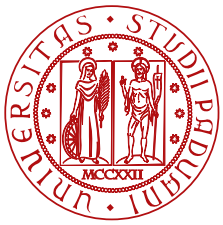

UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA MAGISTRALE IN CONTROL SYSTEMS ENGINEERING

On the Rate of Convergence and Byzantine Resilience of Decentralized Optimization Algorithms over Connected Networks

Relatore
Prof. Carli Ruggero, PhD.

Laureando
Armijos Bacuilima Leonardo M.

ANNO ACCADEMICO 2022-2024

Data di laurea 07/07/2025

To my mom A.B for her unconditional support
To my heart G.Ye for her undying love through it all
A special thanks to my supervisors for their infinite patience,
and to all my friends, (specially K.I) who made Padova home.

Abstract

The present work concerns practical and theoretical aspects regarding decentralized optimization over distributed connected networks, specifically, their rate of convergence and resilience against byzantine attacks. These two aspects are discussed within the framework of common consensus strategies, such as the Relaxed Alternating Direction Method of Multipliers (R-ADMM) and Consensus Algorithms via Stochastic Matrices. Concerning the rate of convergence in the decentralized setting, we focus our attention to regular graphs holding functions of the same convexity, this allows for a simplification of the R-ADMM algorithm to linear form analyzable with respect to the eigenvalues of the matrix dictating the convergence. The result of the analysis highlights heavily the importance of hyper-parameters chosen beyond the classical values used in the ADMM, notably increasing the rate of convergence of the algorithm. Numerical simulations supporting the evidence of the theoretical analysis are provided. The second part of this work discusses Byzantine attacks, the general setup in which they are studied and common strategies used for dealing with this critical security issue. In this section the setting is restricted by limiting our attention towards problems in which the objective functions in each node are all equal. This section heads towards an improved strategy for identifying and filtering adversarial agents. From a theoretical perspective, this involves non-trivial challenges such as ensuring convergence, optimality of the solution, and correct synchronized distributed filtering. The proposed solution addresses these challenges with satisfactory results. To conclude this portion, numerical simulations were performed on popular machine learning datasets to showcase the effectiveness of the proposed strategy. This thesis aims to collaborate to the understanding and development of decentralized optimization due to its importance to the field of Control Systems. The results presented are important both practically and theoretically, having implications for areas such as multi-agent coordination, security over networked components, and decentralized learning. It is in our hope that the present work encourages future work, especially regarding convergence and optimality analysis on consensus algorithms as well security of networked systems under byzantine attacks.

Contents

List of Figures

Introduction

The present work provides an exploration of decentralized optimization in distributed connected networks. We have focused into two primary areas, the rate of convergence of consensus algorithms and their resilience against Byzantine attacks. The significance of this work lies in tis role in the field of Control Systems and its broader implications for multi-agent coordination, security over networked components, and decentralized learning. The foundational mathematical concepts, including matrices, graphs, optimization algorithms (like Gradient Descent and Newton's Method), and various consensus algorithms (such as AB-Push Pull and ADMM/R-ADMM), are introduced to provide a background for the subsequent analyses.

The initial part of the thesis is dedicated to understanding and improving the rate of convergence of the decentralized optimization algorithm R-ADMM. We specifically examine this common consensus strategy, the Relaxed Alternating Direction Method of Multipliers as well as Consensus Algorithms via Stochastic Matrices later on. The study simplifies the R-ADMM algorithm to a linear form by concentrating on regular graphs where functions possess the same convexity. This simplification enables the analysis of the algorithm's convergence properties, specifically, facts regarding the eigenvalues of the matrix in the reduced linear system. A significant finding from this analysis is the importance of tuning hyper-parameter α beyond their conventional value in the standard ADMM, leading to a notable increase in the speed of the algorithm's convergence rate. Numerical simulations are provided to substantiate these theoretical claims.

The latter section of the thesis addresses the challenge of Byzantine attacks in distributed systems. It outlines the typical setup for studying these attacks and discusses existing strategies for handling this security issue. The scope in this part is narrowed to problems where the objective functions across all nodes are identical. The contribution we attempted to make in this area is an improved strategy for identifying and filtering out adversarial agents. This involves theoretical considerations, such as guaranteeing convergence, ensuring the optimality of the solution, achieving correct synchronized distributed filtering, among others. The effectiveness of the proposed strategy is demonstrated through numerical simulations conducted on widely used machine learning datasets.

Chapter 1

Mathematical Background

For the convenience of the lector, in this section we present useful mathematical definitions and equalities used throughout this project.

1.1 On Matrices

Consider an $n \times n$ real matrix A. The eigenvalue problem centers on finding scalar values $\lambda \in \mathbb{C}$ and corresponding vectors $v \in \mathbb{C}^n$ such that the equation $Av = \lambda v$ holds. Then, λ will represent an **eigenvalue** while v will serve as its associated **right eigenvector**. Additionally, we can define **left eigenvectors** as vectors $w \in \mathbb{C}^n$ satisfying the transpose relation $w^T A = \lambda w^T$.

The spectral radius of matrix A, denoted $\rho(A)$, represents the largest absolute value among all eigenvalues in the spectrum of A. Mathematically, $\rho(A) = \max\{|\lambda| \mid \lambda \in \text{spec}(A)\}$, where $\text{spec}(A)$ is the set of all eigenvalues of A. From a geometric perspective, the spectral radius can be interpreted as the radius of the smallest circle in the complex plane that is centered at the origin and encloses all eigenvalues of matrix A.

During our discussions, it will be useful to characterize the convergence behavior of matrix powers in relation to the spectral radius. For any square matrix A, convergence occurs, meaning $\lim_{k\to\infty} A^k = 0_{n\times n}$, when the spectral radius satisfies $\rho(A) < 1$. When the spectral radius equals unity, the matrix is partially convergent: the sequence $\{A^k\}$ approaches a limit different from the zero matrix if and only if the eigenvalue 1 appears in the spectrum with geometric multiplicity equal to its algebraic multiplicity, and every remaining eigenvalue in the spectrum has magnitude strictly less than 1.

For a square matrix $A \in \mathbb{R}^{n\times n}$, we establish the following: the matrix is termed **non-negative** when every entry satisfies $a_{ij} \geq 0$, and **positive** when all entries are strictly positive ($a_{ij} > 0$) across all indices $i, j \in \{1, \ldots, n\}$. We can further describe of non-negative/positive matrices by the behavior of their powers. A matrix A is said to be **irreducible** when the sum of its first $n-1$ powers, $\sum_{k=0}^{n-1} A^k$, yields a matrix with strictly positive entries throughout. A matrix is **primitive** when there exists some positive integer k such that the k-th power A^k contains only positive entries. In later chapter we will observe that primitive matrices represent the strongest form of connectivity in a network, where sufficiently high powers eliminate any periodic or blocking behavior that might prevent complete mixing within the system represented by the matrix.

1.1.1 Stochastic Matrices

Discrete-time linear systems will be important further down the line, they will be characterized by matrices possessing distinctive structural properties. We use matrices containing exclusively non-negative elements where each row sums to unity. A non-negative matrix becomes **row-stochastic** when its rows sum to one (formally, $A\mathbf{1}_n = \mathbf{1}_n$), while it is **column-stochastic** when its columns sum to one (expressed as $A^T\mathbf{1}_n = \mathbf{1}_n$). When a

matrix simultaneously satisfies both row and column stochastic properties, we classify it as **doubly-stochastic**.

Every row-stochastic matrix A possesses spectral properties that arise from its structure. First, the value 1 always appears as an eigenvalue in the spectrum, which follows from the row-sum condition that defines row-stochastic matrices. Second, all eigenvalues of A are confined within the closed unit disk in the complex plane, with the spectral radius achieving its maximum possible value of $\rho(A) = 1$. This property ensures that the eigenvalues satisfy $|\lambda| \leq 1$ for every $\lambda \in \text{spec}(A)$, meaning that we could eventually characterize the long-term behavior of iterative processes involving row-stochastic matrices. A proof of these two properties can be obtained from the Gersgorin Disk Theorem [1]

By the same token, the Perron-Frobenius Theorem as stated in [1], allow us to further discuss convergence on row-stochastic matrices. Take a non-negative matrix A where the dominant eigenvalue λ is both simple (having algebraic multiplicity one) and strictly dominates all remaining eigenvalues in absolute value. Under these spectral conditions, the normalized powers of A comply with: $\lim_{k\to\infty} \frac{A^k}{\lambda^k} = vw^T$, where v and w represent the properly normalized right and left dominant eigenvectors respectively, with $v \geq 0$, $w \geq 0$, and the normalization constraint $v^T w = 1$. The resulting limit matrix vw^T forms a rank-one projection that captures the long-term asymptotic behavior of the matrix transformation that will characterize the steady-state dynamics of the a discrete system such as:

$$x\left(k+1\right) = Ax\left(k\right), \quad x\left(0\right) = x_0 \tag{1.1}$$

1.2 On graphs

Next we link some important ideas from graph theory to our previous review about stochastic matrices and their convergence.

In graph theory, an **undirected graph** (or simply a **graph**) is a mathematical structure that has two components: a collection V of **nodes** (or vertices) and a collection E of **edges** represented as unordered pairs of distinct nodes. When we have nodes $u, v \in V$ with $u \neq v$, the notation $\{u, v\}$ represents an undirected edge connecting them. The concept of adjacency: two nodes u and v are **neighbors** when there exists an undirected edge $\{u, v\}$ between them. The **degree** of a node v counts the total number of edges incident to that node, i.e. how many neighbors it has within the graph structure. A graph achieves **regularity** when every node possesses identical degree.

We now describe how nodes relate to each another through edge connections. A **path** represents an ordered sequence of nodes where each consecutive pair forms a valid edge in the graph structure. When examining walks, we distinguish between general walks and **simple walks**, where the latter prohibits repeated node visits except in the special case where the starting and ending nodes coincide. A graph is **connected** when every pair of nodes can be linked through some walk. Finally **cycles** are defined as simple walks that begin and terminate at the same node. Graphs without cycles are **acyclic**.

1.2.1 Matrices and graphs

For weighted graph $G = (V, E)$ with weights $\{a_e\}_{e\in E}$ where the vertex set is $V = \{1, \ldots, n\}$, we can define a **weighted adjacency matrix** as an $n \times n$ non-negative matrix A that encodes both the structural and weight information of a graph. Whenever an edge (i, j) exists in the edge set E, the matrix entry a_{ij} takes on the value of that edge's associated weight $a_{(i,j)}$, while all positions corresponding to absent edges remain zero.

Consider such weighted graph G with at least two nodes and its corresponding weighted adjacency matrix A. We can establish three equivalent properties. The matrix A exhibits **irreducibility** if and only if no permutation matrix P exists that can transform PAP^T into a block triangular form through row and column reordering. This algebraic condition is further equivalent to the graph G possessing strong connectivity, which means that every node can

reach every other node through some path. These equivalences show how algebraic properties of adjacency matrices directly reflect the structural characteristics of the underlying network topology.

Moving on, we consider the weighted graph G with n nodes with self-loops at every vertex. For such graphs with weighted adjacency matrix A, the graph achieves strong connectivity if and only if the matrix A exhibits primitivity through its $(n-1)$-th power. This primitivity condition requires that A^{n-1} contains strictly positive entries throughout, indicating that within $n-1$ steps, transitions between any pair of nodes become possible. The presence of self-loops at all nodes eliminates potential periodicity issues that might otherwise prevent this equivalence, resulting in a correspondence between strong connectivity and matrix primitivity.

It is further possible to relax the condition on the self-loops. Take a weighted graph G containing at least two nodes with corresponding weighted adjacency matrix A. The graph exhibits both strong connectivity and aperiodicity if and only if the adjacency matrix A is primitive. Matrix primitivity manifests as the existence of some positive integer k such that A^k contains exclusively positive entries, meaning that within exactly k steps, direct transitions become possible between any pair of nodes with positive weights. The elimination of periodicity in strongly connected networks corresponds to matrix powers eventually achieving entry-wise positivity, bridging graph theoretic concepts of connectivity with the structural properties of the matrix representation of a graph.

1.2.2 Averaging Systems

We now introduce useful statements for a discrete time system such as 1.1 and the feasibility of the state x reaching a consensus value among its components. A proof of this statements can be found in [1]

Consider a row-stochastic matrix A and its corresponding graph G, where the network strongly connected and aperiodic. Under these conditions, twe can characterize the long-term behavior of the system. First, from a spectral perspective, the eigenvalue 1 maintains simplicity (algebraic multiplicity one) while all remaining eigenvalues have magnitude strictly less than unity. Second, the matrix exhibits semi-convergence behavior where $\lim_{k\to\infty} A^k = \frac{1}{n}w^T$, with $w \in \mathbb{R}^n$ being a non-negative vector satisfying normalization ($\mathbf{1}_n^T w = 1$) and left-eigenvector properties ($w^T A = w^T$).

When these conditions hold consensus dynamics ensue. For the averaging model $x(k+1) = Ax(k)$, the system converges to $\lim_{k\to\infty} x(k) = (w^T x(0))\mathbf{1}_n$, representing a weighted average of initial conditions with weights determined by the matrix complatible with the network structure. In doubly-stochastic networks, this simplifies to uniform weighting ($w = \frac{1}{n}\mathbf{1}_n$), yielding convergence to the arithmetic mean: $\lim_{k\to\infty} x(k) = \frac{\mathbf{1}_n^T x(0)}{n}\mathbf{1}_n$. The behavior shows that convergence values represent weighted averages of initial conditions, where the weighting coefficients $w_1, \ldots, w_n$ reflect the relative "influence" of each agent in the network.

1.3 Optimization Algorithms

During this project we will make use of simple distributed optimization algorithms. Before introducing their distributed counterpart, we offer a short review of the centralized version of these algorithms, leaving the network case for the next section.

1.3.1 Gradient Descent

The present work mainly concerns applications towards distributed machine learning, and so a quite popular optimization tool was selected in order to display the effectiveness of the applications proposed. Gradient descent stands out because of its simplicity and adaptability to different problems. A quick review is in order and it will later be used as a part of more complex algorithms.

Recall first a simple principle from analysis: for a differentiable function $F(\mathbf{x})$ defined in a neighborhood around point $\mathbf{a}$, the steepest rate of decrease occurs when moving in the direction opposite to the gradient vector $\nabla F(\mathbf{a})$. This leads to the iterative update rule $\mathbf{a}_{n+1} = \mathbf{a}_n - \gamma \nabla F(\mathbf{a}_n)$, where the step size (or learning rate) $\gamma \in \mathbb{R}_+$ controls how far we move in the descent direction. When γ is sufficiently small, this update guarantees that $F(\mathbf{a}_n) \geq F(\mathbf{a}_{n+1})$, ensuring progress toward lower function values. The algorithm subtracts the scaled gradient $\gamma \nabla F(\mathbf{a})$ from the current position because we seek to move against the gradient's direction, which points toward the steepest increase.

Starting from an initial guess $\mathbf{x}_0$ for a local minimum, gradient descent generates the sequence $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots$ using the recurrence relation $\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma_n \nabla F(\mathbf{x}_n)$ for $n \geq 0$. This process creates a monotonically decreasing sequence of function values: $F(\mathbf{x}_0) \geq F(\mathbf{x}_1) \geq F(\mathbf{x}_2) \geq \cdots$, which under appropriate conditions ensures convergence to a local minimum. A complete disscusion on such conditions can be further found in [2]

The flexibility to adjust the step size γ at each iteration allows for adaptive optimization strategies that can improve convergence behavior. Theoretical convergence guarantees can be established under specific regularity conditions on the objective function F, such as convexity and Lipschitz continuity of the gradient ∇F, combined with the aforementioned step size selection strategies.

1.3.2 Newton's Method

A similar result follow from Newton's Algorithm. Consider first a simpler scenario involving single-variable functions before progressing to more relevant multi-variable case in our application. When working with a twice-differentiable function $f : \mathbb{R} \to \mathbb{R}$, our objective is to solve the optimization problem $\min_{x \in \mathbb{R}} f(x)$. Newton's approach to solving this problem the sequence $\{x_k\}$ starting from an initial point $x_0 \in \mathbb{R}$ that converges toward a minimizer x_* of f by using second-order Taylor series approximations around each iteration point. The Taylor expansion provides a quadratic approximation of the original function in the neighborhood of the current point x_k, given by $f(x_k + t) \approx f(x_k) + f'(x_k)t + \frac{1}{2}f''(x_k)t^2$. To find the next iteration point x_{k+1}, we minimize this quadratic approximation by treating it as a function of the step size parameter t, where $x_{k+1} = x_k + t$. When the second derivative is positive (convex function), we can locate the minimum by setting the derivative of the approximation equal to zero: $\frac{d}{dt}\left[f(x_k) + f'(x_k)t + \frac{1}{2}f''(x_k)t^2\right] = 0$, which yields the optimal step size $t = -\frac{f'(x_k)}{f''(x_k)}$. From here we obtain Newton's iterative algorithm $x_{k+1} = x_k + t = x_k - \frac{f'(x_k)}{f''(x_k)}$. This, of course, can be extended to multiple dimensions.

The extension is intuitive, substitute the first derivative with the gradient vector and the reciprocal of the second derivative with the inverse of the Hessian matrix. This generalization leads to $x_{k+1} = x_k - [f''(x_k)]^{-1} f'(x_k)$ for $k \geq 0$. In practical implementations, Newton's method is frequently adapted to incorporate a small step size parameter γ where $0 < \gamma \leq 1$ rather than using $\gamma = 1$, resulting in the modified $x_{k+1} = x_k - \gamma[f''(x_k)]^{-1} f'(x_k)$. This step size modification helps control the magnitude of each update and can improve convergence. The convergence rate of Newton's Algorithm is in general faster than that of gradient descent, providing a quadratic convergence. The trade-off lies in the practical implementation of the inverse hessian at each iteration, which may be costly. For a full in-depth description we refer ourselves to [3]

1.4 Consensus Algorithms

The previous iterative schemes will be now applied to common consensus algorithms in the context of distributed learning. Take for example a system of m agents, where each agent i has a private loss function $f_i(x_i)$, then the collective goal is to collaboratively solve a problem such as minimizing the average sum of these functions. This typically involves each agent maintaining a copy of the decision variable x_i and enforcing that all these copies agree

$(x_i = x_j)$ through a consensus update.

A common approach for this involves each agent i updating its estimate x_i^k by taking a **weighted average** of its own and its neighbours' estimates:

$$x_i^{k+1} = \sum_{j \in N_i} a_{ij} x_j^k \tag{1.2}$$

where N_i is the set of agent i's neighbours, and a_{ij} are positive weights chosen by agent i such that $\sum_{j \in N_i} a_{ij} = 1$. These weights form a row stochastic matrix $A = \{a_{ij}\}$ that is compatible with the graph structure. If the communication graph is connected and the matrix A is row stochastic, the iterates of the consensus algorithm converge to a common point. When the matrix A is doubly stochastic the consensus point is the average of the initial values of the agents' variables. In distributed gradient methods, such as the one described by

$$v_i^k = \sum_{j=1}^{m} a_{ij} x_j^k \tag{1.3}$$

$$x_i^{k+1} = v_i^k - \alpha_k \nabla f_i(v_i^k) \tag{1.4}$$

the mixing step often uses a doubly stochastic matrix. In doing so it ensures that the average of the agents' iterates converges to the solution of the global objective function, and that the individual agent iterates also agree with this average, thereby satisfying both the optimality and agreement requirements of decentralised optimisation. A limitation of these methods is that constructing such a double stochastic matrix in a distributed manner is non-trivial if done online and under special case such as the Byzantine attack one. Moreover, these methods often need a dimishing step-size, which can slow down the algorithm and needs synchronization.

To overcome these limitations gradient-tracking algorithms were developed. These algorithms introduce an additional mixing step for estimating the gradient direction in addition to the mixing of decision variables. This allows the agents to maintain a constant step size while still achieving exact convergence to the optimal solution.

