of good (e.g. oil, oranges, cars ...) from one or more production facilities to one or more consumption facilities, through a given transportation network (e.g. a hydraulic network, a distribution network, a road network etc.).

In this report we examines an algorithm and a solver that address the problem.

1. A1 is a Lagrangian dual-based optimization algorithm of the dual method class, solved by a smoothed gradient method algorithm.
2. S2 is a solver that can be used for a variety of problems, provided that the problem is formulated appropriately.

1.2 Useful Definitions

Before delving into the details of our algorithms, let us provide some definitions that will be useful for our study.
Let's consider the problem defined as:

$$\min\{\, f(\,x\,) \;:\; x \in X \,\} \;\text{ where }\; X = \left\{\, x \in \mathbb{R}^n \;:\; g_i(\,x\,) \leq 0 \;\; i \in \mathcal{I}\,,\; h_j(\,x\,) = 0 \;\; j \in \mathcal{J} \,\right\}$$

$\mathcal{I}$ is the set of inequality constraints and $\mathcal{J}$ is the set of equality constraints.

Definition 1.2.1 (Convex function)**.** *f is convex if: $\forall x, y \in \mathbb{R}^n, \alpha \in [0,1]$:*

$$\alpha f(x) + (1-\alpha) f(y) \;\geq\; f(\alpha x + (1-\alpha) y)$$

Definition 1.2.2 (Lipschitz function)**.** *$f : \Omega \in \mathbb{R}^k \to \mathbb{R}^h$ is Lipschitz continuous if:*

$$\|f(x) - f(y)\| \;\leq\; L\|x - y\| \quad \forall\, x, y \;\in\; X, \;\; L \geq 0$$

(notation: $f \;\in\; C_L^{1,1}(\Omega)$)

Definition 1.2.3 (Lagrangian and Dual Function)**.** *The Lagrangian Function is defined as:*

$$L(\,x\,;\,\lambda\,,\,\mu\,) = f(\,x\,) + \sum_{i \in \mathcal{I}} \nu_i \, g_i(\,x\,) + \sum_{j \in \mathcal{J}} \lambda_j \, h_j(\,x\,)$$

where x is the variable, λ and ν are fixed parameters. In our analysis we will exploit the Lagrangian relaxation, in which will relax just a part of the constraints, not all of them. Starting from those function we can define:

$$\psi(\,\lambda\,,\,\mu\,) = \min_x \left\{\, L(\,x\,;\,\nu\,,\,\lambda\,) \;:\; x \in \mathbb{R}^n \,\right\}$$

and the dual problem as:

$$(D) \quad \max \; \psi(\nu, \lambda) \quad \nu \in \mathbb{R}_+^{|\mathcal{I}|},\, \lambda \in \mathbb{R}^{|\mathcal{J}|}$$

Chapter 2

Lagrangian dual-based algorithm

2.1 Hypotheses of the function

It is important to note that the linear combination of two convex functions remains convex. Therefore, the quadratic form $x^T Qx$ is convex because the matrix Q is positive semidefinite, and the function qx is also convex since it is linear. Furthermore, given that Q is positive semidefinite, the function is differentiable and has a non-negative second derivative within its domain.

As for the convexity of the constraints, it is clear that each constraint defines an affine hyperplane. The intersection of these hyperplanes forms an affine subspace, which is also convex.

2.2 Optimization scheme

Let's start by computing the relaxed Lagrangian function of our problem:

$$(P(\lambda, x)) \quad \min_x \{\ x^T Qx + qx + \lambda(Ex - b)\ :\ 0\ \leq\ x\ \leq\ u\ \}$$

This approach is advantageous due to the fact that the function is separable, meaning that it does not bind the variables together. Additionally, the remaining constraints are also separable. In this manner, the resulting function can be rewritten as a sum of functions of fewer variables. Furthermore, the resulting maximum problem decomposes into smaller sub-problems, which can also be parallelized.

Given our previous observations, and considering that the Lagrangian is composed of convex and linear functions, we can indeed conclude that it is also convex. The problem we want to face is now defined as the Dual problem:

$$(D) \quad \max_\lambda \big\{v(P(\lambda, x))\big\}\ =\ \max_\lambda\ \Big\{\ \min_{0\,\leq\,x\,\leq\,u}\ \{\ x^T Qx + qx + \lambda(Ex - b)\ \}\Big\}$$

Observation 1. *Since our problem is convex with affine constraints, we can conclude that the optimal value of the two formulations coincides, i.e., $v(D) = v(P)$. The proof can be found here [2] and here [1].*

Smoothed Lagrangian

It is crucial to address the issue that arises from the presence of a non-invertible component, namely x^TQx, which is a consequence of the zeros that appear on the diagonal of Q. To address this, we need to introduce a smoothing term that will make the function invertible, allowing us to compute the optimal value, namely x^*. The problem become:

$$(P_\mu(\lambda, x)) \quad \min_x \{ \ x^TQx + qx + \lambda(Ex - b) + \mu||x||_2^2 \ : \ 0 \ \leq \ x \ \leq \ u \ \}$$

Thanks to this new term we are able to smooth the non invertible points and get to the solution. For a more detailed examination of the selection of μ, please refer to paragraph 2.2.1.

Let's take into consideration the following theorem, that will be useful later:

Theorem 1. *The function thus constructed is well defined and continuously differentiable at any $0 \leq x \leq u$. Moreover, this function is convex and its gradient is Lipschitz continuous.*

The proof can be found here [6].