We operates at each node i with two variables: x_i^k, the state, and d_i^k, the estimated global gradient direction. The updates then become:

$$x_i^{k+1} = \sum_{j=1}^{m} a_{ij} x_j^k - \alpha d_i^k \tag{1.5}$$

$$d_i^{k+1} = \sum_{j=1}^{m} a_{ij} d_j^k + \nabla f_i(x_i^{k+1}) - \nabla f_i(x_i^k) \tag{1.6}$$

where α is a constant step size. The initial direction d_i^0 is set to the local gradient $\nabla f_i(x_i^0)$. The term $\nabla f_i(x_i^{k+1}) - \nabla f_i(x_i^k)$ ensures that only the new gradient information influences the direction update, allowing d_i^k to effectively track the average of the local gradients, which approaches the global gradient.

Gradient-tracking can achieve linear convergence to the exact global minimum for smooth and strongly convex problems. This geometric rate of convergence matches the fastest convergence rate observed in centralized gradient methods. This improvement is due to the mitigation of the steady-state error that arises in traditional distributed gradient descent methods when not using gradient tracking. In [4], a history of the development of these algorithms as well as further analysis of their behavior and properties are provided.

1.4.1 AB-Push Pull

The previously described Distributed Gradient Descent (DGD) algorithms face limitations when employing a constant step size (α), as they tend to converge to a suboptimal solution[5].

This suboptimality arises because the global minimizer, x^*, does not typically result in zero local gradients for individual agent functions $\nabla f_i(x^*) \neq 0_p$ in general, unless $\nabla f_i(x^*) = 0_p$ for all i.

To overcome this issue, the concept of Gradient Tracking (GT) was introduced. However, GT-DGD requires the weight matrix to be doubly stochastic, limiting its applicability to undirected or balanced graphs where such matrices are readily constructed as we said before. This doubly stochastic property ensures both agreement among agents, via row stochasticity, and optimality of the converged solution, via column stochasticity. We can take advantage of this separable conditions.

The AB/Push-Pull algorithm allows for different weight matrices for the decision variable updates and gradient tracking updates, removing the requirement for a single doubly stochastic matrix. This flexibility makes it applicable to both directed and undirected graphs. The algorithm proceeds at each node i with a constant step size $\alpha > 0$ as follows:

$$x_i^{k+1} = \sum_{r \in N_{in_i}} a_{ir} x_r^k - \alpha y_i^k, \qquad x_i^0 \in \mathbb{R}^p, \quad y_i^0 = \nabla f_i(x_i^0) \tag{1.7}$$

$$y_i^{k+1} = \sum_{r \in N_{in_i}} b_{ir} y_r^k + \nabla f_i(x_i^{k+1}) - \nabla f_i(x_i^k) \tag{1.8}$$

Here, $A = \{a_{ir}\}$ is a row stochastic matrix, and $B = \{b_{ir}\}$ is a column stochastic matrix. The AB/Push-Pull algorithm achieves linear convergence to the exact global minimum x^* for smooth and strongly convex problems. The formal analysis, which we skip in this small review, uses weighted Euclidean norms to demonstrate a norm-contraction for the row and column stochastic matrices, establishing a linear time-invariant error dynamics, where the spectral radius of the error matrix is less than one for a sufficiently small step size. A full proof is provided in [5]. Furthermore, we need not limit ourselves to using a combination of row and column stochastic matrices. We could use a single one of each.

By defining $\mathbf{\Pi_A}$ as a diagonal matrix with $\pi_\mathbf{A} \otimes \mathbf{1}_p$ on its main diagonal (where $\pi_\mathbf{A}$ is the positive left eigenvector of A corresponding to eigenvalue 1, such that ${\pi_\mathbf{A}}^\top A = {\pi_\mathbf{A}}^\top$ and $A\mathbf{1}_n = \mathbf{1}_n$), a state transformation $\tilde{\mathbf{x}}^k = \mathbf{\Pi_A} \mathbf{x}^k$ can be applied to the compact form of AB/Push-Pull:

$$\tilde{\mathbf{x}}^{k+1} = \tilde{\mathbf{B}} \tilde{\mathbf{x}}^k - \alpha \mathbf{\Pi_A} \mathbf{y}^k \tag{1.9}$$

$$\mathbf{y}^{k+1} = \mathbf{B}\mathbf{y}^k + \nabla f({\mathbf{\Pi_A}}^{-1} \tilde{\mathbf{x}}^{k+1}) - \nabla f({\mathbf{\Pi_A}}^{-1} \tilde{\mathbf{x}}^k) \tag{1.10}$$

Here, $\tilde{\mathbf{B}} := \mathbf{\Pi_A} \mathbf{A} {\mathbf{\Pi_A}}^{-1}$ is a column stochastic matrix.

In practice, because the eigenvector $\pi_\mathbf{A}$ is not locally known to any node, algorithms based on this transformation require separate iterations for estimating this eigenvector. This implementations add gradient tracking to the scheme, then at each node i:

$$x_i^{k+1} = \sum_{r \in N_{in_i}} \tilde{b}_{ir} x_r^k - \alpha y_i^k \tag{1.11}$$

$$z_i^{k+1} = \sum_{r \in N_{in_i}} \tilde{b}_{ir} z_r^k \tag{1.12}$$

$$y_i^{k+1} = \sum_{r \in N_{in_i}} b_{ir} y_r^k + \nabla f_i \left(\frac{x_i^{k+1}}{z_i^{k+1}} \right) - \nabla f_i \left(\frac{x_i^k}{z_i^k} \right) \tag{1.13}$$

where $\{\tilde{b}_{ir}\}$ and $\{b_{ir}\}$ are (potentially different) column stochastic weights. The variable z_i^k asymptotically estimates the right eigenvector $\pi_\mathbf{A}$ of $\tilde{B}$.

Similarly, the AB/Push-Pull framework can be also formulated in terms of algorithms that primarily employ row stochastic weights. This is achieved by applying a state transformation to the gradient tracking update $\mathbf{y}^k$. Let $\mathbf{\Pi_B} := \text{diag}(\pi_\mathbf{B}) \otimes \mathbf{I}_p$, where $\pi_\mathbf{B}$ is the positive right eigenvector of B corresponding to the eigenvalue 1 (such that $B\pi_\mathbf{B} = \pi_\mathbf{B}$ and $\mathbf{1}_n^\top B = \mathbf{1}_n^\top$). The transformation $\tilde{\mathbf{y}}^k := \mathbf{\Pi_B}^{-1}\mathbf{y}^k$ leads to a modified system where the gradient tracking update uses a row stochastic matrix $\tilde{\mathbf{A}} := \mathbf{\Pi_B}^{-1}\mathbf{B}\mathbf{\Pi_B}$.

Similar to the column stochastic case, explicit knowledge of the eigenvector $\pi_\mathbf{B}$ is not available locally at each node. Therefore, additional iterations are introduced to estimate this eigenvector. The iterations at each node i are:

$$x_i^{k+1} = \sum_{r \in N_{in_i}} a_{ir} x_r^k - \alpha \tilde{y}_i^k \tag{1.14}$$

$$e_i^{k+1} = \sum_{r \in N_{in_i}} \tilde{a}_{ir} e_r^k \tag{1.15}$$

$$\tilde{y}_i^{k+1} = \sum_{r \in N_{in_i}} \tilde{a}_{ir} y_r^k + \nabla f_i \left(\frac{x_i^{k+1}}{[e_i^{k+1}]_i} \right) - \nabla f_i \left(\frac{x_i^k}{[e_i^k]_i} \right) \tag{1.16}$$

with $x_i^0 \in \mathbb{R}^p$ arbitrary, $y_i^0 = \nabla f_i(x_i^0)$, and $e_i^0 \in \mathbb{R}^n$ being a vector of zeros with a one at the i-th entry. The auxiliary variable e_i^k asymptotically estimates the eigenvector $\pi_\mathbf{B}$ .This strategy combines the benefits of gradient tracking with row stochastic weights, which are often easier to construct in a distributed manner as they only require knowledge of incoming neighbors' weights.

1.4.2 The ADMM and the R-ADMM

The ADMM algorithm is a method for solving optimization problems structured as the sum of two convex functions subject to linear constraints. It is well-suited for distributed computing due to its structure and numerical efficiency.

The general optimization problem that ADMM addresses is presented as:

$$\min_{x \in X, y \in Y} \{f(x) + g(y)\} \quad \text{s.t.} \quad Ax + By = c \quad [249,\ (1)] \tag{1.17}$$

where X and Y represent Hilbert spaces (general possibly infinite Euclidean spaces with a well defined inner product), and f and g are closed, proper, and convex functions. The variables x and y are the optimization variables, and A, B, c are matrices and a vector, respectively.

To solve this problem, ADMM employs an augmented Lagrangian function, denoted as $\mathcal{L}_\rho(x, y; w)$. This function incorporates the original objective, the linear constraint, and a quadratic penalty term for the constraint violation controlled by the parameter $\rho > 0$. The variable w represents the vector of Lagrange multipliers.

$$\mathcal{L}_\rho(x, y; w) = f(x) + g(y) - w^\top (Ax + By - c) + \frac{\rho}{2} \|Ax + By - c\|^2 \tag{1.18}$$

The ADMM then algorithm proceeds iteratively as follows:

$$y^{(k+1)} = \arg\min_y \mathcal{L}_\rho(x^{(k)}, y; w^{(k)}) \tag{1.19}$$

$$w^{(k+1)} = w^{(k)} - \rho(Ax^{(k)} + By^{(k+1)} - c) \tag{1.20}$$

$$x^{(k+1)} = \arg\min_x \mathcal{L}_\rho(x, y^{(k+1)}; w^{(k+1)}) \tag{1.21}$$

It is important to note that ADMM is guaranteed to converge to the optimal solution of the problem for any $\rho > 0$, provided that the non-augmented Lagrangian has a saddle point, further discussion about these fact lie on [6].

The Relaxed ADMM (R-ADMM) is a more general algorithm from which the standard ADMM can be derived. The R-ADMM's arises from applying the Relaxed Peaceman-Rachford Splitting (R-PRS) method to the Lagrange dual of the original optimization problem.

The notion of proximal operator is important for understanding the formulation. For a closed, proper, and convex function f and a penalty $\rho > 0$, the proximal operator, $\text{prox}_\rho f(y)$, is defined as:

$$\text{prox}_\rho f(y) = \arg\min_{x \in X} \left\{ f(x) + \frac{1}{2\rho}\|x - y\|^2 \right\} \tag{1.22}$$

Then, the reflective operator is defined as $\text{refl}_\rho f = 2\text{prox}_\rho f - I$, where I is the identity operator.

For an optimization problem structured as $\min_{x \in X}\{f(x)+g(x)\}$, the Peaceman-Rachford operator, T_{PRS}, is defined as $T_{\text{PRS}} = \text{refl}_\rho f \circ \text{refl}_\rho g$. The R-PRS iteration is then:

$$z^{(k+1)} = (1-\alpha)z^{(k)} + \alpha T_{\text{PRS}} z^{(k)} \tag{1.23}$$

where z is an auxiliary variable and α is a relaxation parameter. The optimal solution of the problem can be found from the limit of $z^{(k)}$ as $x^* = \text{prox}_\rho g(z^*)$.

An explicit implementation of the R-PRS algorithm can be posed as:

$$\psi^{(k)} = \text{prox}_\rho g(z^{(k)}) \tag{1.24}$$

$$\xi^{(k)} = \text{prox}_\rho f(2\psi^{(k)} - z^{(k)}) \tag{1.25}$$

$$z^{(k+1)} = z^{(k)} + 2\alpha(\xi^{(k)} - \psi^{(k)}) \tag{1.26}$$

This implementation of the computation of proximal operators is split between different equations, which is the origin of the splitting in its name. According to [7], the R-PRS algorithm is guaranteed to converge to a fixed point of T_{PRS}. Next, the R-ADMM algorithm is derived by applying the R-PRS method to the Lagrange dual of the original problem:

$$\min_{w \in W}\{d_f(w) + d_g(w)\} \tag{1.27}$$

where $d_f(w) = f^*(A^\top w)$ and $d_g(w) = g^*(B^\top w) - w^\top c$, with f^* and g^* being the convex conjugates of f and g respectively. This dual problem is used because it involves minimization over a single variable, making it suitable for R-PRS. The proximal updates in R-PRS can be then translated into the forms for the dual variables. A compact representation of the R-ADMM can be implemented as:

$$y^{(k+1)} = \arg\min_y \left\{ \mathcal{L}_\rho(x^{(k)}, y; w^{(k)}) + \rho(2\alpha - 1)\langle By, (Ax^{(k)} + By^{(k)} - c)\rangle \right\} \tag{1.28}$$

$$w^{(k+1)} = w^{(k)} - \rho(Ax^{(k)} + By^{(k+1)} - c) - \rho(2\alpha - 1)(Ax^{(k)} + By^{(k)} - c) \tag{1.29}$$

$$x^{(k+1)} = \arg\min_x \mathcal{L}_\rho(x, y^{(k+1)}; w^{(k+1)}) \tag{1.30}$$

The standard ADMM is recovered from this R-ADMM formulation by setting $\alpha = 1/2$, which eliminates the additional terms weighted by $(2\alpha - 1)$. The R-ADMM features two tunable parameters, ρ and α, unlike ADMM which only has ρ. This additional parameter provides greater flexibility that can potentially lead to faster convergence rates. This will a significant part of our case study.

Chapter 2

On R-ADMM Convergence

The design of algorithms for control over networks has attracted research during the past years [8], [9], [10], [11]. Attention has been devoted to distributed optimizations, mainly because of the practical relevance that these problems have [12],[13] and [14]

A typical problem considered commonly is **distributed consensus optimization** where a network of agents contribute to minimize the sum of their local objective functions over a common variable [15]; the main challenge is that the information exchange is restricted between neighbors.

This problem has been addressed by implementing first-order based methods [16], like gradient tacking, second-order based methods, like Newton-Raphson [17], [18], or Lagrangian dual methods [19], [20]. Within the aforementioned, emphasis has been put on the ADMM algorithm [21], which has been recognized as a powerful tool to solve a large class of optimization problems.

Several ADMM-based algorithms have been designed to solve optimization problems over networks exhibiting different features like asynchronous updates, robustness to packet losses [[22], [23]], inclusion of local or coupled constraints [[24]]; and extensive research has been performed to characterize the convergence rate of these, [[25],[26]].

In [[27]], the authors have introduced a distributed method based on the relaxed ADMM algorithm (R-ADMM). The R-ADMM is more general than the classical ADMM. The ADMM algorithm depends on a parameter, typically denoted by ρ, while the R-ADMM depends on 2, ρ and α, and collapses to the classical ADMM when $\alpha = 1/2$. The algorithm proposed in [[27]], is shown to be acceptant of asynchronous implementations and to be robust against packet losses. Moreover, linear convergence was proved if the local cost functions are strongly convex. A deeper analysis was not provided.

In this section we provide a more compact description of the R-ADMM based algorithm proposed in [[27]], Namely, an edge-based description: introducing a set of variables defined along the edges of the network. We show that, when considering a synchronous implementation, the algorithm in [[27]] can be rewritten as a node-based formulation, reducing the number of variables involved in the dynamics.

Then, assuming that the cost functions are quadratic, with the same convexity, and that the underlying graph is described by a regular graph, we provide closed-form formulas for the eigenvalues of the matrix modeling the dynamics of the R-ADMM. These formulas enable a procedure to evaluate the convergence rate and optimize it with respect to ρ and α. Numerical results are provided for circulant graphs, hinting at the R-ADMM being able to achieve superior convergence rates compared to the classical ADMM, which derives from the additional parameter α which instead is fixed in the ADMM.[28]

2.1 Problem Formulation

[28]

Let $\mathrm{diag}(v)$ be the m-dimensional diagonal matrix whose i-th diagonal entry is $[v]_i$, Denote

as I_m and 0_m the identity and zero matrices in $\mathbb{R}^{m\times m}$. Having $v \in \mathbb{R}^m$ with $[v]_i$ as its i-th component. By $\|\cdot\|$ it is meant the L2 norm of the argument.

An undirected graph will be $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of N nodes named $\{1, \ldots, N\}$ with $\mathcal{E}$ as the set of edges. For $i \in \mathcal{V}$, $\mathcal{N}_i = \{j \in \mathcal{V} : (i,j) \in \mathcal{E}\}$ will be the set of its neighbors, and $d_i = |\mathcal{N}_i|$ its degree. The adjacency matrix of $\mathcal{G}$ is $A \in \mathbb{R}^{N\times N}$, where its elements $[A_{ij}]$ are 1 if $(i,j) \in \mathcal{E}$ and 0 otherwise. Define also the degree matrix of $\mathcal{G}$ as $D \in \mathbb{R}^{m\times m}$, with $[D_{ii}] = d_i$ and zero everywhere else.

Let f be a scalar function such that $f : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$. So, f is closed if the set $\{x \in \mathrm{dom}(f) | f(x) \leq a\}$ is closed $\forall a \in \mathbb{R}$. Moreover, f is also proper if it does not diverge toward $-\infty$. These definitions allow for the characterization of $\Gamma_0(\mathbb{R}^n)$ as the set of all the convex, closed, and proper functions f.

2.2 The Problem

[28]

Consider a network of N agents modeled by an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. $\mathcal{V} = \{1, \ldots, N\}$ denotes the set of agents while $\mathcal{E}$ is the set of edges, specifically $(i,j) \in \mathcal{E}$ means that agents i and j can communicate with each other.

The N agents solve the following optimization problem

$$\min_x f(x) = \sum_{i=1}^{N} f_i(x) \tag{2.1}$$

defined over the common variable $x \in \mathbb{R}$, where $f_i : \mathbb{R} \to \mathbb{R}$ is a local cost function known only by i. We assume that x is scalar, but all the analysis can be extended to the general case $x \in \mathbb{R}^n$, $n > 1$. It is also assumed that $f_i \in \Gamma_0$, $i = 1, \ldots, N$, hence the minimizer of (2.1) is unique. This minimizer is denoted as x^*.

To solve (2.1) in a distributed way:

$$\begin{aligned} &\min_{x_1,\ldots,x_N} \quad \sum_{i=1}^{N} f_i(x_i) \\ &\text{s.t.} \quad x_i = x_j \quad \text{for all } (i,j) \in \mathcal{E} \end{aligned} \tag{2.2}$$

where x_i is the local copy of x at agent i. The agreement constraints $x_i = x_j$ in (2.2) make it equivalent to (2.1).

In [[27]] it is shown that the relaxed ADMM algorithm applied to the Problem in (2.2) is characterized by the following updates for each $i = 1, \ldots, N$ and $j \in \mathcal{N}_i$,

$$x_i(k+1) = \tag{2.3a}$$
$$\arg\min_{x_i} \left\{ f_i(x_i) - \left(\sum_{j\in\mathcal{N}_i} z_{ij}(k) \right) x_i + \frac{\rho d_i}{2} ||x_i||^2 \right\}$$
$$q_{i\to j}(k) = -\alpha z_{ij}(k) + 2\alpha\rho x_i(k+1) \tag{2.3b}$$
$$z_{ij}(k+1) = (1-\alpha) z_{ij}(k) + q_{j\to i}(k) \tag{2.3c}$$

Each i stores in memory x_i and z_{ij}, $j \in \mathcal{N}_i$. Then, at each iteration the algorithm performs:

- Agent i updates x_i as in (2.3a);
- For its neighbors $j \in \mathcal{N}_i$, agent i sends the quantity $q_{i\to j}$ computed as in (2.3b);

- Agent i receives $q_{j\to i}$, $j \in \mathcal{N}_i$;
- based on the received information, agent i updates z_{ij}, $j \in \mathcal{N}_i$, as in (2.3c).

Tthe algorithm involves α and ρ. In [[27]], with assumptions on f_i's, they proved that the algorithm converges to the optimal solution $x_1 = \ldots = x_N = x^*$, for $0 < \alpha < 1$ and $\rho > 0$ and independent from the initialization of $x_i(0)$ and $z_{ij}(0)$, $i = 1, \ldots, N$, $j \in \mathcal{N}_i$.[28]

There are several results on the rate of convergence of the classical distributed ADMM [[25]]. Here we provide a first analysis on the rate of convergence of the more general relaxed-ADMM that we have shown. Special care will be devoted to understand how α can be used to improve performance.[28]

2.3 Circulant Graphs

In order to continue our analysis, we will restrict the network topology of Problem 2.1 to that of Circulant Graphs.