Since the problem is separable, we can rewrite it in the following way:

$$(P_\mu(\lambda, x)) \quad \min_x \{ \sum_{i,j}(Q_{i,j}+\mu)x_{i,j}^2+\sum_{i,j} q_{i,j}x_{i,j}+\sum_{i,j} x_{i,j}(-\lambda_i+\lambda_j) \ : \ 0 \leq x_{i,j} \leq u_{i,j} \ \}-\lambda b =$$

$$= \min_x \{ \sum_{i,j}((Q_{i,j} + \mu)x_{i,j} + q_{i,j} - \lambda_i + \lambda_j)x_{i,j} \ : \ 0 \leq x_{i,j} \leq u_{i,j} \ \} - \lambda b$$

It should be noted that, despite the presence of two distinct indexes, i and j, the underlying issue is a sum of one-dimensional elements. This is due to the fact that the indexes are designed to indicate which two nodes are under consideration at each iteration.

Once λ is fixed, the term $-\lambda b$ is a constant, so it is not necessary to analyze the problem with that term; the result will simply be shifted upward or downward by that value.
Now, each of these problems can be readily resolved by computing the gradient method, but first we need to find the optimal point.

The Optimal Point

Let's focus on a single element of our problem to underline the procedure. For simplicity, we'll define the function $f(i, j)$ as:

$$((Q_{i,j} + \mu)x_{i,j} + q_{i,j} + -\lambda_i + \lambda_j)x_{i,j}$$

We want to find it's unconstrained stationary point, so we start by computing it's derivative:

$$\frac{\partial(f(i,j))}{\partial x_{i,j}} \ = \ 2((Q_{i,j} + \mu)x_{i,j} + q_{i,j} - \lambda_i + \lambda_j$$

By denoting $k = q_{i,j} - \lambda_i + \lambda_j$ and setting the result equal to zero, we obtain:

$$\frac{\partial(f(i,j))}{\partial x_{i,j}} \;=\; 2(Q_{i,j}+\mu)x_{i,j} + k = 0$$

The optimal solution is reached when:

$$x_{i,j} \;=\; \frac{-k}{2(Q_{i,j}+\mu)} \;=\; \frac{-(q_{i,j}-\lambda_i+\lambda_j)}{2(Q_{i,j}+\mu)}.$$

Now, we want to project this value considering the constraints $0 \le x_{i,j} \le u_{i,j}$. We have three different cases:

$$x^*_{i,j} = \begin{cases} 0 & \text{if } x_{i,j} < 0 \\ u_{i,j} & \text{if } x_{i,j} > u_{i,j} \\ \frac{-(q_{i,j}-\lambda_i+\lambda_j)}{2(Q_{i,j}+\mu)} & \text{if } 0 \le x_{i,j} \le u_{i,j} \end{cases}$$

Alternatively, we can see the optimal point as:

$$x^*_{i,j} = \max\left(0, \min\left(u_{i,j}, \frac{-(q_{i,j}-\lambda_i+\lambda_j)}{2(Q_{i,j}+\mu)}\right)\right) \tag{2.1}$$

Substituting this values into $f(i,j)$ we obtain:

$$\begin{cases} 0 & if\ x_{i,j} \le 0 \\ ((Q_{i,j}+\mu)u_{i,j} + q_{i,j} - \lambda_i + \lambda_j)u_{i,j} & if\ x_{i,j} \ge u_{i,j} \\ \frac{-(q_{i,j}-\lambda_i+\lambda_j)^2}{4(Q_{i,j}+\mu)} & if\ 0 \le x_{i,j} \le u_{i,j} \end{cases}$$

This is the Lagrangian function restricted to the nodes i, j and calculated in the optimal point, only dependent from λ and μ.
The Lagrangian function will be now defined as follow

$$\varphi(\lambda) = \begin{cases} -\lambda b & if\ x_{i,j} \le 0 \\ -\lambda b + \sum_{i,j}((Q_{i,j}+\mu)u_{i,j} + q_{i,j} - \lambda_i + \lambda_j)u_{i,j} & if\ x_{i,j} \ge u_{i,j} \\ -\lambda b + \sum_{i,j} \frac{-(q_{i,j}-\lambda_i+\lambda_j)^2}{4(Q_{i,j}+\mu)} & if\ 0 \le x_{i,j} \le u_{i,j} \end{cases}$$

We now want to solve the linked dual problem:

$$(D) \quad \max_{\lambda}\ \varphi(\lambda)$$

To apply the gradient method, we need to calculate the gradient of the function $\varphi(\lambda)$ with respect to the dual variable λ. It's immediate to see that

$$\nabla_\lambda \varphi(\lambda) = Ex^* - b$$

and thanks to theorem 1 we know that is Lipschitz continuous.
This element, $Ex^* - b$, represents the violation of the conservation laws. More specifically:

- When $Ex^* = b$, the flow conditions are satisfied and the gradient is zero.
- If $Ex^* \neq b$, the value of the gradient indicates by how much the flows do not satisfy the conservation constraints.

This gradient expression has a clear meaning: given that E is the incidence matrix that defines the relationships between nodes and arcs in the graph (the structure of the flow problem), x^* represent the flows on the arcs and b is the vector of supply/demand constraints at the nodes, we can see that, when Ex^* is calculated, the total flow budget at each node is obtained:

- $(Ex^*)_i = b_i$ then the flow at node i exactly satisfies the supply/demand.
- $(Ex^*)_i \geq b_i$, there is excess flow in the node i (more flux entering than leaving).
- $(Ex^*)_i \leq b_i$ there is a flow deficit at node i (more flow goes out than comes in).

2.2.1 The choice of μ

In light of the work conducted by Y. Nesterov [6], it can be established that the value of μ is a positive, fixed value, and its selection is of paramount importance. If μ is insufficiently large, the function remains close to being non-differentiable, which can result in numerical instabilities. Conversely, if the value of μ is excessively large, the process of smoothing will result in a significant alteration of the original problem, which may lead to the generation of suboptimal solutions. It is essential to find a compromise between accuracy and stability.

For simplicity we call our smoothed function f_μ and f the initial function. We can easily observe that as soon as $\mu \to 0$, $f_\mu \to f$ as well. In particular, we have the following sequence of inequalities:

$$f_\mu \, - \, \mu K \; \leq \; f \; \leq \; f_\mu$$

where $K = \max_x \{ \, ||x||_2^2 \, : \, 0 \leq x \leq u \, \}$. This means that, for $\mu \to 0$, $argmin \, \{ \, f_\mu(\, x \,) \, \} \to x_*$, the optimal point of f.

It is evident that the decomposition of our problem yields the result that $K_i = |u_i|^2$. Consequently, for the primary problem, we have that $K = ||u||_2^2$. The choice of μ is now defined as:

$$\mu = \frac{\varepsilon}{2K}$$

where ε is the chosen tolerance.

It should be noted that the convergence of the gradient method is contingent upon the Lipschitz constant of the gradient and the L-smoothness of the function. The crux of the matter is that as μ approaches zero, the function is assumed to become increasingly smooth, yet simultaneously exhibit a decline in L-constancy. It can be demonstrated that the Lipschitz constant will exhibit an inverse relationship with μ, exhibiting growth as μ decreases.
In particular, we can say that

$$L = \; \frac{||E||^2}{\mu} \; = \; \frac{2||E||^2 K}{\varepsilon}$$

The advantage is that f_μ is now smoothed, and consequently, the fast gradient method can be employed to solve $f(x)$, which will have a *sublinear convergence* magnitude.
More about this can be found here [6] and here [4] .

2.3 Nesterov's Fast Gradient Method

The gradient method involves computing the partial derivatives of the dual function $\varphi(\lambda)$, with respect to the dual variable λ, and iteratively updating them in the direction of the gradient. In this way we will arrive at the optimal solution. The dual update at each iteration is usually defined as:

$$\lambda_{k+1} = \lambda_k + \alpha_k \nabla_\lambda \varphi(\lambda_k)$$

where k is the iteration and α_k is the learning rate.

In our study, we will use the *fast gradient method*, an accelerated variant of the standard gradient method. The key difference lies in the fact that the gradient is computed at a point y , which is an extrapolation of λ_k in the direction of $\lambda_k - \lambda_{k-1}$. This method introduces additional sequences of variables: the iterates λ_k and the auxiliary sequences y_k and v_k, where k denotes the k -th iteration. The general algorithm works as follows:

Algorithm 1 Fast Gradient Descent

Initialize: $\lambda_0 \in \mathbb{R}^n$, $\theta_0 = 1$, $v_0 = \lambda_0$
for $k \geq 1$ **do**
    $y_k = (1 - \theta_k)\,\lambda_k + \theta_k\, v_k$
    $\lambda_{k+1} = y_k - t\,\nabla_\lambda \varphi(y_k)$
    $v_{k+1} = \lambda_k + \frac{1}{\theta_k}(\lambda_{k+1} - \lambda_k)$
    if $\|\nabla_\lambda \varphi(y_k)\| < \varepsilon$ **then**
        break
    end if
end for

Where t is fixed in the interval $(0, \frac{1}{L}]$, L is the Lipschitz constant of the gradient, and $\theta_k = \frac{2}{k+2}$. More information about this algorithm can be found here [3].

2.4 The convergence

By Theorem 1, we know that the resulting function $\varphi(\lambda)$, is convex and its gradient is Lipschitz continuous. Given the specific choices of t and θ_k in the Algorithm 1, we can deduce the convergence of the defined algorithm by the following theorem:

Theorem 2. *Let f be convex with L-Lipschitz continuous gradient. The iterations of 1 with constant step size $t \in (0, 1/L]$ and with $\theta_k = \frac{2}{2+k}$ satisfy*

$$f(\lambda_k) - f^* \leq \frac{2}{(k+1)^2\, t} ||\lambda_0 - \lambda^*||_2^2$$

for all $k \geq 1$.

The proof can be found here [3].

Chapter 3

Experiments

To commence, let us briefly review the nature of the problem under consideration. The optimization problem in this study involves the resolution of a network flow problem through the utilization of a quadratic cost function. Given a directed graph with a node-arc incidence matrix E, an arc capacity vector u, a supply vector b, and a quadratic cost matrix Q, the objective is to ascertain the optimal flow allocation that minimizes the Lagrangian dual function.

Mathematically, the problem is expressed as:

$$\min_x \{ \frac{1}{2} x^T Q x + q^T x \} \quad \text{subject to } Ex = b, \quad 0 \leq x \leq u.$$

To solve this problem, we will ll use the Nesterov's Fast Gradient Method and some variations.

3.1 Instances generation

Before presenting the algorithms employed, it is necessary to outline the process by which the problem instances were generated.

To generate meaningful instances of the minimum cost flow problem, reference was made here. Specifically, the experiments used randomly generated convex quadratic cost network design problems, constructed using the *netgen* generator to create the "base" (linear) instance, to which quadratic costs were subsequently added.

In this setup, from the .dmx file, values for q, u, and b were obtained, along with the graph characteristics, the number of arcs, and the number of nodes (i.e., m and n). Additionally, the node-arc incidence matrix E was reconstructed. The data was extracted from the files using custom functions.

Regarding the quadratic costs, each element Q_{ij} of the matrix Q was randomly generated within an interval centered on q_{ij}/u_{ij}, permitting only non-negative values. Furthermore, the width of this interval was modulated by a parameter α, allowing for the generation of larger or smaller quadratic costs depending on the chosen value.

Additionally, since the issue of positive semidefinite Q-matrices was considered, an extra parameter, ρ, was introduced. This parameter specifies the percentage of arcs in

which the quadratic cost is randomly set to zero within the newly created matrix. This approach enables the generation of quadratic cost matrices with varying dimensionalities based on α, while also adjusting the percentage of zeros according to ρ.