2.3.1 Circulant Matrices

Consider the following type of matrices $\Lambda \in \mathbb{R}^{n\times n}$:

$$\Lambda = \begin{bmatrix} a_0 & a_{n-1} & \ldots & a_2 & a_1 \\ a_1 & a_0 & a_{n-1} & & a_2 \\ \vdots & a_1 & a_0 & & \vdots \\ a_{n-2} & & & \ddots & a_{n-1} \\ a_{n-1} & a_{n-2} & \ldots & a_1 & a_0 \end{bmatrix}$$

Their eigenvalues and eigenvectors are related to the n-roots of unity. Concretely, the eigenvalues can be determined in closed form as follows:

$$\lambda_j = a_0 + a_1\omega^j + a_2\omega^{2j} + \cdots + a_{n-1}\omega^{(n-1)j} \tag{2.4}$$

Where:

- $j = 0, 1, \ldots, n-1$
- $\omega = exp\left(\frac{2\pi i}{n}\right)$
- i is the imaginary unit

2.3.2 Special Case: Circulant and Symmetric Adjacency

Now, consider only the adjacency matrices given by the circulant graphs with the following properties

- Regular of even degree $k = 2, 4, 6, 8, \ldots$
- Each node is connected to the same number of nearest neighbours to the left and right
- $n > k$, Number of nodes strictly greater than its constant degree

Figure 1 depicts a typical graph of such characteristics:

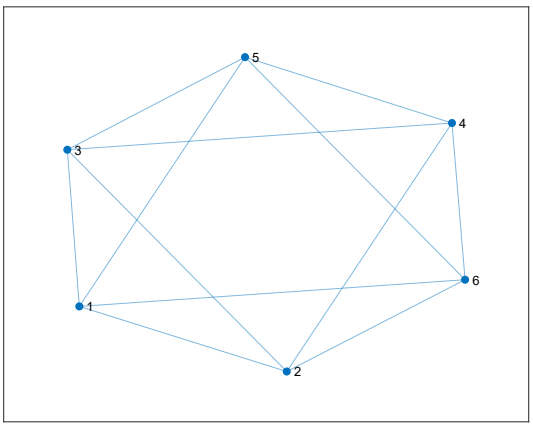

Figure 2.1: Circulant Graph for $n = 6$ and $k = 4$

These type of graphs will have the following adjacency:

$$A = \sum_{m=1}^{k/2} \left[\Pi^m + (\Pi^\intercal)^m\right], \qquad k = 2, 4, 6, 8, \ldots$$

$$\text{where} \quad \underbrace{\Pi}_{\in \mathbb{R}^{n\times n}} = \begin{bmatrix} 0 & 1 & 0 & .. & .. & 0 \\ & & 1 & & & . \\ \vdots & & \ddots & \ddots & & . \\ & & & \ddots & 1 & 0 \\ 0 & & & & & 1 \\ 1 & 0 & & \ldots & & 0 \end{bmatrix}$$

For the purposes of finding the eigenvalues, it is convenient to look at the first row of A in expanded form: (blank spaces **are not** zeros):

$$[A]_{1,*} = \begin{bmatrix} 0 & 1 & \ldots & 1 & 0 & \ldots & 0 & 1 & \ldots & 1 \end{bmatrix}$$

Notice that in the first row:

- The first element is zero followed by $k/2$ ones
- This will be followed by $(n - k - 1)$ zeros and then another $k/2$ ones

Then each successive row after row 1 is formed by shifting the previous one to the right. As a consequence the main diagonal is zero and no self loops appear.
For this type of adjacency, equation (1) can be used to calculate the eigenvalues for any size n:

$$\lambda_j = \omega^j + \omega^{2j} + \cdots + \omega^{\frac{k}{2}j} + \underbrace{\omega^{(\frac{k}{2}+1)j}}_{\omega^{(n-\frac{k}{2})j}} + \cdots + \omega^{(n-2)j} + \omega^{(n-1)j}$$

$$\lambda_j = \omega^j + \omega^{-j}(\omega^n)^j + \omega^{2j} + \omega^{-2j}(\omega^n)^j + \cdots + \omega^{\frac{k}{2}j} + \omega^{-\frac{k}{2}j}(\omega^n)^j$$

Recall $\omega^n = exp\,(2\pi i) = 1$

$$\lambda_j = \omega^j + \omega^{-j} + \omega^{2j} + \omega^{-2j} + \cdots + \omega^{\frac{k}{2}j} + \omega^{-\frac{k}{2}j}$$

$$\lambda_j = e^{i\left(\frac{2\pi}{n}j\right)} + e^{-i\left(\frac{2\pi}{n}j\right)} + e^{i\left(\frac{2\pi}{n}2j\right)} + e^{-i\left(\frac{2\pi}{n}2j\right)} + \cdots + e^{i\left(\frac{2\pi}{n}\frac{k}{2}j\right)} + e^{-i\left(\frac{2\pi}{n}\frac{k}{2}j\right)}$$

$$\boxed{\lambda_j = 2\sum_{l=1}^{k/2} \cos\left(\frac{2\pi}{n}lj\right), \qquad j = 0, 1, \ldots, n-1, \qquad k = 2, 4, 6, 8, \ldots} \tag{2.5}$$

2.4 Node-representation

[28]

We now restrict our analysis to the case of quadratic functions:

$$f_i(x_i) = b_i(x_i - v_i)^2 \tag{2.6}$$

where $b_i \neq 0$. Then, (2.3a) reduces to:

$$x_i(k+1) = \frac{1}{2b_i + \rho d_i}\left(2b_i v_i + \sum_{j\in\mathcal{N}_i} z_{ij}(k)\right). \tag{2.7}$$

Let us show that it is possible to rewrite the update equations in (2.3) in a more compact way, to this aim declare for each $i \in \mathcal{V}$, the variables:

$$z'_i(k) = \sum_{j\in\mathcal{N}_i} z_{ij}(k) \tag{2.8}$$

$$z''_i(k) = \sum_{j\in\mathcal{N}_i} z_{ji}(k). \tag{2.9}$$

Let $\mathbf{z}'$ and $\mathbf{z}''$ be the N-th dimensional vectors

$$\mathbf{z}' = \begin{bmatrix} z'_1 \\ \vdots \\ z'_N \end{bmatrix}, \qquad \mathbf{z}'' = \begin{bmatrix} z''_1 \\ \vdots \\ z''_N \end{bmatrix}.$$

In addition,

$$B = 2\ \mathrm{diag}\{b_1, \ldots, b_N\} \in \mathbf{R}^{N\times N}, \quad \mathbf{v} = \begin{bmatrix} v_1 \\ \vdots \\ v_N \end{bmatrix} \in \mathbb{R}^{N\times 1}$$

Given $\mathcal{G}$, D and A are the degree and adjacency matrices of it. Consider (2.3) for the quadratic functions (2.6). Then, the evolution of $\mathbf{z}'$ and $\mathbf{z}''$ is given by the dynamical system:

$$\begin{bmatrix} z'(k+1) \\ z''(k+1) \end{bmatrix} = F \begin{bmatrix} z'(k) \\ z''(k) \end{bmatrix} + GB\mathbf{v}, \tag{2.10}$$

where

$$F = \begin{bmatrix} (1-\alpha)I + 2\alpha\rho A(B+\rho D)^{-1} & -\alpha I \\ -\alpha I + 2\alpha\rho D(B+\rho D)^{-1} & (1-\alpha)I \end{bmatrix} \in \mathbb{R}^{2N\times 2N},$$

$$G = \begin{bmatrix} 2\alpha\rho A(B+\rho D)^{-1} \\ 2\alpha\rho D(B+\rho D)^{-1} \end{bmatrix} \in \mathbb{R}^{2N\times N}.$$

Use (2.7) and (2.8):

$$x_i(k+1) = \frac{2b_i v_i + z'_i(k)}{2b_i + \rho d_i} \tag{2.11}$$

Expand (2.8) and (2.9) one-step ahead using (2.11):

$$z'_i(k+1) = (1-\alpha)z'_i(k) + 2\alpha\rho \sum_{j\in\mathcal{N}_i} \frac{z'_j(k)}{2b_j + \rho d_j} \tag{2.12}$$

$$- \alpha z''_i(k) + 2\alpha\rho \sum_{j\in\mathcal{N}_i} \frac{2b_j v_j}{2b_j + \rho d_j}$$

$$z_i''(k+1) = -\alpha z_i'(k) + 2\alpha\rho \frac{z_i'(k) d_i}{2b_j + \rho d_j} + (1-\alpha) z_i''(k) + 2\alpha\rho \frac{2b_i v_i d_i}{2b_i + \rho d_i} \tag{2.13}$$

Stack z_i' and z_i'' as in (2.4), then:

$$\begin{bmatrix} \vdots \\ 2\alpha\rho \sum_{j \in \mathcal{N}_i} \frac{z_j'(k)}{2b_j + \rho d_j} \\ \vdots \end{bmatrix} = 2\alpha\rho A (B + \rho D)^{-1} z'(k) \tag{2.14}$$

$$\begin{bmatrix} \vdots \\ 2\alpha\rho \frac{z_i'(k) d_i}{2b_j + \rho d_j} \\ \vdots \end{bmatrix} = 2\alpha\rho D (B + \rho D)^{-1} z' \tag{2.15}$$

The term with B and D is always invertible, from the fact that $\rho > 0$, D is a degree matrix of a connected graph and B has terms different from zero in the diagonal by assumption.[28]. Equation (2.10) arises by stacking each term in (2.12) and (2.13) for each node and using (2.14) and (2.15).

Convergence of the dynamical system in (2.10) are related to the eigenvalues of the matrix $\mathbf{F}$, and we subsequently head to analyze its spectrum.[28]

2.5 Spectrum of F for regular connected graphs

Theoretical analysis of the spectrum of $\mathbf{F}$ is not trivial. Interesting facts can be understood by restricting the study to the family of circulant graphs and taking all the functions f_i with the same convexity, meaning $b_1 = \ldots = b_N = \bar{b} = 1/2$. Remember that a graph is regular if all nodes have the same degree d, that is, $d_1 = \ldots = d_N = d$. Notice that in this case $D = dI$. [28]

Let A be the adjacency matrix of a circulant graph and let

$$\text{Spec}(A) = \{\lambda_1, \ldots, \lambda_N\}$$

be its spectrum. Assume that the eigenvalues $\lambda_1, \ldots, \lambda_N$ are ordered such that $\lambda_i \geq \lambda_{i+1}$. Then the following properties follow from Perron-Frobenius Theorem[1]:

$$\lambda_1 = d \tag{2.16}$$

$$|\lambda_i| < d, \quad i = 2, \ldots, N. \tag{2.17}$$

The matrix F becomes

$$F = \begin{bmatrix} (1-\alpha)I + 2\alpha\rho A (I + \rho D)^{-1} & -\alpha I \\ -\alpha I + 2\alpha\rho D (I + \rho D)^{-1} & (1-\alpha) I \end{bmatrix}.$$

Spectrum of F: Realize that, since $A = A^T$, then A can be diagonalize by an orthogonal matrix U, that is $A = U \Lambda U^T$, where $\Lambda = \text{diag}\{\lambda_1 = d, \lambda_2, \ldots, \lambda_N\}$.

Now, let

$$T = \begin{bmatrix} U & \mathbf{0}_{N \times N} \\ \mathbf{0}_{N \times N} & U \end{bmatrix}$$

then, algebraically, it is possible to do:

$$F = T \Phi T^T$$

where

$$\Phi = \left[\begin{array}{ccc|ccc} 1-\alpha+\lambda_1 k & & & -\alpha & & \\ & \ddots & & & \ddots & \\ & & 1-\alpha+\lambda_n k & & & -\alpha \\ \hline -\alpha+dk & & & 1-\alpha & & \\ & \ddots & & & \ddots & \\ & & -\alpha+dk & & & 1-\alpha \end{array}\right]$$

with $k = \frac{2\alpha\rho}{1+\rho d}$. With a permutation matrix P, rewrite Φ as a block diagonal matrix with 2×2 entries in the diagonal

$$\mathcal{F} = P\Phi P^{-1} = \begin{bmatrix} M_1 & & \\ & \ddots & \\ & & M_N \end{bmatrix} \tag{2.18}$$

where

$$M_i = \begin{bmatrix} 1-\alpha+\lambda_i \frac{2\alpha\rho}{1+\rho d} & -\alpha \\ -\alpha+d\frac{2\alpha\rho}{1+\rho d} & 1-\alpha \end{bmatrix} \tag{2.19}$$

Observe that the eigenvalues of F coincide with the eigenvalues of $\mathcal{F}$ that are given by the union of the eigenvalues of the matrices M_i, $i = 1, \ldots, N$.[28]

In addition, note that matrix M_i depends on eigenvalue λ_i of matrix A. Thus, it is possible to write:

$$\text{Spec}(F) = \{\Lambda_1, \Lambda_2, \ldots, \Lambda_{2i-1}, \Lambda_{2i}, \ldots, \Lambda_{2N-1}, \Lambda_{2N}\}$$

where $\Lambda_{2i-1}, \Lambda_{2i}$ are the eigenvalues of M_i as a function of λ_i.

We provide an explicit expression for the eigenvalues of $\mathcal{F}$ together with their characterization.

Let $\rho > 0$ and $0 < \alpha < 1$. For $i = 1, \ldots, N$, the eigenvalues of $\mathcal{F}$ are

$$\Lambda_{2i-1} = 1 - \alpha + \frac{\alpha}{1+d\rho}\left(\lambda_i\rho + \sqrt{\rho^2\left(\lambda_i^2 - d^2\right)+1}\right) \tag{2.20}$$

$$\Lambda_{2i} = 1 - \alpha + \frac{\alpha}{1+d\rho}\left(\lambda_i\rho - \sqrt{\rho^2\left(\lambda_i^2 - d^2\right)+1}\right) \tag{2.21}$$

where λ_i is the i-th eigenvalue of A.

Then we have that

$$\Lambda_1 = 1,$$

while all the other eigenvalues of F are strictly inside the unitary circle, that is,

$$|\Lambda_j| < 1,$$

for $j = 2, \ldots, 2N$.

The expressions for Λ_{2i-1} and Λ_{2i} come from directly calculating the eigenvalues of M_i in (2.19).[28]

For $i = 1$, we have that $\lambda_1 = d$ and, hence, $\Lambda_1 = 1$ and

$$\Lambda_2 = 1 - \alpha + \alpha\frac{-1+d\rho}{1+d\rho} < 1.$$

Now consider the cases $i = 2, \ldots, N$. First assume that $\rho^2(\lambda_i^2 - d^2) + 1 > 0$. Since $|\lambda| < d$ then $\rho^2(\lambda_i^2 - d^2) < 0$ and, in turn, $\rho^2(\lambda_i^2 - d^2) + 1 < 1$. Hence,

$$-1 < \frac{\lambda_i\rho \pm \sqrt{\rho^2\left(\lambda_i^2 - d^2\right)+1}}{1+d\rho} < 1$$

which implies that the corresponding eigenvalues Λ_{2i-1} and Λ_{2i} are strictly inside the unitary circle.[28]

Now assume that $\rho^2(\lambda_i^2 - d^2) + 1 < 0$. Then $\frac{\lambda_i\rho \pm \sqrt{\rho^2(\lambda_i^2-d^2)+1}}{1+d\rho}$ is a complex number such that

$$\left| \frac{\lambda_i\rho \pm \sqrt{\rho^2\left(\lambda_i^2 - d^2\right) + 1}}{1 + d\rho} \right| = \frac{\sqrt{\rho^2 d^2 - 1}}{1 + \rho d} < 1.$$

This concludes the proof.

Based on the result stated above and on the result of Proposition 1 in [27] we have that the optimal convergence rate of the relaxed-ADMM, denoted as R, is given by

$$R = \min_{\rho>0, 0<\alpha<1} \max\left\{|\Lambda_j|,\ \ j = 2, \ldots, 2N\right\}. \tag{2.22}$$

Having explicit expressions for the eigenvalues Λ_i, $i = 2, \ldots, 2N$, allows for implementing efficient numerical procedure to compute R.[28]

2.6 Numerical Simulation

2.6.1 The ρ parameter

We simulating the behavior of the parameter ρ for fixed values of α for different number of nodes:[28]

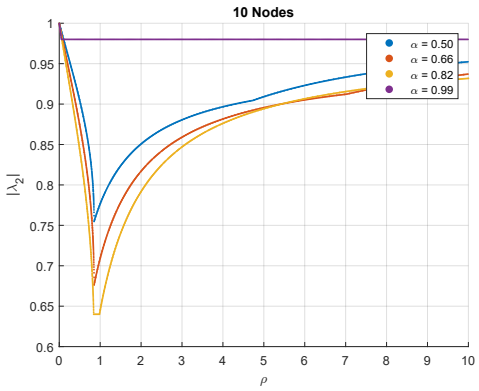

Figure 2.2: Best $|\lambda_2|$ for given ρ, $N = 10$

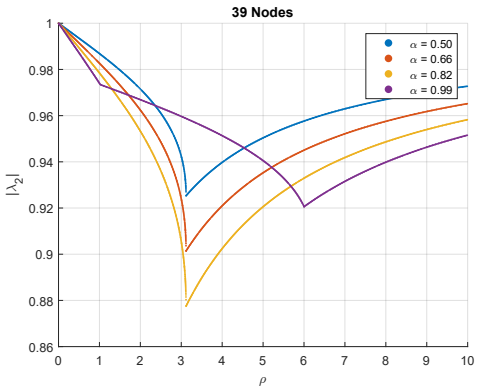

Figure 2.3: Best $|\lambda_2|$ for given ρ, $N = 39$

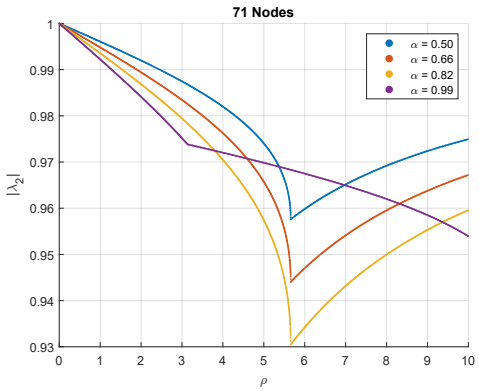

Figure 2.4: Best $|\lambda_2|$ for given ρ, $N = 71$

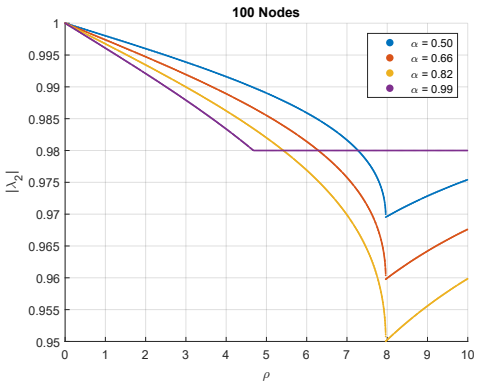

Figure 2.5: Best $|\lambda_2|$ for given ρ, $N = 100$

The figures show examples for both odd and even number of nodes since their behavior is different, attention to the minimal values of the red/blue/yellow curves, their ρ value seems to be the same.