3.2 General Structure of the Code

The codebase is organized into multiple Jupyter notebooks, each corresponding to a specific variant of the developed methods:

- `fastGradientMethod.ipynb`: Contains the standard implementation of the fast gradient method.
- `fastGradientMethodWithOpt.ipynb`: Uses the optimal value of the objective function to dynamically choose the parameter μ at each iteration.
- `fastGradientMethodWithoutOpt.ipynb`: Implements a variant that dynamically chooses μ at each iteration, without using the optimal value.

In addition, there are two supplementary notebooks:

- `plots.ipynb`: Generates the relevant plots used for analysis and presentation.
- `solver.ipynb`: Solves the test instances using the `SCS` solver.

Three utility scripts support the main notebooks:

- `getProblemData.py`: Reads the DMX instance files and processes the problem data.
- `plots.py`: Defines plotting functions used in `plots.ipynb`.
- `gradientUtils.py`: Contains shared utility functions for all three fast gradient method variants.

3.2.1 Fast Gradient Method with Constant Step Size

The method iterates as described in the second Chapter, specifically in the section Nesterov's Fast Gradient Method 2.3. We used a constant step size and the following hyperparameters:

- $\epsilon = 10^{-3}$: tolerance for stopping criterion.
- $\alpha = 100$: Scaling factor for the generation of matrix Q.
- $\rho = 0.3$: Proportion of zero diagonal elements in Q.
- $\beta = 0.5$: Acceleration parameter.
- μ: Regularization parameter, computed as $\mu = \frac{\epsilon}{2||u||_2^2}$.
- L (Lipschitz constant): Computed as $L = \frac{||E||_2^2}{\mu}$.

- t (Step size): Fixed and defined as $t = \frac{1}{L+1e-5}$.
- θ_k: Iteration update factor, defined as $\theta_{k+1} = \frac{2}{k+2}$.
- $max\_iter = 300\ 000$: Maximum number of iterations allowed.

We opted to devise a variety of graphs to assess the efficacy of our algorithm. Specifically, we developed a graph to compare the theoretical and observed convergence, thereby facilitating a comparison between the anticipated and actual behavior. This graph was constructed on a logarithmic scale to enhance visual clarity, and the theoretical convergence with a fixed step size was calculated as follows:

$$\frac{2}{(k+1)^2\, t}||\lambda_0 - \lambda^*||_2^2$$

as specified in Theorem 2.

In order to facilitate the monitoring of progress, three additional plots were developed. The first plot is intended to demonstrate the decline in gradient norm with successive iterations. The second plot is designed to illustrate the variation in the value of the Lagrangian function over time. The third plot is employed to assess the relative gap between the optimal value obtained through the solver and the values of φ, i.e., values determined by the following formula:

$$\frac{f_{opt} - \varphi}{|f_{opt}|}$$

Results

The initial implementation of the fast gradient method was tested on a small "test" matrix, and the following was considered:

$$Q = \begin{bmatrix} 94.4171 & 0 & 0 & 0 \\ 0 & 40.5792 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 82.6752 \end{bmatrix} \qquad E = \begin{bmatrix} 1 & 0 & 0 & 1 \\ -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & -1 \end{bmatrix}$$

$$q = (3, 1, 1, 2) \quad u = (2, 2, 2, 2) \quad b = (3, 0, 0, 3)^T$$

For this first matrix, whose results are shown in Figure 3.1, the method successfully converges after approximately 28,784 iterations.