The minimal value of $|\lambda_2|$ in each graph seems to be independent from the choice of α. For the following figures we will take advantage of this fact to get the location and possible formulation of the ρ hyper-parameter. [28]

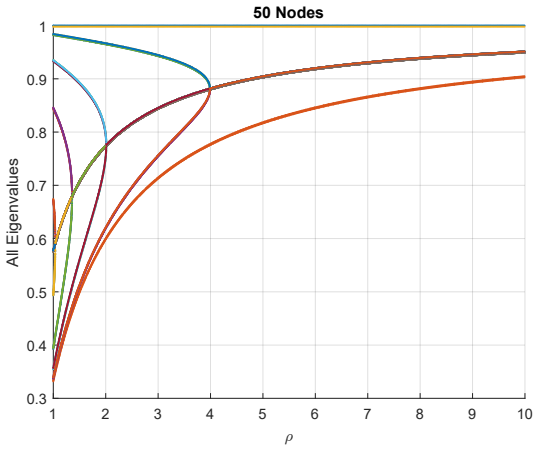

Figure 2.6: Paths traced by all the eigenvalues for variable ρ, with $\alpha = 0.999$, $N = 50$

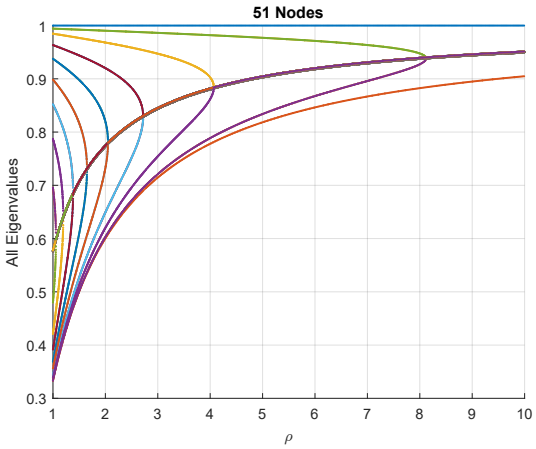

Figure 2.7: Paths traced by all the eigenvalues for variable ρ, with $\alpha = 0.999$, $N = 51$

The preceding plots are rich in information. First we notice the clear differences between odd and even number of nodes. Then through an exhaustive search it was determined that:

- The minimal $|\lambda_2|$ is located at the crossing between the peaks and the middle line.
- This minimum occurs in the right-most peak for the graphs of even number of nodes and occurs in the second to last peak for the graphs of odd node-number. This is not obvious and was confirmed later analytically.
- It is possible and very simple to characterize where this minimum occurs. The minimum $|\lambda_2|$ occurs in specific eigenvalue curves, and is located where the eigenvalue transitions from real to complex. In the following paragraphs we elaborate further.

Before delving deeper recall the eigenvalues of a given network with regular topology:

$$\tilde{\lambda}_{j+1} = 2\sum_{l=1}^{d/2} \cos\left(\frac{2\pi}{n} lj\right), \qquad j = 0, 1, \ldots, n-1, \qquad d = 2, 4, 6, 8, \ldots \tag{2.23}$$

where d represents the regular degree and $j+1$ is the corresponding eigenvalue index $\{1, \ldots, n\}$

Recall also that the closed-form eigenvalues of the R-ADMM are:

$$\lambda_{i,i+1} = 1 - \alpha + \frac{\alpha}{1+d\rho}\left(\rho\tilde{\lambda}_{j+1} \pm \sqrt{\rho^2(\tilde{\lambda}_{j+1}^2 - d^2) + 1}\right) \tag{2.24}$$

The curves containing the minimum $|\lambda_2|$ are

- For graphs with an even number of nodes: $\tilde{\lambda}_{2,n}$ using the positive root of Eq.2.24 and $\tilde{\lambda}_{\frac{n}{2},\frac{n}{2}+2}$ using the negative root of Eq.2.24
- For graphs with an odd number of nodes: $\tilde{\lambda}_{2,n}$ using the positive root of Eq.2.24 and $\tilde{\lambda}_{\lceil\frac{n}{2}\rceil,\lceil\frac{n}{2}\rceil+1}$ using the negative root of Eq.2.24

To generalize better, we will consider only the curve generated by $\tilde{\lambda}_2$ with the positive root of Eq.2.24 which is common for both even and odd cases and moreover, agrees with the characterization of the convergence of matrices in terms of their second eigenvalue.

As stated before, a closer look at these curves reveals that the minimum occurs when the eigenvalue transitions from a purely real number to a complex one. Given that $\rho > 0$ and that $\tilde{\lambda}_2 < d$, the square root term of Eq. 2.24 is ever decreasing and will dictate the transition of λ_2 as it passes through zero; hence, we need:

$$\rho^2(\tilde{\lambda}_2^2 - d^2) + 1 = 0 \implies \rho^* = \frac{1}{\sqrt{d^2 - \tilde{\lambda}_2^2}} \tag{2.25}$$

To raise confidence in our analysis we perform a grid-search for both α and ρ and compare again

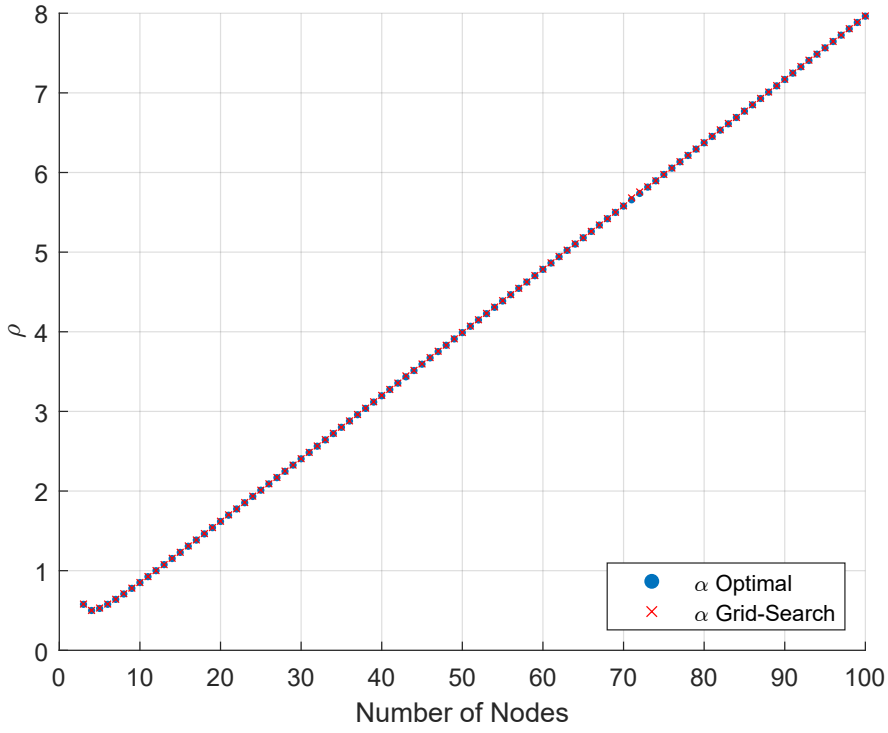

Fig. 2.6.1 confirms our analysis.

2.6.2 The α parameter

The search for the optimal α will follow a similar path to that of ρ. We start by looking at how alpha varies for the fixed values of ρ and for different number of nodes, odd and even :

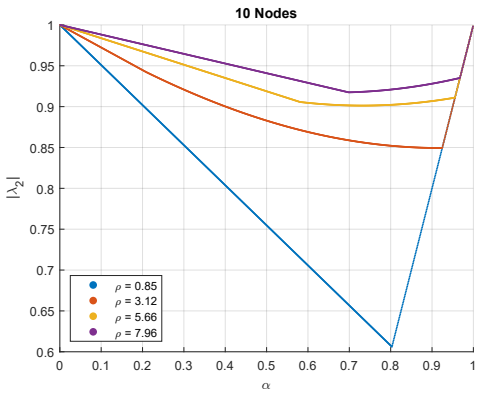

Figure 2.8: Best $|\lambda_2|$ for given α, $N = 10$

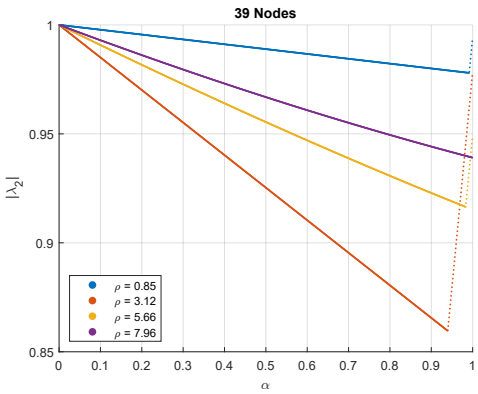

Figure 2.9: Best $|\lambda_2|$ for given α, $N = 39$

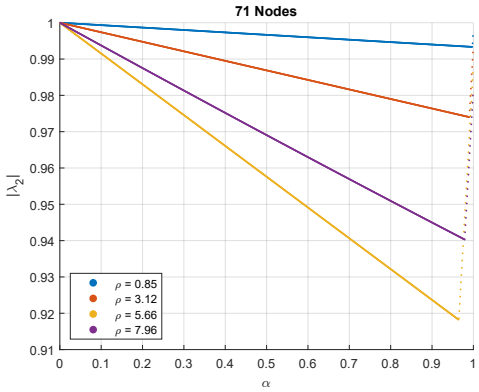

Figure 2.10: Best $|\lambda_2|$ for given α, $N = 71$

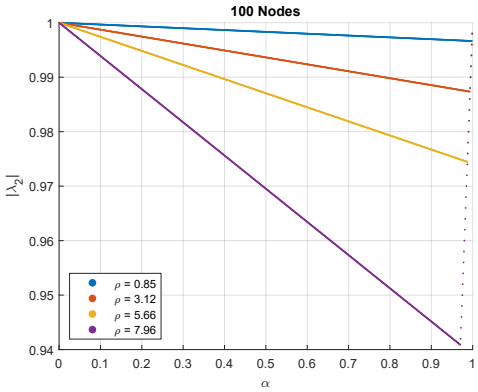

Figure 2.11: Best $|\lambda_2|$ for given α, $N = 100$

Notice how there exist in fact a best value for each curve. By observation, we take notice of the minimal value of $|\lambda_2|$ and its corresponding best α. Next, we will search for a condition related to the eigenvalues in Eq. 2.24, for this purpose we observe how all the eigenvalues of a particular number of nodes behave for the **optimal** ρ computed with Eq.2.25:

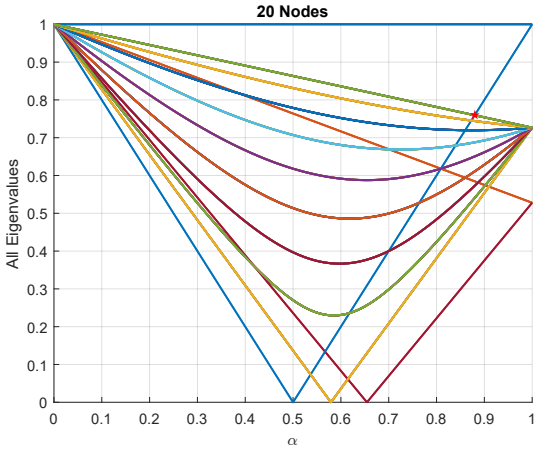

Figure 2.12: Paths traced by all the eigenvalues for variable α, with ρ^*, $N = 20$

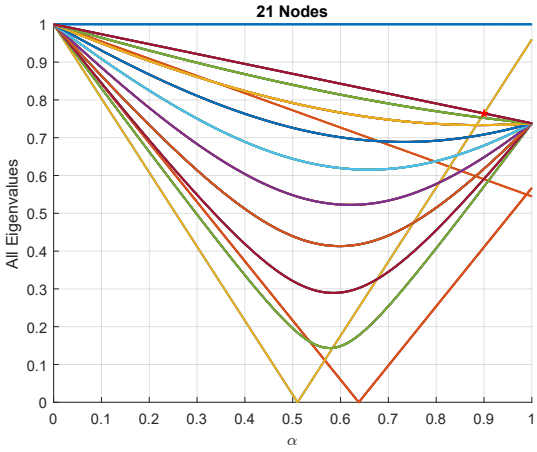

Figure 2.13: Paths traced by all the eigenvalues for variable α, with ρ^*, $N = 21$

The minimum magnitude that the eigenvalues attain is marked by a red star in Figures 11 and 12. By a process of exhaustive search it was determined that this point is the crossing of specific eigenvalue curves.

- For networks with an even number of nodes: it is the crossing of the eigenvalue curves $\tilde{\lambda}_{2,n}$ using either of the roots of Eq.2.24, and $\tilde{\lambda}_{\frac{n}{2}+1}$ with the negative root of Eq.2.24
- For networks with an odd number of nodes: it is the crossing of the eigenvalue curves $\tilde{\lambda}_{2,n}$ using using either of the roots of Eq.2.24, and $\tilde{\lambda}_{\lceil\frac{n}{2}\rceil,\lceil\frac{n}{2}\rceil+1}$ with the negative root of Eq.2.24

For the sake of consistency we chose $\tilde{\lambda}_2$ along with the positive root of Eq.2.24 and $\tilde{\lambda}_{\lceil\frac{n}{2}\rceil+1}$ with the negative root of Eq.2.24 in order to be able to generalize for both odd and even number of nodes.

We then find the crossing of this equations recalling that the curves that cross are actually the magnitude of the eigenvalue they trace, then:

$$\sqrt{\left(1-\alpha+\frac{\alpha\rho^*\tilde{\lambda}_2}{1+d\rho^*}\right)^2}=\sqrt{\left[1-\alpha+\frac{\alpha}{1+d\rho^*}\left(\rho^*\tilde{\lambda}_{\lceil\frac{n}{2}\rceil+1}-\sqrt{\rho^{*2}(\tilde{\lambda}^2_{\lceil\frac{n}{2}\rceil+1}-d^2)+1}\right)\right]^2} \tag{2.26}$$

Some remarks are in order:

- The sides of the equation differ only in the eigenvalues that they use, notice moreover that ρ^* is used.
- The LHS of Eq.2.26 has simplified to a purely real number since ρ^* will make the square root zero if we are using the $\tilde{\lambda}_2$ eigenvalue
- The RHS is also purely real, reason for this is given in the following paragraph.

Reorganizing the terms under the square root using Eq.2.25:

$$\rho^{*2}(\tilde{\lambda}^2_{\lceil\frac{n}{2}\rceil+1}-d^2)+1=(-1)\frac{\tilde{\lambda}^2_{\lceil\frac{n}{2}\rceil+1}-d^2}{\tilde{\lambda}^2_2-d^2}+1 \tag{2.27}$$

The periodicity of the eigenvalues of the network in Eq. 2.23 helps stablish that:

$$\tilde{\lambda}^2_2<\tilde{\lambda}^2_{\lceil\frac{n}{2}\rceil+1}\leq d^2 \tag{2.28}$$

This renders the ratio on the RHS of Eq.2.27 a postive number smaller than 1. Therefore the whole expression will be always positive and the corresponding eigenvalue will be real. Given this fact we proceed to solve equation 2.26 and after some lengthy algebra we arrive to:

$$\begin{aligned}
\alpha^* &= \frac{A}{A+B} \\
A &= R+2\rho^*\left(\tilde{\lambda}_2-\tilde{\lambda}_{\lceil\frac{n}{2}\rceil+1}\right) \\
B &= \frac{1}{1+d\rho^*}\left[1-R\rho^*\tilde{\lambda}_{\lceil\frac{n}{2}\rceil+1}+\rho^{*2}\left(2\tilde{\lambda}^2_{\lceil\frac{n}{2}\rceil+1}-\tilde{\lambda}^2_2-d^2\right)\right] \\
R &= 2\sqrt{\rho^{*2}\left(\tilde{\lambda}^2_{\lceil\frac{n}{2}\rceil+1}-d^2\right)+1}
\end{aligned} \tag{2.29}$$

To confirm our analysis we plot the best α found by grid-search against our proposed closed form in Eq.2.29:

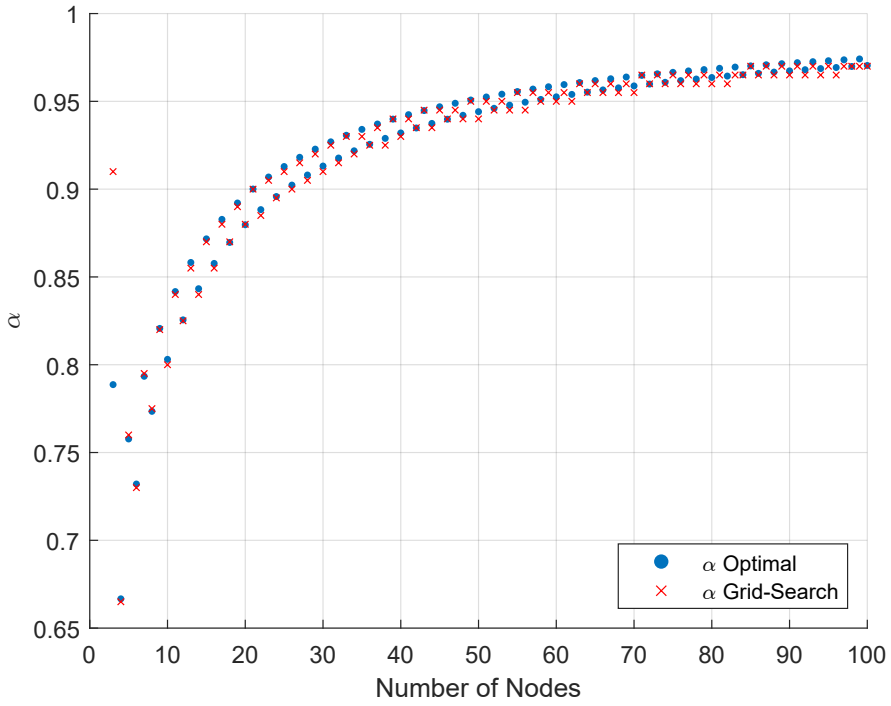

Figure 2.6.2 confirms our analysis.

2.7 Discussion

We restrict our analysis only to the figures below. In Fig.2.14 observe how R changes with respect to the variation of $\alpha \in (0, 1)$ for different values of ρ and fixed N. Regardless of the various values of ρ a better value of R is achieved for $\alpha > 0.5$. So, R holds a convex shape with respect to α, with the minimizer lying on $\alpha > 0.5$.

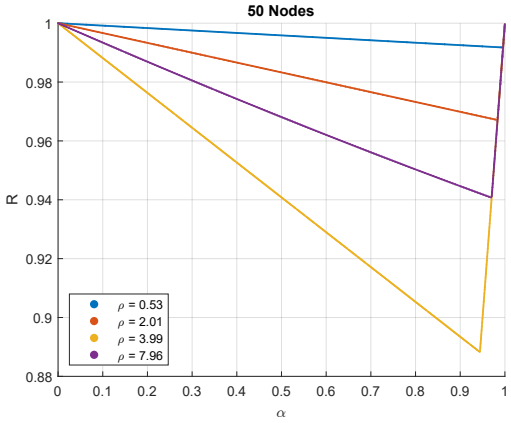

Figure 2.14: Value of R as α varies in a $d = 2$ graph.

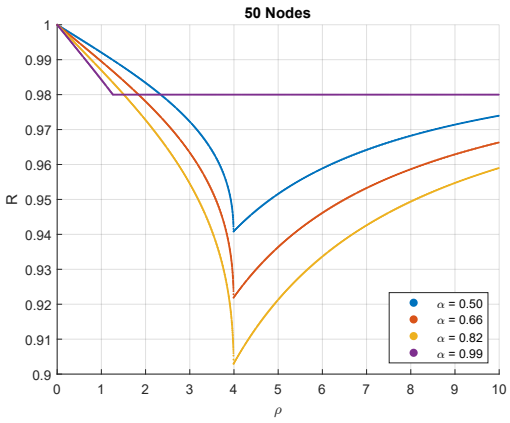

Figure 2.15: Value of R as ρ varies in a $d = 2$ graph.

With respect to ρ, Fig.2.15 displays its behavior with regards to a fixed N and increasing values of α. Note that the optimal value of ρ remains constant for most of the range of alpha, changing only when α is very close to 1.

To explore further, we report in the table below verbatim the grid-search over the hyper-parameter space to determine, given N and d, the value of R and the corresponding optimal pair (α^*, ρ^*) that was done in [28].