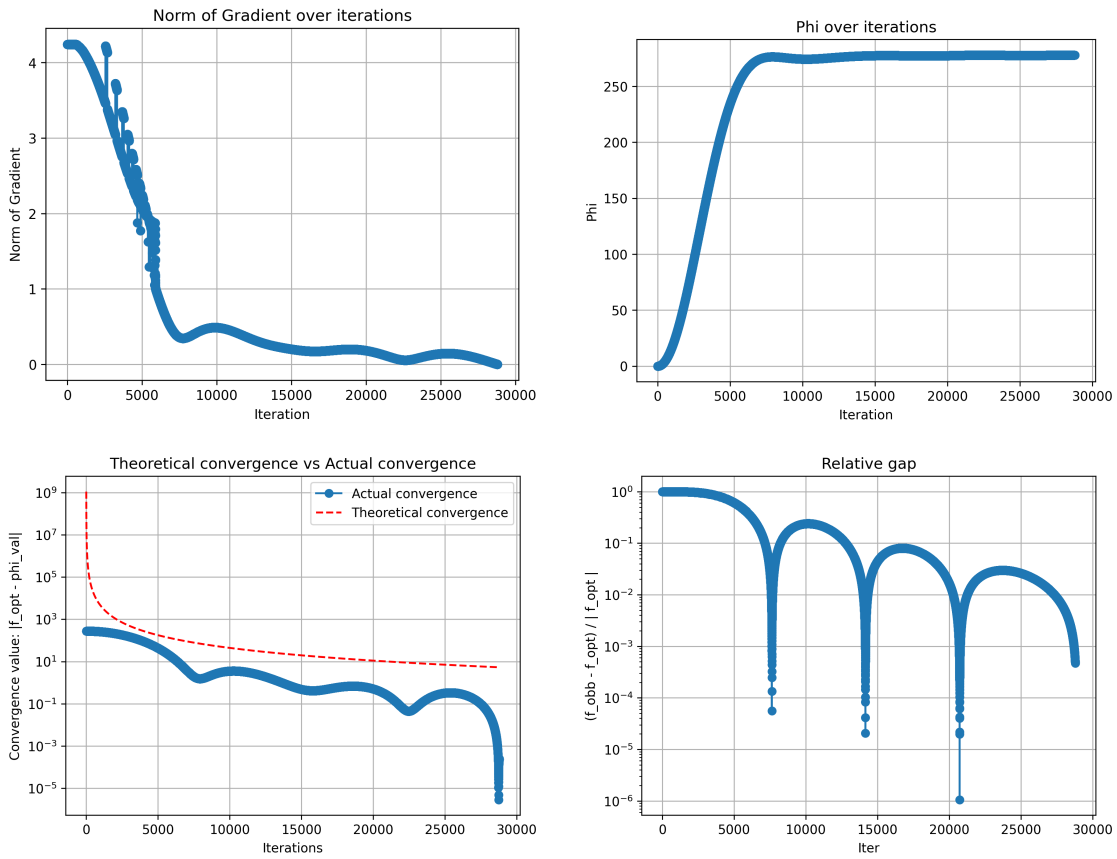

Figure 3.1: Results on the small test matrix

The findings of this study can seem promising. The value obtained by the FG method is nearly equivalent to the value found by the solver, which is approximately 277,9192. Additionally, the norm of the gradient is $9,9 \times 10^{-4}$. However, a substantial amount of iteration was necessary to reach this value for a relatively small matrix.

As suspected, for the larger matrix, the method does not converge. Despite the discernible initial movement in the φ function, it was observed that 300.000 iterations proved insufficient to elicit a perceptible change in the gradient norm. The corresponding results are displayed in Figure 3.2.

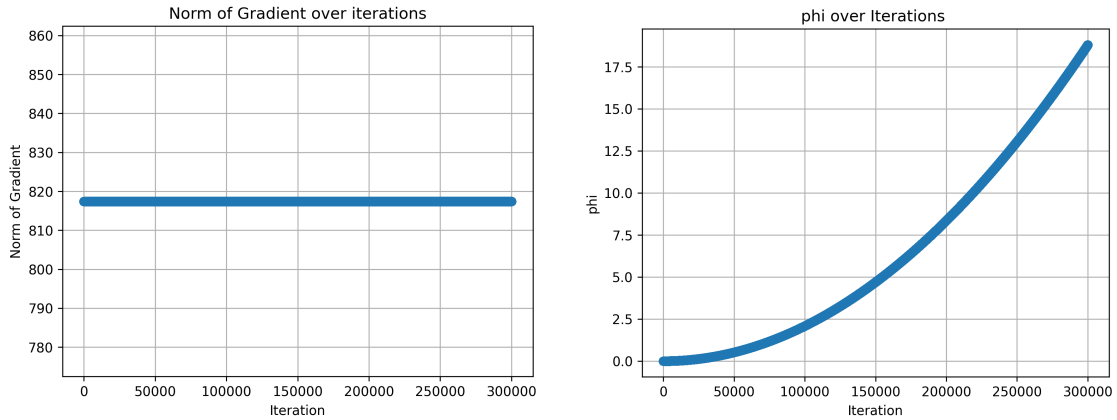

Figure 3.2: Results on the decently big test matrix

The final values obtained from this analysis yield a gradient norm of $817,4154$ and a smoothed lagrangian function of $18,8057$. The solver result, on the other hand, yields a value of $373214,4170$.

This phenomenon can be attributed to the fact that, given the size of the matrix, the obtained value of the μ parameter reaches an order of magnitude of 10^{-14}. As discussed in section 2.2.1, as μ approaches zero, the function becomes increasingly smooth, but concurrently, there is a decline in L-constancy. Furthermore, it is evident that the smaller the μ is, the more diminutive the stepsizes will be.

For this reason, an attempt was made to implement an alternative version of the fast gradient method.

3.3 Fast Gradient Method with dynamic smoothness parameter

In the following versions, the same hyperparameter previously defined are used, with the exception of the μ parameter, which undergoes adjustment at each iteration. Consequently, the stepsize undergoes modification as well.
It should be noted that the same plots are employed in this case as well, with the requisite modifications.

3.3.1 Implementation with the usage of the optimal value

As demonstrated by Frangioni et al. ([5]), the alteration of the mu parameter has the potential to enhance the efficacy of the initial phase of the algorithm, thereby facilitating a better convergence.

A significant challenge associated with this approach pertains to the necessity of establishing an upper bound on the optimal value of the function. In this particular instance, however, the exact solution was available through the utilization of a solver.

Consequently, the following methodology was employed to calculate μ during each iteration:

$$\mu \;=\; \frac{\max\{\, f^* - \varphi_k \,,\, \varepsilon\,|f^*|\,\}}{2K}$$

Where f^* is the optimal value, φ_k is the value of the Smoothed Lagrangian function at the k-th iteration and K is equal to $||u||_2^2$, as assessed in Section 2.2.1.

This variation resulted in a substantial enhancement of the outcomes.

In Figure 3.3 are presented the result for the small matrix. In this case, the method successfully converges after approximately 1.676 iterations, yielding consistent results in relation to the initial predictions.

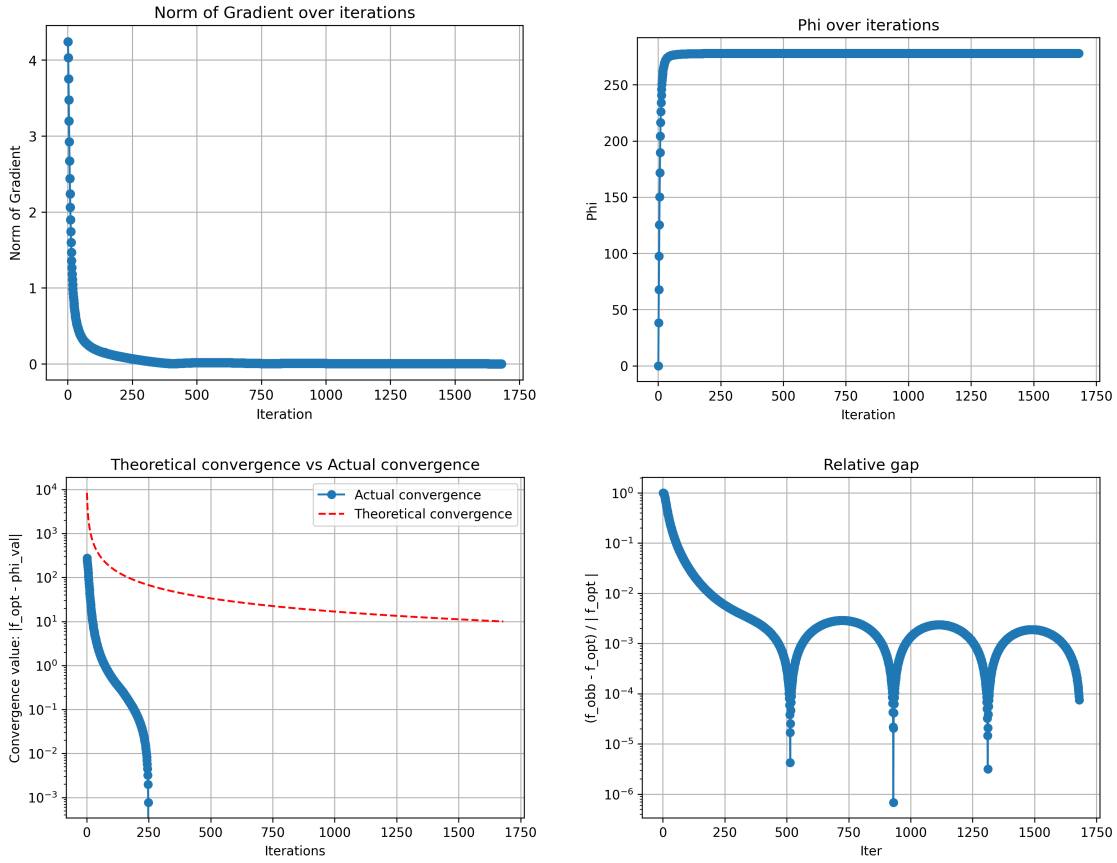

Figure 3.3: Results on the small test matrix with a dynamic μ

We then proceeded to assess the functionality of the newly developed algorithm within the larger matrix framework. The result is presented in the reference Figure 3.4. The superiority of the results obtained with the present algorithm, in comparison to those yielded by the previous one, is unmistakable. However, it must be acknowledged that the process is not without its imperfections. The culmination of 300.000 iterations of the gradient norm yields an approximate value of $4,6038$. The optimal value identified approaches the desired outcome, yet it does not fully align with the expected result.

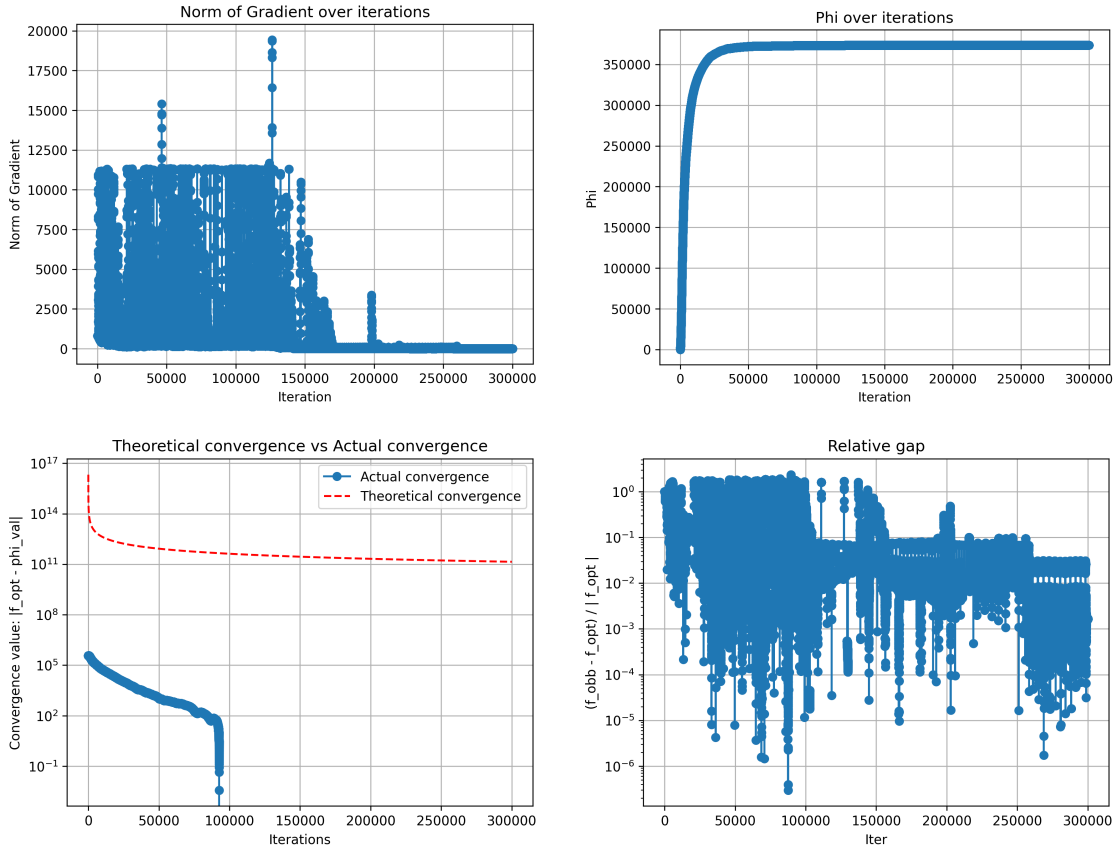

Figure 3.4: Results on the big test matrix with a dynamic μ

It is important to acknowledge that, given the algorithm's inability to attain the optimal solution, we decided to consider the theoretical convergence rate with respect to the solver estimate of x_{opt}. This was used to get an estimate of how our algorithm is performing compared to the theoretical convergence.