As both the number of nodes and the degree of the graphs increase, the optimal α^* achieving R, approaches 1. To compare against the standard ADMM we fix $\alpha = 0.5$ and with

	R		α^*		ρ^*	
N	d=2	d=4	d=2	d=4	d=2	d=4
5	0.3623	0.1270	0.7577	0.8873	0.5257	0.2582
50	0.8882	0.8188	0.9441	0.9998	3.9894	1.2643
100	0.9409	0.9053	0.9704	0.9999	7.9630	2.5210
150	0.9598	0.9359	0.9799	0.9999	11.9401	3.7777
200	0.9695	0.9515	0.9848	0.9999	15.9181	5.0352

Table 2.1: Optimal Values for graphs with $d = 2, 4$ and increasing N[28]

a grid-search we find the optimal for ρ in terms of the minimization of R. Fig.2.16 shows the results for graphs of degree $d = 2$. [28]

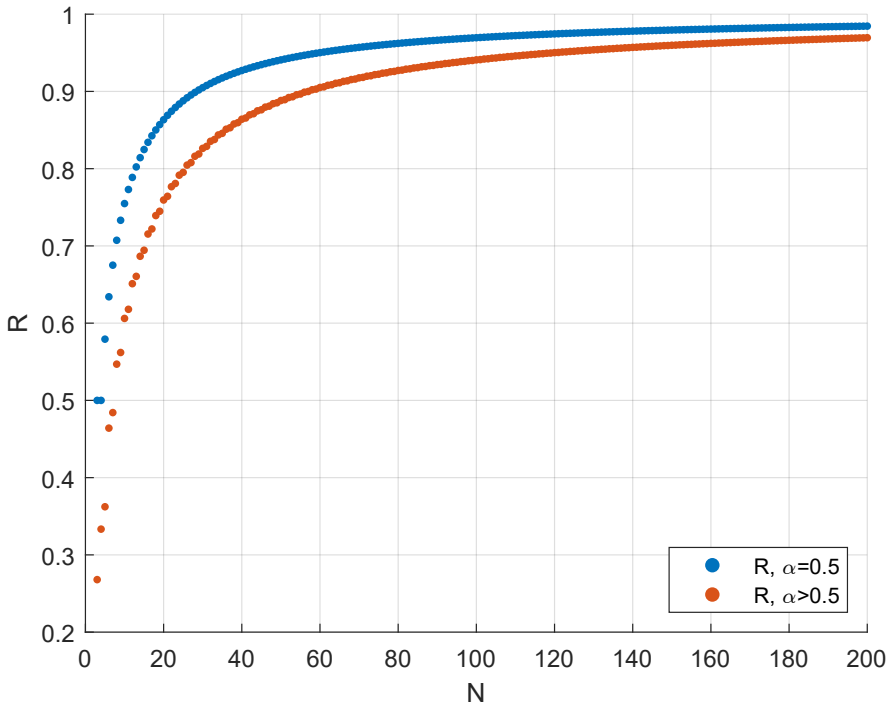

Figure 2.16: Best R as N in creases, a comparison between $\alpha = 0.5$ and $\alpha > 0.5$

From Fig.2.16, the superiority of the relaxed ADMM algorithm is evident with respect to the classical ADMM.

Chapter 3

On Byzantine Resilience

Byzantine faults, also known as Byzantine failures or attacks, represent the **most general and challenging type of adversarial behavior** in distributed systems[29]. Unlike simpler **crash faults**, where a malfunctioning node merely ceases to operate or transmit data, Byzantine nodes can behave arbitrarily and maliciously. This means they can send **arbitrary, false, or tampered messages**, **collude with other faulty agents** to achieve their malicious goals, they can also possess **complete knowledge of the system**, including data, algorithms, and even random bits generated by the parameter server and **change their behavior** across different iterations or communication rounds[30].

The Byzantine threat model originated from the Byzantine Generals Problem, which established the impossibility of safeguarding against traitors (Byzantine nodes) when their number is not more than one-third of the total loyal nodes. However, for statistical inference and machine learning, this critical fraction can be higher[31].

The rise of large-scale and distributed machine learning, particularly in paradigms like **Federated Learning** has brought security and robustness to the forefront[29]. In these settings, **data is generated and stored at different locations**, and transmitting all locally collected data to other centers might be constrained by transmission capacity or privacy concerns[32]. Individual computing units are **more unpredictable and susceptible to malicious or coordinated attacks**. Traditional machine learning algorithms and systems often **assume a non-adversarial setting**, leading to fragility when faced with Byzantine failures[31].

The primary goal of developing Byzantine-resilient algorithms is to **ensure accurate learning despite adversarial interruptions**, aiming for optimal performance while maintaining communication efficiency[33].

Byzantine attacks present significant challenges that prevent standard distributed optimization from achieving its goals:

- Arbitrary Skewing of Results: A single Byzantine agent can completely skew the average value of aggregated gradients, preventing convergence or leading to arbitrary outcomes[30].
- Theoretical Analysis: The arbitrary behavior of Byzantine machines creates complicated and unspecified probabilistic dependencies across iterations and aggregated gradients, making theoretical analysis difficult[33].
- Impossibility of Exact Optimization: In the presence of crash or Byzantine faults, optimizing a global cost function exactly becomes impossible. Weaker versions of the problem must be defined and solved[29].
- Computational Cost in High Dimensions: Some robust approaches from distributed consensus, while theoretically sound, incur prohibitive computational costs in the high-dimensional parameter spaces common in machine learning[34].

Many conventional methods for distributed optimization prove ineffective against Byzantine attacks. Current approaches based on linear combinations of worker vectors, like **simple averaging**, cannot tolerate even one Byzantine failure. A Byzantine worker can force the parameter server to choose any arbitrary vector, preventing convergence[34]. Decentralized systems, where agents communicate peer-to-peer without a central server (federated learning), introduce further challenges to Byzantine resilience. **Disagreement**, many robust aggregation rules proven effective in the federated setting (e.g., Coordinate-wise Median, Geometric Median, Krum) may **fail to reach consensus** in decentralized scenarios, even without Byzantine workers. Then, the agents will fail to converge to a common value[32]. Non-Doubly Stochastic Virtual Mixing Matrix, if the effective "mixing matrix" between honest workers after aggregation is not doubly stochastic, it leads to a **larger asymptotic learning error**, as the system may only minimize a convex combination of local cost functions rather than the true average[32].

3.1 Problem Formulation

We consider a distributed computing system modeled as an undirected connected graph $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{1, 2, \ldots, N\}$ represents the set of nodes $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ defines the communication links between nodes. Then for each node $i \in \mathcal{V}$, its communication neighborhood is defined as $\mathcal{N}_i = \{j \in \mathcal{V} \mid (i, j) \in \mathcal{E}\}$

Each node i maintains a private, strongly convex objective function $f_i : \mathbb{R}^d \to \mathbb{R}$ that operates on a shared optimization variable $x \in \mathbb{R}^d$. The global optimization objective is:

$$\arg\min_{x} \sum_{i=1}^{N} f_i(x) \tag{1}$$

To enable distributed computation, we introduce local copies of the optimization variable. Let x_i denote the local copy maintained by node i. The original problem can be then reformulated as:

$$\arg\min_{x_1,\ldots,x_N} \sum_{i=1}^{N} f_i(x_i) \tag{2}$$

subject to the consensus constraints:

$$x_i = x_j \quad \forall (i, j) \in \mathcal{E} \tag{3.1}$$

The consensus constraints ensure that neighboring nodes maintain identical local variables, making the reformulated problem equivalent to the original formulation. A distributed solution approach employs a gradient tracking algorithm with the following update rules for each node i:

$$x_i^{k+1} = \sum_{j \in \mathcal{N}_i \cup \{i\}} w_{ij} x_j^k - \alpha z_i^k z_i^{k+1} \qquad = \sum_{j \in \mathcal{N}_i \cup \{i\}} w_{ij} z_j^k + \nabla f_i(x_i^{k+1}) - \nabla f_i(x_i^k) \tag{3.2}$$

where z_i is an auxiliary variable that tracks gradient information, ∇f_i represents the gradient of function f_i, w_{ij} are elements of a doubly stochastic matrix W that respects the graph topology, $w_{ij} = 0$ if $(i, j) \notin \mathcal{E}$, α is the step size parameter. The matrix W can be constructed using standard techniques such as the Metropolis-Hastings method. This algorithm requires only first-order gradient information and facilitates information exchange exclusively between neighboring nodes.

A speed-improved version incorporating second-order information follows a GIANT-like approach:

$$x_i^{k+1} = \sum_{j \in \mathcal{N}_i \cup \{i\}} w_{ij} x_j^k - \alpha [\nabla^2 f_i(x_i^k)]^{-1} z_i^k z_i^{k+1} \quad = \sum_{j \in \mathcal{N}_i \cup \{i\}} w_{ij} z_j^k + \nabla f_i(x_i^{k+1}) - \nabla f_i(x_i^k) \tag{3.3}$$

where $\nabla^2 f_i$ denotes the Hessian matrix of function f_i. This second-order method typically exhibits superior convergence properties compared to first-order approaches.

The network will operate under adversarial conditions, meaning Byzantine attacks may occur. During the optimization process, a subset of nodes may malfunction or act maliciously, they will be called the set of Byzantine workers. These Byzantine nodes can deviate from the prescribed algorithmic protocol either permanently or intermittently. In consequence, We partition the node set as $\mathcal{V} = \mathcal{L} \cup \mathcal{M}$, where $\mathcal{L}$ represents the set of honest, well-functioning nodes $\mathcal{M}$ represents the set of Byzantine (malicious or faulty) nodes. Our primary goal is to develop a robust variant of the algorithms presented in equations (3.3) or (3.2) that enables honest nodes to: Identify and isolate Byzantine nodes in their neighborhoods, aggregate information exclusively from verified honest neighbors and maintain convergence to the optimal solution despite the presence of Byzantine adversaries. This approach will ensure the reliability and security of distributed optimization in adversarial network environments.

3.2 Common Byzantine Attacks

The Gaussian attack is a type of classic Byzantine attack where malicious workers replace their computed gradients or messages with values sampled from a Gaussian distribution. This attack is often employed in experiments to evaluate the robustness of Byzantine-tolerant algorithms by introducing random characterizable noise into the system. In an experimental setup, this attack involves replacing gradient/message vectors with independent and identically distributed (i.i.d.) random vectors. These vectors are drawn from a Gaussian distribution, typically with an specified mean and standard deviation. [35]

For a Byzantine worker b in the set of Byzantine neighbors B_n for worker n, its message $\tilde{x}_{b,n}^k$ at iteration k could be generated as:

$$\tilde{x}_{b,n}^k \sim \mathcal{N}\left(\bar{x}_n^k, \sigma_{attack}^2\right) \tag{3.4}$$

where $\bar{x}_n^k$ might represent the mean of honest messages, and σ_{attack} is the attack's standard deviation. This attack aims to corrupt the aggregated gradient by injecting random noise, degrading the model's performance and potentially its convergence.[32]

Inverse/Sign-Flipping Attack involve Byzantine agents manipulating their messages to point in the opposite direction of the desired/true update. The goal is to prevent the learning algorithm from converging by consistently pushing the model parameters away from the optimal solution. In a basic sign-flipping scenario, a Byzantine worker $b \in B_n$ might set its message $\tilde{x}_{b,n}^k$ to be the negative of the weighted average of messages from its honest neighbors [conversation history]:

$$\tilde{x}_{b,n}^k = -\bar{x}_n^k$$

where $\bar{x}_n^k$ is the weighted average of messages from honest neighbors.[32]

The inverse/sign-flipping attack is highly disruptive. It prevents the parameter server from taking steps in the correct direction, leading to a breakdown of the learning process. Numerical results in [32] show that naive aggregation methods like averaging are impacted, showing very low accuracy or failure to converge to an optimal solution.[34]

A **Constant attack** is a disruption caused by the byzantine neighbors in the form of a constant message. They could mimic a large value aiming to throw the learning process out of synchronization or simply a constant value coming from a faulty worker.

$$\tilde{x}_{b,n}^k = C \qquad \forall k \tag{3.5}$$

where C is a constant.

The **A Little Is Enough (ALIE)** attack has been specially designed to circumvent defenses for distributed learning. It implies an attack strategy that is effective even with limited malicious intervention, subtly derailing the learning process. Its effectiveness is demonstrated through numerical experiments in [32],[29], comparing its impact against other attack types.

These Byzantine attacks underscore the critical need for robust aggregation rules and resilient algorithms in distributed and decentralized machine learning systems.

3.3 Algorithms and Robust Aggregation Rules

Robust aggregation rules are a common tool to safeguard the learning process from Byzantine attacks. These rules enable healthy agents to mitigate, to a certain extent, adversarial influences and ensure the system's resilience.

The **coordinate-wise median (CooMed)** aggregation rule involves each worker in a decentralized setting computing the median value for each individual dimension of the received gradient vectors or model parameters. For a set of vectors, the k-th coordinate of the aggregated output is simply the standard one-dimensional median of the k-th coordinates of all input vectors. This effectively treats each dimension independently.

If we have a set of S input vectors $x_1, \ldots, x_S$, each in $\mathbb{R}^d$, the coordinate-wise median $g = \text{med}\{x_i : i \in [S]\}$ is a vector where its k-th coordinate g_k is calculated as:

$$[g]_k = \text{Median}([x_1]_k, \ldots, [x_S]_k) \tag{3.6}$$

It can achieve an error rate of $\tilde{O}(\frac{\alpha}{\sqrt{n}} + \frac{1}{\sqrt{nm}} + \frac{1}{n})$ for strongly convex loss functions, which is order-optimal when the number of local data points n is sufficiently large compared to the number of machines m** ($n \gg m$). An advantage is that it does not require knowledge of the fraction α of Byzantine machines. However, the CM requires assumptions of bounded variance and bounded absolute skewness of the gradient. In decentralized settings, while successful in i.i.d. cases, the CM can become unsatisfactory for non-i.i.d. data distribution. It can also suffer from a "disagreement" issue, where honest workers may not reach consensus, and it is not considered to be a "Robust Contractive Aggregation" [32] because its contraction constant ρ is generally greater than 1, leading to potentially large asymptotic learning errors. The computational complexity for the screening procedure is $O(Md)$, where M is the number of nodes and d is the dimension of the parameter vector. For details on this formulation we refer to [32]

The **Geometric median** is a generalization of the median to multiple dimensions. It aims to find a point that minimizes the sum of Euclidean distances to all input messages. Unlike the mean, which minimizes the sum of squared Euclidean distances, the geometric median minimizes the sum of Euclidean distances. The geometric median is not necessarily one of the input points and is unique unless all points lie on a line.

Given a multi-set of S vectors $\{x_1, \ldots, x_S\} \subseteq R^D$, their geometric median, denoted geo_med$\{x_1, \ldots, x_S\}$, is defined as:

$$\text{geo\_med}\{x_1, \ldots, x_S\} = \arg\min_{x \in R^D} \sum_{n=1}^{S} \|x - x_n\| \tag{3.7}$$

The **GeoMed** has a breakdown point of 0.5[36], meaning that it can tolerate up to 50% of arbitrary data corruption before its estimate can be arbitrarily distorted. This makes it highly robust. While effective in federated scenarios, directly extending the GeoMed to decentralized settings may lead to a "disagreement" issue where honest workers cannot reach consensus, and it is not considered to be an RCA Robust Contractive Aggregation[32]. Computing the exact geometric median is expensive. However, a $(1+\gamma)$-approximate geometric median can be computed efficiently, and this approximation is also robust. The screening computational complexity for the GeoMed is cited as $O(Md + bd\log^3(1/\gamma))$ for a $(1+\gamma)$-approximate version.

Krum is a robust update rule designed for use in Gradient Descent implementations in a Byzantine worker scenario. It works by selecting the gradient vector proposed by a worker that is "closest to everyone else"[32] in the hopes of being closest to its non-faulty neighbors.

For each worker i, Krum calculates a "score" $s(i)$ by summing the squared distances to its $m-f-2$ closest neighbours (where m is the total number of workers and f is the number of Byzantine workers). The worker whose vector $\tilde{v}_k$ yields the minimal score is then chosen:

$$\text{Krum}(\{\tilde{v}_i : i \in [m]\}) = \tilde{v}_k \tag{3.8}$$

$$k = \arg\min_{i\in[m]} \sum_{i\to j} \|\tilde{v}_i - \tilde{v}_j\|^2 \tag{3.9}$$

Here, $i \to j$ means v_j is among the $m-f-2$ closest vectors to v_i. Multi-Krum is an extension where instead of selecting just one vector, it **iteratively selects m vectors using the Krum function and then averages them. **Krum** is proven to be (α, f)-Byzantine resilient under the condition that $2b+2 < n$ where b is the number of byzantine attackers incoming to a node with n neighbors including self-information. An advantage of using Krum is that it excludes the vectors that are too far away, avoiding issues where Byzantine workers collude to move the barycenter of the true information away from correct vectors. However, Krum (and Multi-Krum) primarily addresses the classic Byzantine model where faulty workers are entirely malicious. It is not "dimensional Byzantine resilient", meaning it can fail when Byzantine values are spread across different dimensions rather than concentrated in a few workers. Experimental results in [32] show that Krum performs well against "Gaussian" and inverse attacks. However, under "bit-flip", a per-dimension inverse attack, Krum-based algorithms can fail to converge or converge to a suboptimal solution. The computational complexity of Krum is $O(n^2 \cdot (d + \log n))$, which can be high for very high-dimensional parameter vectors, multi-Krum has a similar computational cost.

Iterative Outlier Scissor (IOS) is a recent robust aggregation for decentralized stochastic optimization. It overcomes two major challenges faced by existing robust aggregation: **disagreement** among honest workers and **non-doubly stochastic virtual mixing matrices**. For an honest worker n, IOS iteratively discards messages from its neighbors. What follows is a qualitative description of the algorithm in [32].

1. It begins by constructing a **doubly stochastic and symmetric mixing matrix** W'.
2. In each inner iteration, it maintains a "trusted set" of received models, initially including all neighbors (honest and Byzantine) and itself.
3. It computes the weighted average of the models in the current trusted set.
4. It then **discards the model that is farthest away from this weighted average** (excluding its own model).
5. This process repeats for a number of iterations equal to the estimated number of Byzantine neighbors (q_n), after which the final weighted average of the remaining "trusted models" is outputted as the aggregated value.

The aggregation at worker n takes the form:

$$A_n(x_n, \{\tilde{x}_{m,n}\}_{m\in N_n\cup B_n}) = (1-r_n) \sum_{m\in U_n^{(q_n)}\setminus\{n\}} w''_{nm}\tilde{x}_{m,n} + r_n x_n \tag{3.10}$$

where $U_n^{(q_n)}$ is the final trusted set from the iterations described above, and r_n and w''_{nm} are specific weights decided after a rescaling. IOS directly addresses the issues of disagreement where workers fail to reach consensus, and of non-doubly stochastic virtual mixing matrices which lead to larger asymptotic learning errors. It is both a Robust Contractive Aggregation

(RCA) and Robust Doubly Stochastic Aggregation (RDSA), meaning it aims for a small contraction constant ρ and a doubly stochastic virtual mixing matrix W. This is crucial for reducing estimation, mixing, and consensus errors. * Unlike some other methods, the parameters for IOS (q_n) are estimates of the number of Byzantine neighbors, which are assumed to be known in practice. Its contraction factor, as defined in [32] is bounded by $O(\frac{\mu}{1-3\mu})$, where μ is the proportion of Byzantine workers, which can satisfy the RCA condition.

These robust aggregation rules represent different approaches to ensuring stability and convergence in distributed and decentralized machine learning systems facing the complex challenge of Byzantine failures. Their effectiveness varies depending on the specific attack model, network topology, and data distribution.

3.4 Trust Based Medoid Evaluation

3.4.1 The Soft Medoid

The Soft Medoid is proposed as a **differentiable generalization of the traditional Medoid**. Unlike the mean, which minimizes the sum of squared L2 distances, the Medoid identifies a data point within the input set that minimizes the sum of Euclidean distances to all other input points, acting as an intuitive central point. However, the classic Medoid is constrained to be one of the original input data points, which may not represent the true center if all inputs are distant from it. The Soft Medoid overcomes this limitation by employing a **softmax function to compute a weighted average** of all input data points. This allows the aggregated point to reside anywhere within the **convex hull** of the input data points, offering greater flexibility.