We also notice that, from equation 2.2, there is a linear dependence between x_{opt} and λ. Therefore, the final value of x provided by the solver was utilized instead of the last value of λ found by the method to plot the theoretical convergence.

3.3.2 Implementation without the usage of the solver

Building on the approach outlined in the previous section, we developed an algorithm that does not depend on the optimal value of the objective function.

In particular, we employed a general strategy to progressively reduce the value of μ over the course of the iterations. Specifically, we defined:

$$\text{big\_}\mu(k) = 10^{-\left\lfloor \frac{k}{1000} \right\rfloor}$$

where $k \in \mathbb{N}$ denotes the iteration index. At each iteration, we set:

$$\mu_k = \max\left\{\text{big\_}\mu(k), \mu(k)\right\}$$

where $\mu(k)$ is the theoretically derived value defined in Section 2.2.1.

The preliminary results are encouraging. Figure 3.5 illustrates the outcome obtained for the largest matrix considered in the previous examples. Following 300.000 iterations, a value of $\varphi = 373201,7825$ was ascertained, which is in close agreement with the value determined by the solver, i.e., $373214,4171$.

Additional results and a more comprehensive discussion are provided in the Conclusions 5.

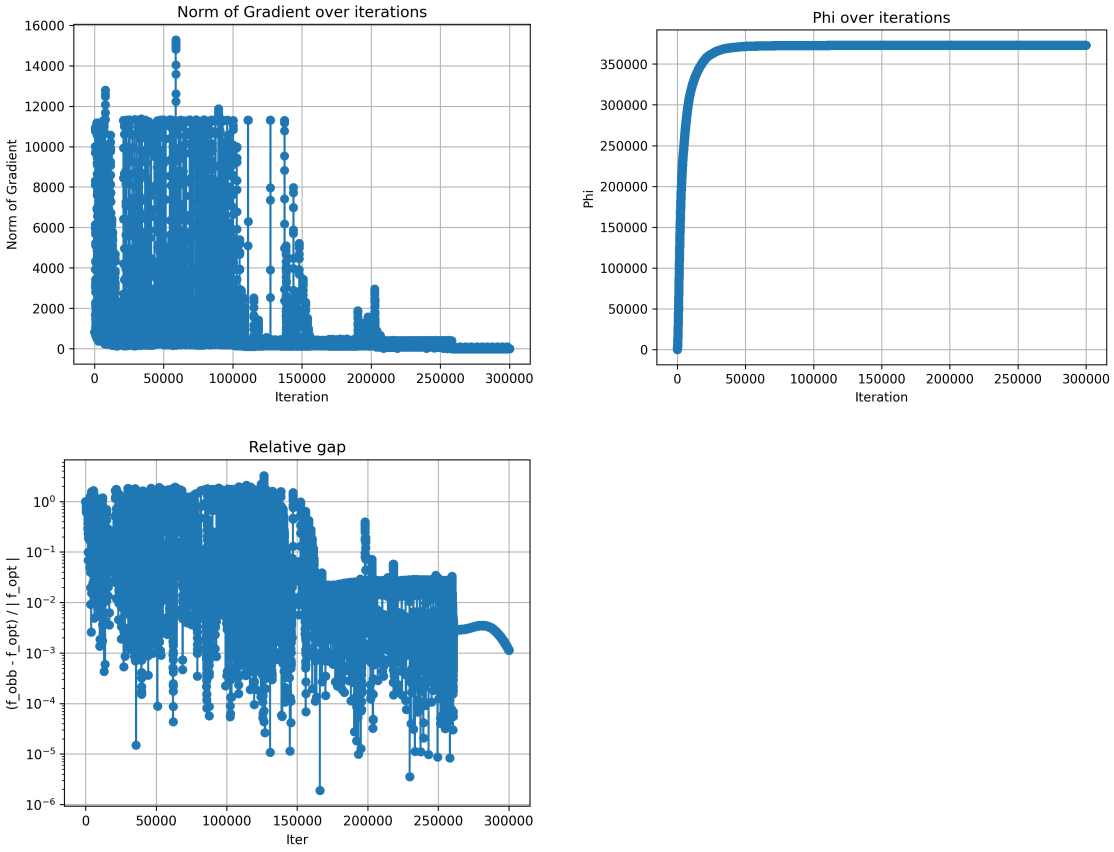

Figure 3.5: Results on the big test matrix with a dynamic μ

Chapter 4

Solver

4.1 CVXPY Solver

For solving the quadratic minimum cost flow problem, we used the Python optimization modeling library **CVXPY**.

`CVXPY` is an open-source Python library for defining and solving convex optimization problems. It is a domain-specific language that allows users to express problems in a high-level, mathematical syntax while relying on efficient underlying solvers. It provides a convenient interface to multiple solvers and supports quadratic objectives, making it an appropriate choice for our problem.

In our implementation, CVXPY is used to define the decision variables, quadratic objective function, and linear constraints corresponding to flow conservation and capacity limits. The optimization problem is then solved using one of the supported solvers, in our case `SCS` (Splitting Conic Solver).

`SCS` is a numerical optimization solver designed to efficiently solve large-scale convex cone problems. It offers direct and indirect solution methods. The direct method is often faster when sufficient memory is available; however, the indirect method is more memory-efficient and is preferred for extremely large-scale problems. In our case, we used the settings with the indirect method.