In an iterative algorithm where a node i receives information $\{x_j(k)\}$ from its neighbors $j \in N_i \cup \{i\}$ (with $m = |N_i \cup \{i\}|$ messages), the Soft Medoid is applied at each iteration to perform this robust aggregation.

For a collection of input data points $X = \{x_1, \ldots, x_n\} \subseteq \mathbb{R}^d$, the **Soft Medoid** $t_{SM}(X)$ is defined as a weighted average:

$$\mathbf{x_{sm}}(\mathbf{X}) = \sum_{i=1}^{n} \hat{s}_i \mathbf{x}_i \tag{3.11}$$

where $\hat{s}_i$ are the normalized weights derived from the softmax function, based on distances to other data points:

$$\hat{s}_i = \frac{\exp\left(-\frac{1}{T}\sum_{j=1}^{n} \|\mathbf{x}_j - \mathbf{x}_i\|\right)}{\sum_{q=1}^{n} \exp\left(-\frac{1}{T}\sum_{j=1}^{n} \|\mathbf{x}_j - \mathbf{x}_q\|\right)} \tag{3.12}$$

Here, $T \in (0, \infty)$ is a **temperature parameter** that controls the steepness of the softmax approximation between the standard medoid and a weighted average. A point with smaller aggregate distances to other data points will receive a larger $\hat{s}_i$ weight, while remote points will have weights close to zero.

When incorporating pre-existing non-negative weights $\mathbf{a} = [a_1, \ldots, a_n]^\top$, the concept extends to the **Weighted Soft Medoid** $\tilde{x}_{WSM}(X, a)$:

$$\tilde{\mathbf{t}}_{\mathbf{WSM}}(\mathbf{X}, \mathbf{a}) = c(\mathbf{s} \circ \mathbf{a})^\top \mathbf{X} \tag{3.13}$$

where $\circ$ denotes the element-wise Hadamard product, and the weights s_i are calculated as:

$$s_i = \frac{\exp\left(-\frac{1}{T}\sum_{j=1}^{n} a_j \|\mathbf{x}_j - \mathbf{x}_i\|\right)}{\sum_{q=1}^{n} \exp\left(-\frac{1}{T}\sum_{j=1}^{n} a_j \|\mathbf{x}_j - \mathbf{x}_q\|\right)} \tag{3.14}$$

Where c is a normalization constant,

$$c = \frac{\sum_{j=1}^{n} a_j}{\sum_{j=1}^{n} s_j a_j} \tag{3.15}$$

Let the term:

$$\mathbf{a}'_j = cs_j a_j \tag{3.16}$$

represent the **soft-medoid weights**, which indicate the relative weighting assigned to each input vector $\mathbf{x}_j$. As before, a smaller $\mathbf{a}'_j$ suggests that $\mathbf{x}_j$ is more likely an outlier. The Weighted Soft Medoid adapts to the weighted nature of robust aggregation schemes in a decentralized setting, with $\mathbf{s} \circ \mathbf{a}$ acting as a weighted mean.

The Soft Medoid has several properties contributing to its robustness against adversarial interference. The temperature parameter T influences the Soft Medoid's behavior, as $T \to 0$, the Soft Medoid approaches the **exact Medoid**, making it focused on a central point among the inputs. Conversely, as $T \to \infty$, it tends towards the **sample mean**, becoming less robust. Heuristic analysis done in [37] suggests that a value of $T \in [0.2, 1]$ typically offers good robustness, enabling the Soft Medoid to average over non-malicious data points.

A fundamental measure of a location estimator's robustness is its **breakdown point**, which quantifies the maximum fraction of perturbed data points it can tolerate before its estimate can be arbitrarily distorted. The Soft Medoid possesses a **asymptotic breakdown point of 0.5**. This implies that the aggregated result remains bounded and resistant to arbitrary distortion as long as less than 50% of the input data is corrupted. This property holds true for any finite value of $T \in [0, \infty)$[37]. In the weighted case, the breakdown occurs if the sum of weights for perturbed data ($\sum a_{pert}$) equals or exceeds the sum of weights for clean data ($\sum a_{clean}$).

The formal definition of the breakdown point for an estimator t on a dataset X of size n with m perturbed examples is

$$\epsilon^*(\mathbf{t}, \mathbf{X}) = \min_{1 \le m \le n} \left\{ \frac{m}{n} : \sup_{\tilde{X}_\epsilon} \|\mathbf{t}(\mathbf{X}) - \mathbf{t}(\tilde{\mathbf{X}}_\epsilon)\| = \infty \right\} \tag{3.17}$$

where $\tilde{X}_\epsilon$ denotes the perturbed dataset. The Soft Medoid's estimation guarantee stems from the observation that as perturbations approach infinity, the softmax weights for perturbed samples approach zero, effectively singling them out if the number of clean points outweighs the perturbed ones.

Complementing the breakdown point, the **maxbias curve** describes the maximum possible deviation of the location estimate between clean and perturbed data as a function of the perturbation ratio ϵ, as per the definiton in [37]. The Soft Medoid guarantees a **finite maxbias curve** for $\epsilon < \epsilon^*(t_{SM}, X)$. This means the maximum deviation between an estimate based on clean data and one based on perturbed data remains bounded. Empirical results in [37] suggest that for appropriate T and bounded perturbations, the Soft Medoid can achieve a lower bias than the Medoid, especially when the perturbation is large, as it averages over the clean points.

The Soft Medoid is **orthogonal equivariant**. This means that if the input data points are rotated or translated, the Soft Medoid undergoes the same transformation. This property simplifies theoretical analysis, allowing for assumptions like centered data. Furthermore, the Soft Medoid is effective against **dimensional Byzantine attacks**. In such attacks, malicious values can be spread across different dimensions rather than being concentrated in a few workers. This contrasts with methods like Krum, which primarily address the "classic Byzantine model" where faulty workers are concentrated. The Soft Medoid's nature makes it resilient even when a certain fraction of values within each dimension is corrupted, as long as the number of Byzantine values is less than the number of correct ones for each dimension.

It is now evident that, compared to other robust aggregation methods, the Soft Medoid offers several advantages. Its full differentiability makes it integrable into learning frameworks, unlike the traditional Medoid, it also makes it appealing for theoretical analysis purposes. It has, as well, the best possible breakdown point of 0.5, a strong theoretical guarantee against arbitrary distortions.It is computationally efficient, while Krum has a time complexity of $O(m^2 d)$, the Soft Medoid can achieve $O(nk^2)$ or $O(n)$ in certain configurations, making it

more computationally efficient for large-scale applications[37]. The temperature parameter T allows for a flexible trade-off between strict robustness (lower T) and higher accuracy (higher T), making it adaptable to diverse scenarios. It is effective against various Byzantine attack models, as we will soon see in the numerical results.

The Soft Medoid stands as a robust, differentiable aggregation rule with strong theoretical guarantees against Byzantine failures, driven by its high breakdown point and controlled bias. Its adaptability, enhanced by the temperature parameter and its ability to incorporate explicit information through weights, makes it a valuable tool for building resilient distributed and decentralized machine learning systems.

3.4.2 Trust Variables

It is possible to enhance the detection and rejection capabilities of robust aggregation schemes by adding historical information. For this purposes we can add trust variables which are designed to quantify the trustworthiness of information exchanged between agents[38]. A trust variable, typically denoted as $\alpha_{ij}(k)$ or α_i, represents a **probabilistic measure of trustworthiness** associated with an agent or a transmission.

An **Trust Observation ($\alpha_{ij}(k)$)** is a random variable representing the probability that agent j is a trustworthy neighbor of agent i at a given time instant k. In our work, $\alpha_{ij}(k) \geq 1/2$ suggests that agent j is likely a trustworthy neighbor, while $\alpha_{ij}(k) < 1/2$ indicates that j is likely a Byzantine (malicious) node. An $\alpha_{ij}(k) = 1/2$ means ambiguity, offering no useful information. For Trust Variables to be effective in achieving consensus and performance guarantees, certain assumptions need to be made[39]:

Homogeneity of Trust Variables: The expected values of $\alpha_{ij}(k)$ are constant for transmissions originating from malicious agents and legitimate agents:

$$\begin{aligned} c &= E(\alpha_{ij}(k)) - 1/2 \quad \text{for } i \in L, j \in N_i \cap M \quad \text{(malicious transmissions)} \\ d &= E(\alpha_{ij}(k)) - 1/2 \quad \text{for } i \in L, j \in N_i \cap L \quad \text{(legitimate transmissions)} \end{aligned}$$

Where L/M are the set of trustworthy/malicious actors and N_i are the neighbors of node i. It is needed that $\mathbf{c} \neq \mathbf{d}$. Furthermore, for effective decay of misclassification probabilities, it is generally assumed that $\mathbf{c} < \mathbf{0}$ and $\mathbf{d} > \mathbf{0}$. Implying that, in expectation, $\alpha_{ij}(t)$ values for malicious transmissions are less than $1/2$, and for legitimate transmissions, they are greater than $1/2$. The ability for c and d to be bounded away from zero indicates that the $\alpha_{ij}(t)$ observations contain meaningful trust information.

Independence of Trust Observations: The observations $\alpha_{ij}(k)$ are assumed to be **independent for all time instances** k and for all pairs of agents i and j. Moreover, for any given i and j, the sequence of observations $\{\alpha_{ij}(k)\}$ is identically distributed.

Now, in addition to the instantaneous trust observations, their history can be aggregated into an **accumulated trust variable** $\beta_{ij}(k)$. This cumulative history allows for a more robust classification of agents over time.The accumulated trust variable $\beta_{ij}(k)$ is defined as the sum of past deviations of α_{ij} from the $1/2$ threshold:

$$\beta_{ij}(k) = \sum_{t=0}^{k} (\alpha_{ij}(t) - 1/2) \quad \text{for } k \geq 0, i \in V, j \in N_i \tag{3.18}$$

where V is the set of all nodes.Then as k increases, $\beta_{ij}(k)$ tends towards **positive values** for legitimate agent transmissions and **negative values** for malicious agent transmissions. This enables the dynamic classification of agents into trusted ($U_i(k)$) or Byzantine ($W_i(k)$) sets for agent i at time k:

$$U_i(k) = \{j \in N_i : \beta_{ij}(k) \geq 0\} \quad \text{(trusted neighbours)} \tag{3.19}$$

$$W_i(k) = \{j \in N_i : \beta_{ij}(k) < 0\} \quad \text{(Byzantine neighbours)} \tag{3.20}$$

The longer the observation window k, the higher the probability of correctly classifying agents and the exponential decay of misclassifications errors.

According to [39], trust variables enable **almost sure convergence to a common limit value** even when malicious agents constitute more than half of the network connectivity, a stark contrast to classical impossibility results in consensus systems. The presence of trust observations allows the influence of malicious agents on the consensus value to be **bounded with high probability**. This deviation can be arbitrarily reduced by increasing an initial testing window (T_0) during which agents collect trust values before initiating consensus [39]. The expected convergence rate of the consensus protocol **decays exponentially** as a function of the quality of the trust observations. This eases the correct classification of malicious and legitimate agents in finite time, although characterizing this specific time is beyond the analysis presented here. Trust values can be derived from physical channels of information, such as wireless signals, camera data, lidar, and radar, providing a robust mechanism for data validation in cyber-physical systems. This approach capitalizes on information that often already exists in multi-agent networks[38].

3.4.3 Dynamic Reweighting Scheme

To complete our strategy we then use β_{ij} (more specifically $\mathcal{W}_i(k)$) to help us choose the weightings w_{ij} for the incoming information. These weightings will form the virtual consensus matrix and are chosen as follows:

$$w_{ij}(k) = \begin{cases} w_{ii}(k-1) + \displaystyle\sum_{m \in \mathcal{W}_i(k-1)} w_{im}(k-1), & \text{if } i = j \\ 0, & \text{if } j \in \mathcal{W}_i(k-1) \\ w_{ij}(k-1) & \text{otw.} \end{cases} \tag{3.21}$$

Notice that these weights are designed to preserve double stochasticity in the case that the distributed filtering scheme correctly identifies the attackers. The possibility exits that, if an agent is misclassified enough times during the initial iterations of the consensus then it will be very hard for it to come back and be considered into the good set $\mathcal{L}$. Practically speaking this can be ameliorated by putting special care into the initial conditions of the algorithm as we mentioned before. As consensus ensues, we assume that a well tuned strategy will yield:

$$\mathcal{L} = \bigcup_{i \in \mathcal{V}} \mathcal{U}_i(k), \quad \text{for } \bar{k} \leq k < \infty \tag{3.22}$$

and therefore the assumptions on the trust variables α_{ij} in the aforementioned subsection will hold, leading to a proof for the algortihm simlar to that in [39].

3.5 The Proposed Algorithm

What follows is the decentralized consensus algorithm proposed and designed to handle Byzantine attacks in distributed computing environments. The algorithm is composed of gradient descent optimization with Byzantine fault tolerance to ensure reliable consensus even when some nodes in the network behave maliciously.

3.5.1 Consensus and Optimization

What follow is a comprehensive description of Algorithm 1, which we use for consensus and optimization. The algorithm builds upon the Network-GIANT[40] framework but incorporates Byzantine resilience through a medoid-based filtering system. Each participating worker node begins with several components. A fixed learning rate parameter α that controls how much the algorithm adjusts parameters in each iteration. Each node also receives one row

from a row stochastic matrix W, which ensures that all row sums equal 1. The initial values x_0 for the optimization variables being learned, and an initial gradient variable that tracks gradient information across iterations. Even though the algorithm can start from arbitrary initial points, it is suggested that for this Byzantine-resilient approach all nodes to agree on a common starting point. This is in an effort to facilitate the work of the filter. This coordination may help establish a reference baseline for the Medoid-Based filtering mechanism in the first iterations of consensus. Hence, it prevents widely divergent initial values from interfering with the medoid's ability to distinguish between legitimate data and Byzantine attacks and ensures the filtering algorithm can effectively identify outliers during the initial consensus-building phase.

A feature of this algorithm is that the communication weights between nodes are not static. Instead the weights used in the averaging steps (steps 5 and 7) are dynamically updated using the medoid implementation These updated weights differ from the original doubly stochastic matrix W. At each iteration, the algorithm resets to the original weights w_{ij}^0 and recomputes them, allowing for detection of nodes that may have recovered from Byzantine behavior

Algorithm 1 Byzantine-resilient Decentralized GD

Require: constant step size α; initialization $x_i^0 = x^0$ for all nodes
Require: Doubly stochastic weights $W = [w_{ij}]$

1: **for** $k = 0, 1, 2, \cdots$ **do**
2: **for** all honest workers $i \in \mathcal{V}$ **do**
3: Receive $\{x_j^k\}, \{z_j^k\}$ where $j \in \mathcal{V}_i \cup \{i\}$
4: Reweigh $w_{ij}^{k+1} = \text{medoid\_filter}(x_j^k, w_{ij}^0)$
5: $u_i^{k+1} = \sum_{j \in \mathcal{V}_i \cup \{i\}} w_{ij}^{k+1} \, x_j^k$
6: $x_i^{k+1} = u_i^{k+1} - \alpha \nabla^2 f_i(x_i^k)^{-1} z_i^k$
7: $v_i^{k+1} = \sum_{j \in \mathcal{V}_i \cup \{i\}} w_{ij}^{k+1} \, z_j^k$
8: $z_i^{k+1} = v_i^{k+1} + \nabla f_i(x_i^{k+1}) - \nabla f_i(x_i^k)$
9: Send x_i^{k+1}, z_i^{k+1} to all neighbors
10: **end for**
11: **for** all Byzantine workers **do**
12: Send $x_i^{k+1} = *$ to all neighbors
13: Send $z_i^{k+1} = *$ to all neighbors
14: **end for**
15: **end for**

For Honest Workers (Steps 2-10) each honest node receives parameter values $\{x_j^k\}$ and gradient tracking values $\{z_j^k\}$ from all its neighbors and itself. In step 4 we apply a medoid-based filter to detect and mitigate Byzantine behavior the filter adjusts the communication weights based on how much each neighbor's data deviates from the medoid (most central point). Suspicious or outlying data a zero weight, limiting the influence of Byzantine nodes. Step 5 computes a weighted average of parameter values using the Byzantine-filtered weights. This creates a consensus estimate that is robust against malicious inputs. In Step 6, the parameter x is updated using a Newton-like method, this step uses the Hessian inverse $(\nabla^2 f_i(x_i^k)^{-1})$ for second-order optimization while also incorporating the gradient tracking variable z_i^k as the moving direction. The use of Hessian information provides faster convergence compared to

first-order methods, though at increased computational cost.

Step 7 applies the same Byzantine-filtered weights to average the gradient tracking variables using the difference in gradients received form the node neighbors. This maintains consistency in gradient information across the network. The gradient tracking mechanism ensures that the decentralized algorithm achieves the same convergence properties as centralized methods, even in the presence of Byzantine attacks. Then, each honest node broadcasts its updated parameter and gradient tracking values to all neighbors.

Byzantine nodes send arbitrary values (denoted by $*$) for both parameters and gradient tracking variables. This represents the worst-case scenario where malicious nodes can send any information they choose. The algorithm must be robust against such arbitrary malicious inputs. The algorithm can maintain consensus and convergence even when a subset of nodes behave maliciously, provided the number of Byzantine nodes doesn't exceed certain theoretical bounds.

The medoid-based filtering provides a robust statistical approach to identifying and mitigating the influence of outliers and malicious actors while preserving the benefits of decentralized optimization. In the next subsection we outline this process.

3.5.2 The trust-medoid algorithm

Now, we describe Algorithm 2, which will be in charge of our byzantine defenses. At each node, the filter starts with a T parameter for the soft-medoid. The T parameter will control the soft-medoid interpolation between a weighted average and the medoid. We further require an estimate of the amount of byzantine nodes affecting the node in question, as well as an initial row of stochastic weights w_{ij}. The operation of the filter can be summarized as follows: we initially construct, iteratively, two sets of incoming information using the soft-medoid. The probably Byzantine/Trusted set $\mathcal{W}'/\mathcal{U}'$ will be filtered according to the $[0, 1]$ coefficients output by the medoid. The bigger the coefficient the more trusted the vector parameter.

After this first classification, we draw a trust variable form a distribution (a uniform distribution is depicted in the algorithm). This distribution will need to comply with assumptions made upon the trust variables in the previous section. Namely, that all nodes use the same distribution and that, for the probably trusted/probably malicious set the mean of the distribution is bigger/lesser than 0.5. next we start to build historical information on the trust variables through the variable β. As the consensus ensues through each iteration we accumulate information on each neighboring node. The ones that have a positive sum-of-trusts will form the trusted/byzantine set $\mathcal{U}/\mathcal{W}$.

Finally, based on which neighboring nodes (and original weights) are in the set $\mathcal{W}$ we reweigh the row stochastic vector w given at the start throught the scheme proposed above, ensuring **at least** row stochasticity.

3.6 Numerical Results

In order to numerically shown the effectiveness of the proposed solution we resort to the Covertype and MNIST datasets. Where the full Covertype dataset was used, meaning 581012 samples with $d = 54$ features. While for MNIST, the set was reduced, by means of PCA, to 70000 samples (60000 for training and 10000 for testing) and $d = 300$ features.

The samples were classified using regularized multinomial logistic regression, meaning the local cost functions in each node were:

$$f_i(x) = \frac{1}{m_i} \sum_{j=1}^{m_i} \log\left(1 + e^{-y_j^{(i)}(x^T c_j^{(i)})}\right) + \frac{\lambda}{2}\|x\|^2, \tag{3.23}$$

where λ and α (for the gradient descent) were tuned to values listed below. Numerical test where carried out on 2 types of graphs with different levels of attack coverage. The first graph, a sun graph and a second regular graph. The data will be evenly divided among the

Algorithm 2 Medoid based filtering

Require: Temperature parameter $T > 0$ and, q_i an estimate of the number of attackers
Require: Stochastic weights w_{ij} and vector data $\{\mathbf{x}_j\}$ with $j \in \mathcal{N}_i \cup \{i\}$
Require: A parameter l, the tunable standard deviation for the trust's uniform distribution.