Problem Formulation

`SCS` solves problems in the following conic form:

$$\min_{x} \quad c^T x$$

subject to:

$$Ax + s = b$$

with:

$$s \in \mathcal{K}$$

where $\mathcal{K}$ is a Cartesian product of convex cones.

`CVXPY` automatically converts high-level problem descriptions into standard forms compatible with solvers like `SCS`.

Problem Integration and Execution

The function `solver()` in our Python code takes a problem file as input, reads the instance data using the `getProblemData()` utility, and sets up the optimization model in CVXPY. The function formulates the objective as a quadratic form:

$$\min_{x} \quad x^T Q x + q^T x$$

subject to:

$$Ex = b, \quad 0 \leq x \leq u$$

where x, Q, q, E, b and u are defined as specified in the Introduction 1.1.

Chapter 5

Conclusions

In this section, we present the concluding results obtained for algorithms that utilize a dynamic smoothness parameter. Due to the complexity inherent in the convergence of the classic FG method, we opted to abstain from further analysis.

A decision was made to undertake a comparative analysis of the algorithms implemented with the solution of the CVXPY solver. In particular, we compare the results of the FG method with a dynamic smoothness parameter with the optimal solution, without the optimal solution, and the results of the solver, in which the 'SCS' method was selected for the choice of the solving algorithm.

It needs to be be noted that, for the construction of the instances, the results are obtained by considering $\alpha = 100$ and $\rho = 0{,}3$ for each row of the table 5.1. In addition, the parameters are held constant at a value: $\varepsilon = 10^{-3}$, and the maximum number of iterations is set at 150.000 for both FG methods. With regard to CVXPY, it should be noted that the default setting for the parameter ε is 1e-4, as specified by the manual.

It is imperative to acknowledge that, in light of the time complexity inherent in the methods, a decision was made to select the smaller instances for the purpose of conducting this comparison. This is also the rationale behind our decision to use 150.000 iterations for this ultimate comparison, as opposed to the 300.000 employed in the preceding chapter.

5.1 The Results

Document ID	Iterations FG (opt)	Result FG (opt)	Iterations FG (no opt)	Result FG (no opt)	Iterations SCS	Result SCS
Test matrix	509	277.9400	1680	277.9515	25	277.9190
rmfgen 4-4-500	150000	365912.4916	150000	368963.1100	675	373214.4171
rmfgen 4-8-100	150000	408731.5520	150000	743103.4740	800	420140.9819
rmfgen 4-8-500	150000	372595.3829	150000	320478.9815	775	231211.1281
rmfgen 4-16-500	150000	549087.0860	150000	3030360.9037	2925	598035.2890

Table 5.1: Final values of FG methods and SCS

As demonstrated in Table 5.1, a systematic comparison of the performance of the Fast Gradient Methods and SCS solver is presented. The analysis yielded substantial disparities in both computational efficiency and solution quality.

The SCS solver exhibits superior computational efficiency, converging to acceptable solutions in a significantly smaller number of iterations (25-2925) than the FG methods, which reach a maximum of 150.000 iterations in most cases. This discrepancy is particularly pronounced in the case of more complex matrices, where SCS continues to exhibit reasonable convergence times.

A more nuanced picture emerges when the quality of the solutions is analyzed. For certain matrices, including the test matrix and rmfgen 4-4-500, the convergence of all methods results in solutions that are highly analogous, indicating that these problems are well-conditioned for both approaches. However, substantial discrepancies emerges for more intricate matrices, thereby unveiling distinct challenges. In certain instances, a considerable disparity has been identified among the FG variants, too.

To further assess the accuracy of the Fast Gradient methods, the relative errors of the final solutions are reported in Table 5.2, with respect to the reference value obtained from the SCS solver.

Document ID	Relative error (opt)	Relative error (no opt)
Test matrix	7.6e-5	1.2e-4
rmfgen 4-4-500	2.0e-2	1.1e-2
rmfgen 4-8-100	2.7e-2	7.7e-1
rmfgen 4-8-500	6.1e-1	3.9e-1
rmfgen 4-16-500	8.2e-2	4.1e+0

Table 5.2: Relative errors between FG methods and SCS

The results of the study indicate a consistent trend. For simpler problem instances, both FG variants converge to solutions with low relative error, indicating a strong alignment with the SCS results. In particular, the FG method, when equipped with access to the optimal value, consistently yields significantly reduced error margins in comparison to its version lacking this capability.

As the complexity of the instances increases, the relative error of the variant without the optimal value grows significantly, reaching up to 4.1 in the most challenging case. This underscores the challenge of preserving solution accuracy in settings characterized by ill-conditioning or high dimensionality, without access to the optimal value. This phenomenon can be attributed to the challenges associated with a very small mu, as discussed in the previous chapters.

These findings serve to reinforce the earlier observation that while FG methods have the capacity to achieve high-quality results under favorable conditions, they exhibit a heightened sensitivity to problem complexity and prior knowledge of the optimal solution. Moreover, substantial practical constraints are encountered with regard to convergence speed and robustness. In contrast, the SCS solver demonstrated a consistent ability to maintain both accuracy and efficiency across all tested instances.

Bibliography

[1] S. Balakrishnan. 10-725: Convex optimization lecture 11, 2023. Spring 2023, lecture notes.

[2] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[3] H. Fawzi. Topics in convex optimization, 2019. Michaelmas 2019, lecture notes.

[4] A. Frangioni. Nonsmooth unconstrained optimization, 2023. Optimization Methods for Data Science, Master in Data Science and Business Informatics, University of Pisa 2023/2024.

[5] A. Frangioni, B. Gendron, and E. Gorgone. Dynamic smoothness parameter for fast gradient methods. *Optimization Letters*, 12:43–53, 2018.

[6] Y. Nesterov. Smooth minimization of non-smooth functions. *Mathematical Programming*, 103(1):127–152, 2005.