1: Construct an initial set of probably trusted indexes $\mathcal{U}'^0 = \mathcal{N}_i \cup \{i\}$

2: Construct an initial set of probably byzantine indexes $\mathcal{W}'^0 = \emptyset$

3: Construct the weights for the soft-Medoid as:

$$a_{ij} = \frac{1}{|\mathcal{N}_i \cup \{i\}|}, \quad j \in \mathcal{N}_i \cup \{i\}$$

4: **for** $k = 0, 1, \dots q_i - 1$ **do**

5: Compute $\mathbf{r}^k = [\dots a'_j \dots] \quad j \in \mathcal{N}_i \cup \{i\}$ using (3.16), $\{\mathbf{x}_j\}$, and $\{a_{ij}\}$

6: Obtain the index of the minimum weight in $\mathbf{r}^k$ excluding self-info:

$$m^k = \arg \min_{m \in \mathcal{U}'^k \setminus \{i\}} \mathbf{r}^k$$

7: Discard m^k from the probably trusted set $\mathcal{U}'^{k+1} = \mathcal{U}'^k \setminus \{m^k\}$

8: Add m^k to the probably byzantine set $\mathcal{W}'^{k+1} = \mathcal{W}'^k \cup \{m^k\}$

9: **end for**

10: Calculate the new trust by drawing a sample from a uniform: $\alpha'_{ij} \sim U\left[E(\alpha_{ij}) - \frac{l}{2}, E(\alpha_{ij}) + \frac{l}{2}\right]$, where:

$$E(\alpha_{ij}) = 0.55,\ j \in \mathcal{U}'$$
$$E(\alpha_{ij}) = 0.45,\ j \in \mathcal{W}'$$

11: Calculate $\{\beta_{ij}\}, \quad j \in \mathcal{N}_i \cup \{i\}$ using (3.18)

12: Construct $\mathcal{U}_i$ using (3.19) and $\mathcal{W}_i$ using (3.20)

13: Calculate the new stochastic weights $\{w'_{ij}\}$ using (3.21) and $\mathcal{W}_i$

14: **Return** $\{w'_{ij}\}$ and $\{\alpha'_{ij}\}, \quad j \in \mathcal{N}_i \cup \{i\}$

nodes in the corresponding undirected graph. We vary the number of attackers depending of the graph. The Byzantine Attackers perform the following attacks: A **Gaussian attack** in which a byzantine node $b \in \mathcal{B}$ sends a vector x_b^k from a random Gaussian distribution with mean $\bar{x}_b^k = (\sum_{m\in\mathcal{N}_b} w_{bm}x_m^k)/(\sum_{m\in\mathcal{N}_b} w_{bm})$ and variance equal to 1. An **inverse attack**, where b computes its message to send as $x_b^k = -\bar{x}_b^k$ and a **constant attack** where b sends a constant x_b^k value, in this case $x_b^k = \mathbf{0}$.

Results are presented for both the accuracy attained by the classification and the value for the cost function which has been normalized as:

$$\log\left[\frac{\frac{1}{n}\sum_{i=1}^{n} f_i(x) - f(x^*)}{f(x^*)}\right] \tag{3.24}$$

where $f(x^*)$ is a value obtained from a centralized gradient descent optimization on the corresponding data set. The results were then compared for the same logistic regression with other popular successful Byzantine defense strategies[32].

3.6.1 Sun Graph

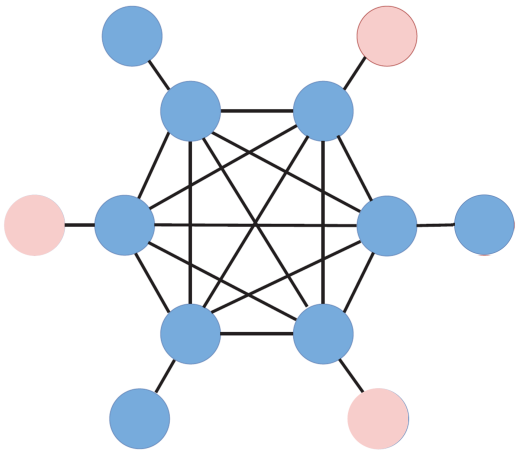

Figure 3.1: Sun Graph

Also called an octopus graph in [32]. As per figure 3.1, we locate 3 attackers and test for accuracy, as a percentage of well classified samples, and for the value of the cost function in Eq. (3.24). Results are presented for the type of attacks mentioned above.

- **Results on the Covertype Dataset**

We first simulate on the Covertype Dataset which has an imbalaced dataset, in terms of the amount of samples per category.

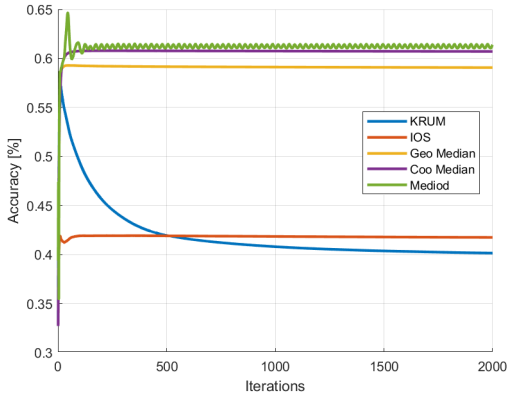

Figure 3.2: Accuracy under an Inverse Attack, CT-Sun Graph

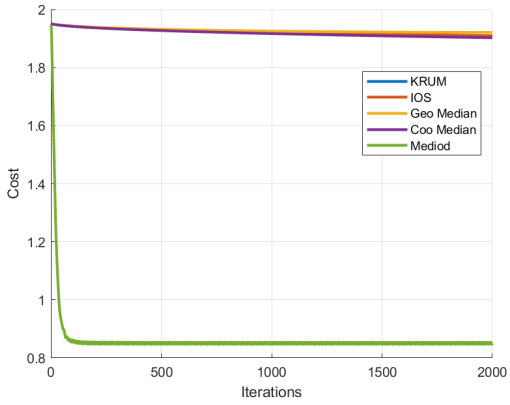

Figure 3.3: Cost under an Inverse Attack, CT-Sun Graph

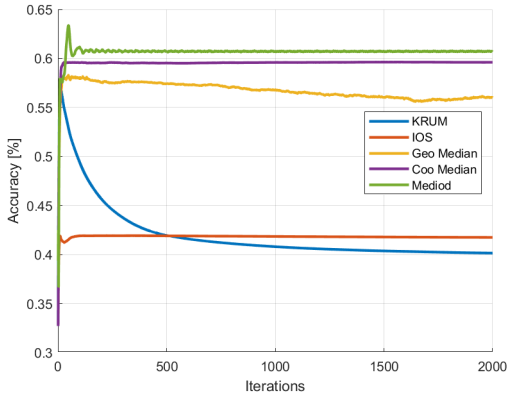

Figure 3.4: Accuracy under a Random Attack, CT-Sun Graph

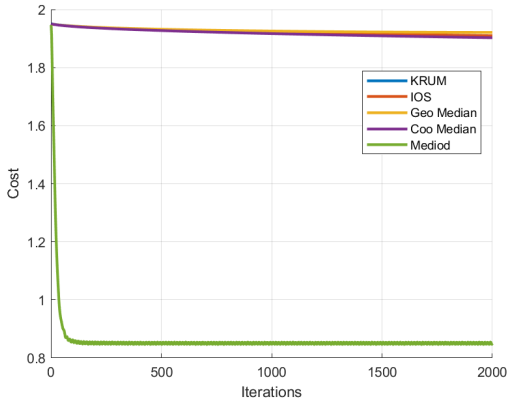

Figure 3.5: Cost under a Random Attack, CT-Sun Graph

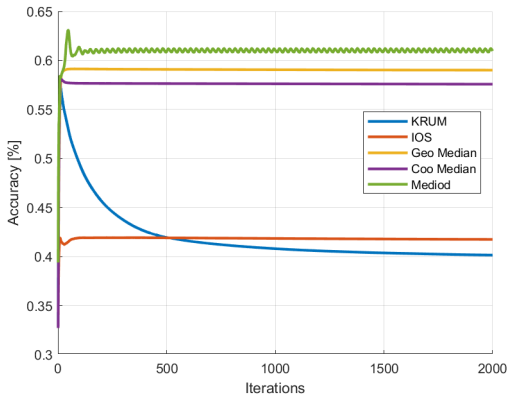

Figure 3.6: Accuracy under a Constant Attack, CT-Sun Graph

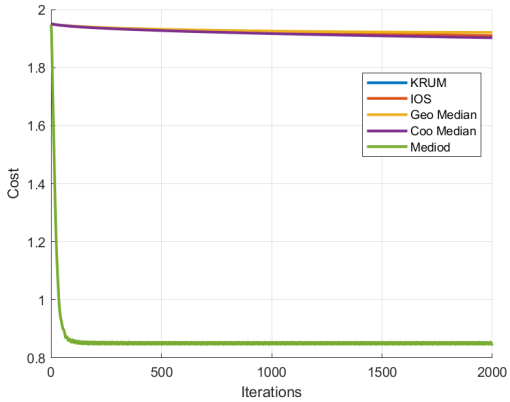

Figure 3.7: Cost under a Constant Attack, CT-Sun Graph

- **Results on the MNIST Dataset**

Now we proceed onto the popular MNIST dataset. The numerical results are for the same type of attacks and the same measures are presented as for the Covertype Dataset.

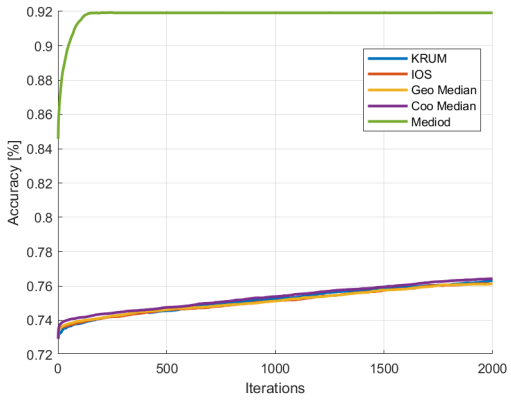

Figure 3.8: Accuracy under an Inverse Attack, MNIST-Sun Graph

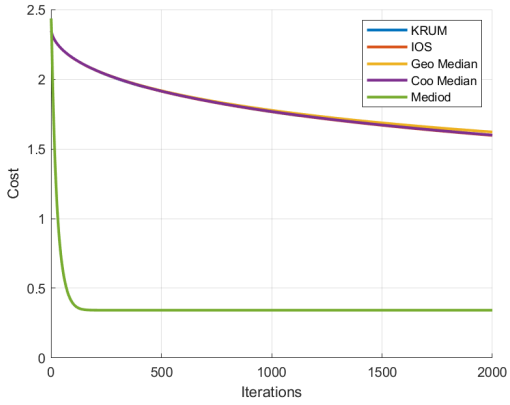

Figure 3.9: Cost under an Inverse Attack, MNIST-Sun Graph

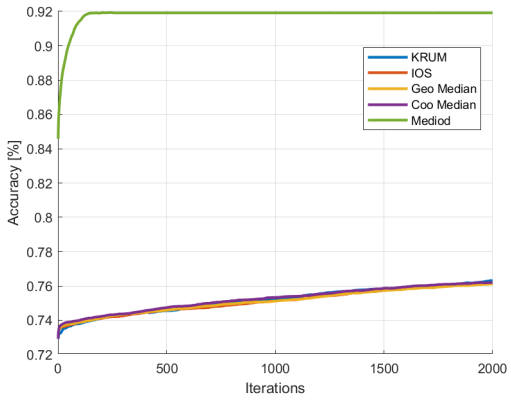

Figure 3.10: Accuracy under a Random Attack, MNIST-Sun Graph

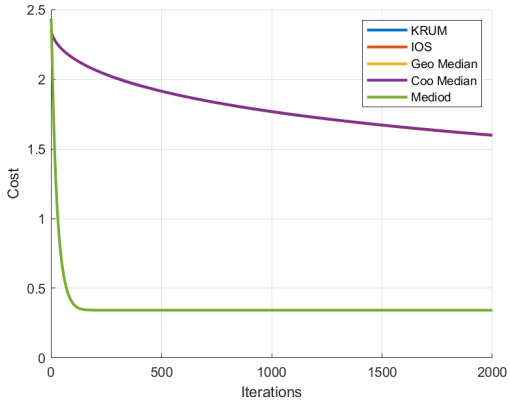

Figure 3.11: Cost under a Random Attack, MNIST-Sun Graph

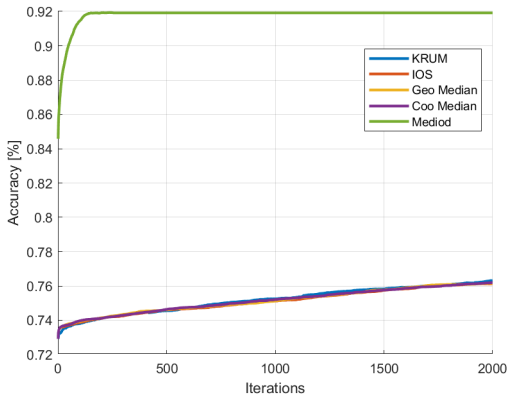

Figure 3.12: Accuracy under a Constant Attack, MNIST-Sun Graph

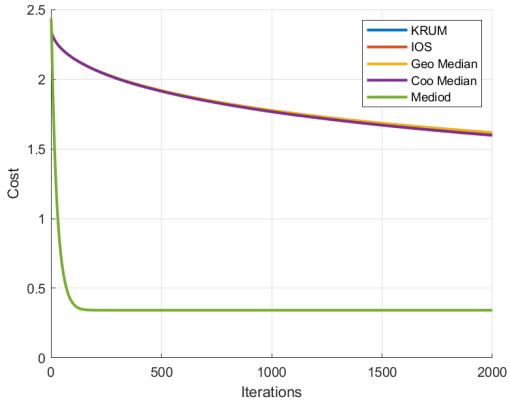

Figure 3.13: Cost under a Constant Attack, MNIST-Sun Graph

3.6.2 Regular Graph

We conclude the numerical simulations by using a Regular Graph, with degree of connection $d = 4$ and with affected nodes as in figure 3.14. The we test for accuracy, as a percentage of well classified samples, and for the value of the cost function in Eq. (3.24). Results are presented for the type of attacks mentioned at the start of this section.

- **Results on the Covertype Dataset**

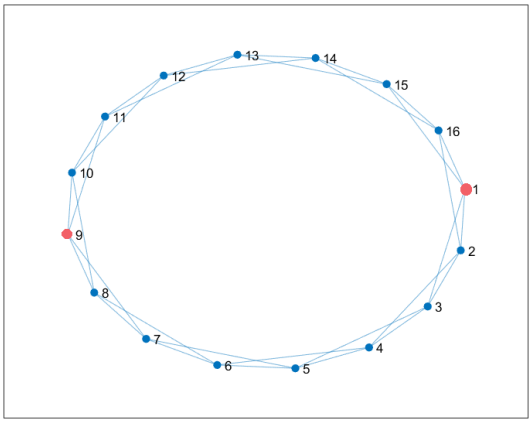

Figure 3.14: Regular Graph $d = 4$

First, we simulate on the Covertype Dataset which has an imbalaced dataset, in terms of the amount of samples per category.

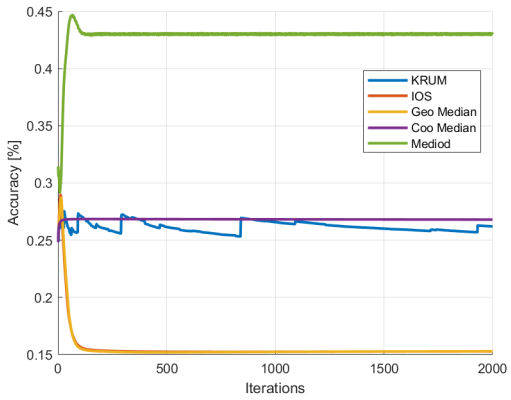

Figure 3.15: Accuracy under an Inverse Attack, CT-Reg Graph

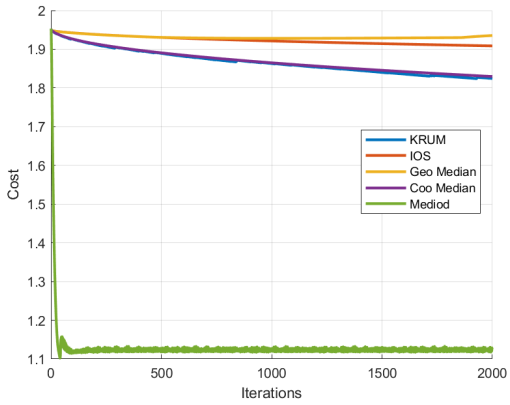

Figure 3.16: Cost under an Inverse Attack, CT-Reg Graph

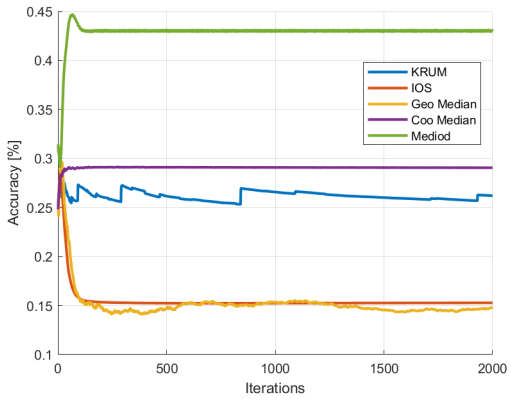

Figure 3.17: Accuracy under a Random Attack, CT-Reg Graph

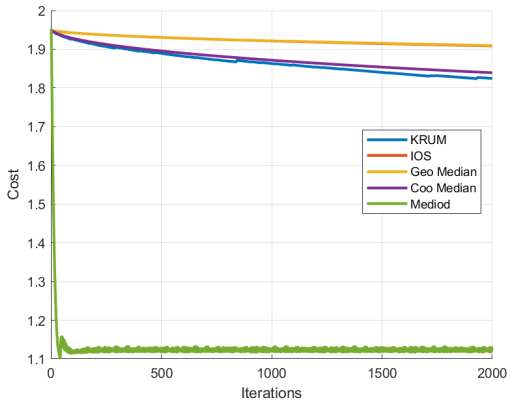

Figure 3.18: Cost under a Random Attack, CT-Reg Graph

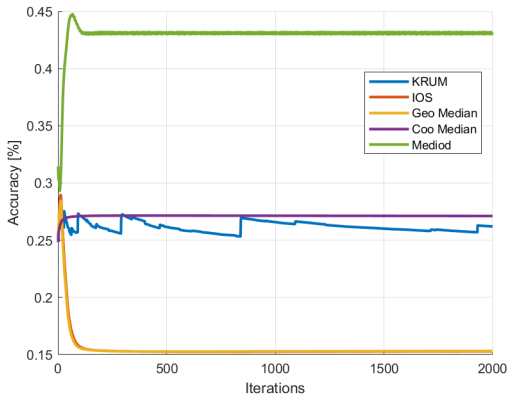

Figure 3.19: Accuracy under a Constant Attack, CT-Reg Graph

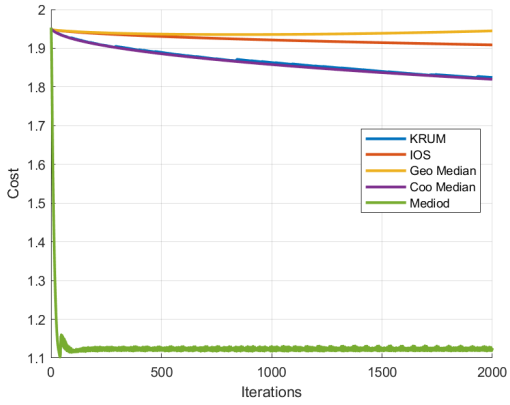

Figure 3.20: Cost under a Constant Attack, CT-Reg Graph

- **Results on the MNIST Dataset**

Lastly, we look at the performance on the MNIST dataset. The numerical results are for the same type of attacks and the same measures are presented as for the Covertype Dataset.

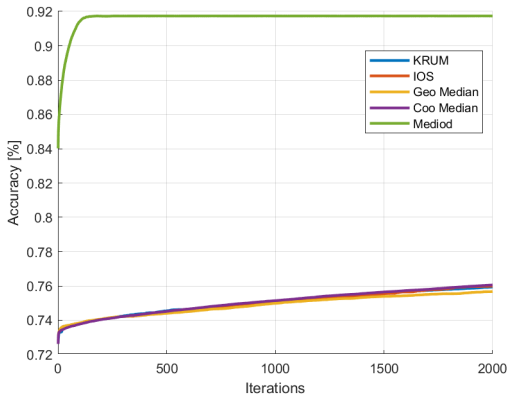

Figure 3.21: Accuracy under an Inverse Attack, MNIST-Reg Graph

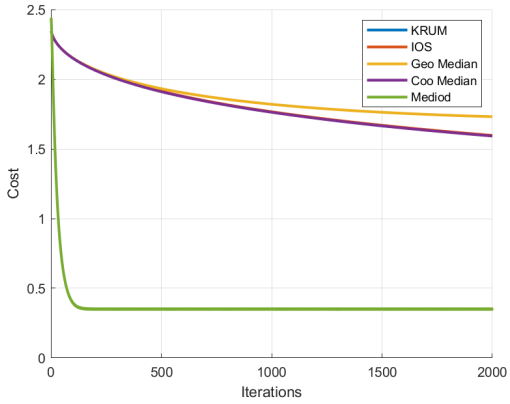

Figure 3.22: Cost under an Inverse Attack, MNIST-Reg Graph

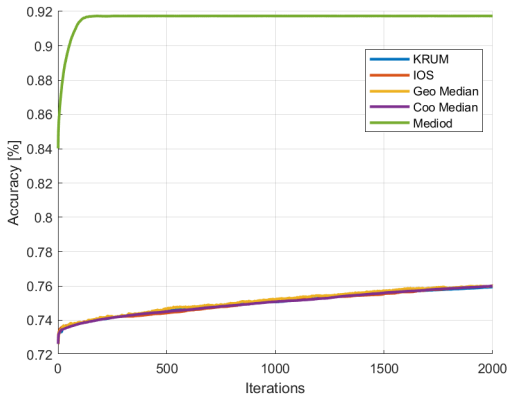

Figure 3.23: Accuracy under a Random Attack, MNIST-Reg Graph

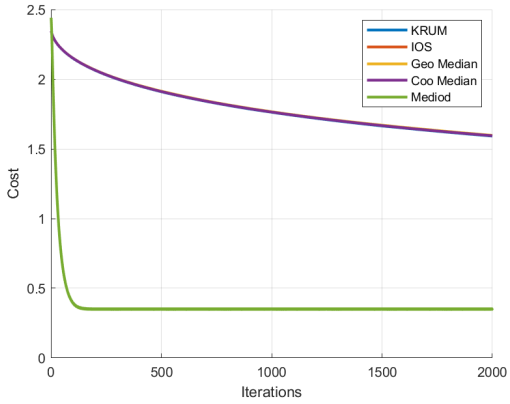

Figure 3.24: Cost under a Random Attack, MNIST-Reg Graph

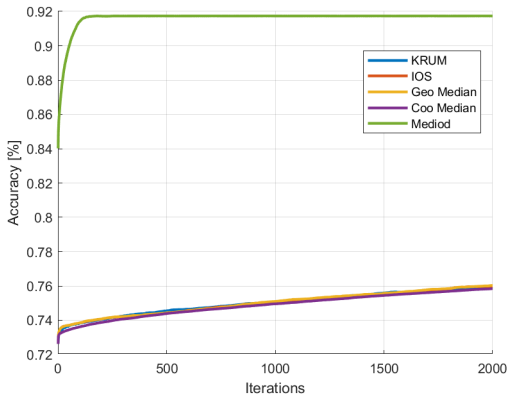

Figure 3.25: Accuracy under a Constant Attack, MNIST-Reg Graph

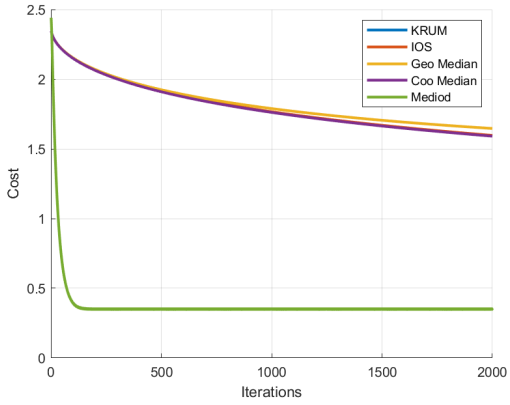

Figure 3.26: Cost under a Constant Attack, MNIST-Reg Graph

3.7 Discussion

3.7.1 Observations on the Sun Graph

For the Covertype Dataset, our Algorithm 2 outperforms commonly used methods in terms of speed of convergence. This is expected due to the usage of a Newton-like algorithm. In terms of accuracy, it is consistent with the standard accuracy reached by a multinomial classifier for this specific data set, furthermore it shows significant improvement over the other Robust Aggregation rules specially in the cases of Random and Constant Attacks.

With regards to the MNIST dataset, a much more pronounced accuracy is noticed. This is due in part to the prescense of a balanced sample set in each node, which eases the classification process, an advantage that trickles over to the filtering process. In terms of the cost function, again, as expected a much faster convergence is present, with our proposal topping over the current solutions in the literature.

It is worth mentioning that, part of the success in this present example (Fig. 3.1) is due to the attackers being the outer nodes, in a way, this an injection attack that can be easily countered by a good robust rule such as the Medoid, since most of the neighbors of the attacked nodes are clean.

3.7.2 Observations on the Regular Graph

A more difficult case presents itself for the regular graph in Fig. 3.14. The high degree of connectivity and the fact that the attackers are connected to multiple nodes yields the following: 2 attacker affecting 8 nodes in total, leaving 6 healthy nodes to help the nodes being attacked to filter out the Byzantine nodes.

The Covertype accuracy has been degraded noticeably. However our implementation still manages to keep a consistent response despite degraded accuracy. The cost function remains steep in comparison to the other Byzantine defenses. It is worth mentioning that, although fast, the Newton+Medoid algorithm cannot decrease the cost as farther as before, mainly due to the influence of the attackers in the network

The MNIST data set numerical simulations remain similar to the previous case. Displaying a good accuracy and fast convergence towards the optimal cost. This implies that the degraded performance on the Covertype dataset may also be due to the imbalance of the number of sample types held by each node.

Conclusions

Regarding the rate of convergence part of this work, we have offered a more compact description and eigenvalue analysis of the R-ADMM. We restructured the R-ADMM into a linear form for regular graphs with functions of the same convexity, to then proceed to find the sprectum of interest in closed form. From this analysis, $\alpha > 0.5$ was shown to increase the algorithm's performance, a finding that was corroborated by numerical experiments. The presence of a tunable α hints at the superiority of R-ADMM over classical ADMM but this remains unproven in the general case.

For Byzantine attacks, the thesis proposed a new Meodoid-Trust scheme to identifying and filter adversarial agents in a distributed maner. This approach, detailed in Algorithm 1 and Algorithm 2, successfully guarantees convergence, ensuring that we lie in the neighborhood of the optimal solution (a fact we suspect to be true due to the proof in [39]), and achieves correct synchronized distributed filtering. The proposed Soft Medoid, a differentiable generalization of the traditional Medoid, was proposed due to its desirable statistical properties, including an asymptotic breakdown point of 0.5, making it highly robust against arbitrary distortions. Additionally, the incorporation of trust variables and a dynamic reweighting scheme allows the system to build historical information on node trustworthiness. This has helped to obtain reliable classification of agents, preserving (in the best of cases) network properties like double stochasticity. Numerical simulations, conducted on common machine learning datasets such as Covertype and MNIST, consistently demonstrated the effectiveness of our proposed strategy; with superior accuracy and faster convergence compared to other robust aggregation rules.

Future Work. When talking about the realm of convergence properties of consensus algorithms, the possibilities to extend the current topics listed in this work are quite vast. More specifically, it would be a useful contribution to strictly prove that α does in fact enhance the convergence speed of the R-ADMM for specific cases. The more general case eludes us due to a lack closed form solution to the R-ADMM optimization step. However, restricting the analysis to commonly used quadratics, could lead to an interesting characterization of the hyper-parameters α and ρ. With respect to consensus algorithms under the Byzantine attack case, one significant fact remains to be proven and it is an statistical characterization of the Medoid in therms of an arbitrary attack. Meaning, there is still research to be done regarding the effectiveness of the Medoid, we hope that a strong result, such as showing that the Medoid is able to correctly filter the attackers in finite time with rpobability 1, could be attained in the future.

Bibliography

[1] Francesco Bullo. *Lectures on Network Systems*. First edition. North Charleston, South Carolina: CreateSpace, 2018. 1 p. ISBN: 978-1-9864-2564-3.

[2] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. 1st ed. Cambridge University Press, Mar. 8, 2004. ISBN: 978-0-521-83378-3. DOI: 10.1017/CBO9780511804441. URL: https://www.cambridge.org/core/product/identifier/9780511804441/type/book (visited on 06/15/2025).

[3] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer Series in Operations Research. New York: Springer, 1999. ISBN: 978-0-387-98793-4.

[4] Angelia Nedic. "Distributed Gradient Methods for Convex Machine Learning Problems in Networks: Distributed Optimization". In: *IEEE Signal Processing Magazine* 37.3 (May 2020), pp. 92–101. ISSN: 1053-5888, 1558-0792. DOI: 10.1109/MSP.2020.2975210. URL: https://ieeexplore.ieee.org/document/9084356/ (visited on 06/15/2025).

[5] Ran Xin et al. "A General Framework for Decentralized Optimization With First-Order Methods". In: *Proceedings of the IEEE* 108.11 (Nov. 2020), pp. 1869–1889. ISSN: 0018-9219, 1558-2256. DOI: 10.1109/JPROC.2020.3024266. URL: https://ieeexplore.ieee.org/document/9241497/ (visited on 06/15/2025).

[6] Stephen Boyd. "Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers". In: *Foundations and Trends® in Machine Learning* 3.1 (2010), pp. 1–122. ISSN: 1935-8237, 1935-8245. DOI: 10.1561/2200000016. URL: http://www.nowpublishers.com/article/Details/MAL-016 (visited on 06/15/2025).

[7] N. Bastianello et al. "Distributed Optimization over Lossy Networks via Relaxed Peaceman-Rachford Splitting: A Robust ADMM Approach". In: *2018 European Control Conference (ECC)*. 2018 17th European Control Conference (ECC). Limassol: IEEE, June 2018, pp. 477–482. ISBN: 978-3-9524269-8-2. DOI: 10.23919/ECC.2018.8550322. URL: https://ieeexplore.ieee.org/document/8550322/ (visited on 06/15/2025).

[8] Angelia Nedić and Ji Liu. "Distributed Optimization for Control". In: *Annual Review of Control, Robotics, and Autonomous Systems* 1.1 (May 2018), pp. 77–103. ISSN: 2573-5144, 2573-5144. DOI: 10.1146/annurev-control-060117-105131. (Visited on 02/08/2025).

[9] Gianluca Antonelli. "Interconnected Dynamic Systems: An Overview on Distributed Control". In: *IEEE Control Systems Magazine* 33.1 (Feb. 2013), pp. 76–88. ISSN: 1941-000X. DOI: 10.1109/MCS.2012.2225929. (Visited on 02/10/2025).

[10] Marcello Farina and Ruggero Carli. "Partition-Based Distributed Kalman Filter With Plug and Play Features". In: *IEEE Transactions on Control of Network Systems* 5.1 (Mar. 2018), pp. 560–570. ISSN: 2325-5870. DOI: 10.1109/TCNS.2016.2633786. (Visited on 02/10/2025).

[11] Saverio Bolognani et al. "A Randomized Linear Algorithm for Clock Synchronization in Multi-Agent Systems". In: *IEEE Transactions on Automatic Control* 61.7 (July 2016), pp. 1711–1726. ISSN: 1558-2523. DOI: 10.1109/TAC.2015.2479136. (Visited on 02/10/2025).

[12] Ji Liu et al. "From distributed machine learning to federated learning: a survey". en. In: *Knowledge and Information Systems* 64.4 (Apr. 2022), pp. 885–917. ISSN: 0219-3116. DOI: 10.1007/s10115-022-01664-x. URL: https://doi.org/10.1007/s10115-022-01664-x (visited on 02/08/2025).

[13] R. Olfati-Saber and J.S. Shamma. "Consensus Filters for Sensor Networks and Distributed Sensor Fusion". In: *Proceedings of the 44th IEEE Conference on Decision and Control*. ISSN: 0191-2216. Dec. 2005, pp. 6698–6703. DOI: 10.1109/CDC.2005.1583238. (Visited on 02/08/2025).

[14] Marco Todescato et al. "Online Distributed Voltage Stress Minimization by Optimal Feedback Reactive Power Control". In: *IEEE Transactions on Control of Network Systems* 5.3 (Sept. 2018), pp. 1467–1478. ISSN: 2325-5870. DOI: 10.1109/TCNS.2017.2722818. (Visited on 02/10/2025).

[15] Konstantinos I. Tsianos, Sean Lawlor, and Michael G. Rabbat. "Consensus-Based Distributed Optimization: Practical Issues and Applications in Large-Scale Machine Learning". In: *2012 50th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. Oct. 2012, pp. 1543–1550. DOI: 10.1109/Allerton.2012.6483403. (Visited on 02/08/2025).

[16] Angelia Nedic and Asuman Ozdaglar. "Distributed Subgradient Methods for Multi-Agent Optimization". In: *IEEE Transactions on Automatic Control* 54.1 (Jan. 2009), pp. 48–61. ISSN: 1558-2523. DOI: 10.1109/TAC.2008.2009515. (Visited on 02/10/2025).

[17] Alessio Maritan et al. "Fully-Distributed Optimization with Network Exact Consensus-GIANT". In: *2024 IEEE 25th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*. Sept. 2024, pp. 436–440. DOI: 10.1109/SPAWC60668.2024.10694261. (Visited on 02/08/2025).

[18] Nicoletta Bof et al. "Multiagent Newton–Raphson Optimization Over Lossy Networks". In: *IEEE Transactions on Automatic Control* 64.7 (July 2019), pp. 2983–2990. ISSN: 1558-2523. DOI: 10.1109/TAC.2018.2874748. (Visited on 02/10/2025).

[19] Pascal Bianchi, Walid Hachem, and Franck Iutzeler. "A Coordinate Descent Primal-Dual Algorithm and Application to Distributed Asynchronous Optimization". In: *IEEE Transactions on Automatic Control* 61.10 (Oct. 2016), pp. 2947–2957. ISSN: 1558-2523. DOI: 10.1109/TAC.2015.2512043. (Visited on 02/10/2025).

[20] Ivano Notarnicola, Ruggero Carli, and Giuseppe Notarstefano. "Distributed Partitioned Big-Data Optimization via Asynchronous Dual Decomposition". In: *IEEE Transactions on Control of Network Systems* 5.4 (Dec. 2018), pp. 1910–1919. ISSN: 2325-5870. DOI: 10.1109/TCNS.2017.2774010. (Visited on 02/10/2025).

[21] Guido Carnevale et al. "ADMM-Tracking Gradient for Distributed Optimization over Asynchronous and Unreliable Networks". In: *IEEE Transactions on Automatic Control* (2025).

[22] Zhimin Peng et al. "ARock: An Algorithmic Framework for Asynchronous Parallel Coordinate Updates". In: *SIAM Journal on Scientific Computing* 38.5 (Jan. 2016), A2851–A2879. ISSN: 1064-8275. DOI: 10.1137/15M1024950. (Visited on 02/10/2025).

[23] Nicola Bastianello and Ruggero Carli. "ADMM for Dynamic Average Consensus over Imperfect Networks". In: *IFAC-PapersOnLine*. 9th IFAC Conference on Networked Systems NECSYS 2022 55.13 (Jan. 2022), pp. 228–233. ISSN: 2405-8963. DOI: 10.1016/j.ifacol.2022.07.264. (Visited on 02/10/2025).

[24] Alessandro Falsone et al. "Tracking-ADMM for Distributed Constraint-Coupled Optimization". In: *Automatica* 117 (July 2020), p. 108962. ISSN: 0005-1098. DOI: 10.1016/j.automatica.2020.108962. (Visited on 02/10/2025).

[25] Nicola Bastianello, Luca Schenato, and Ruggero Carli. "A Novel Bound on the Convergence Rate of ADMM for Distributed Optimization". In: *Automatica* 142 (Aug. 2022), p. 110403. ISSN: 0005-1098. DOI: 10.1016/j.automatica.2022.110403. (Visited on 02/08/2025).

[26] Nicoletta Bof, Ruggero Carli, and Luca Schenato. "Is ADMM Always Faster than Average Consensus?" In: *Automatica* 91 (May 2018), pp. 311–315. ISSN: 0005-1098. DOI: 10.1016/j.automatica.2018.01.009. (Visited on 02/10/2025).

[27] Nicola Bastianello et al. "Asynchronous Distributed Optimization Over Lossy Networks via Relaxed ADMM: Stability and Linear Convergence". In: *IEEE Transactions on Automatic Control* 66.6 (June 2021), pp. 2620–2635. ISSN: 1558-2523. DOI: 10.1109/TAC.2020.3011358. (Visited on 02/08/2025).

[28] Leonardo Armijos-Bacuilima et al. "On the Rate of Convergence of Distributed Relaxed-ADMM Algorithms in Distributed Optimization". In: *IFAC Conference on Networked Systems*. IFAC Conference on Networked Systems. June 5, 2025.

[29] Lili Su and Nitin Vaidya. *Fault-Tolerant Multi-Agent Optimization: Part III*. Version 1. 2015. DOI: 10.48550/ARXIV.1509.01864. URL: https://arxiv.org/abs/1509.01864 (visited on 06/21/2025). Pre-published.

[30] Dong Yin et al. *Byzantine-Robust Distributed Learning: Towards Optimal Statistical Rates*. Version 2. 2018. DOI: 10.48550/ARXIV.1803.01498. URL: https://arxiv.org/abs/1803.01498 (visited on 06/21/2025). Pre-published.

[31] Zhixiong Yang, Arpita Gang, and Waheed U. Bajwa. "Adversary-Resilient Distributed and Decentralized Statistical Inference and Machine Learning: An Overview of Recent Advances Under the Byzantine Threat Model". In: *IEEE Signal Processing Magazine* 37.3 (May 2020), pp. 146–159. ISSN: 1053-5888, 1558-0792. DOI: 10.1109/MSP.2020.2973345. URL: https://ieeexplore.ieee.org/document/9084329/ (visited on 06/21/2025).

[32] Zhaoxian Wu, Tianyi Chen, and Qing Ling. "Byzantine-Resilient Decentralized Stochastic Optimization With Robust Aggregation Rules". In: *IEEE Transactions on Signal Processing* 71 (2023), pp. 3179–3195. ISSN: 1053-587X, 1941-0476. DOI: 10.1109/TSP.2023.3300629. URL: https://ieeexplore.ieee.org/document/10208131/ (visited on 06/21/2025).

[33] Yudong Chen, Lili Su, and Jiaming Xu. *Distributed Statistical Machine Learning in Adversarial Settings: Byzantine Gradient Descent*. Version 2. 2017. DOI: 10.48550/ARXIV.1705.05491. URL: https://arxiv.org/abs/1705.05491 (visited on 06/21/2025). Pre-published.

[34] Peva Blanchard et al. *Byzantine-Tolerant Machine Learning*. Version 1. 2017. DOI: 10.48550/ARXIV.1703.02757. URL: https://arxiv.org/abs/1703.02757 (visited on 06/21/2025). Pre-published.

[35] Cong Xie, Oluwasanmi Koyejo, and Indranil Gupta. *Phocas: Dimensional Byzantine-resilient Stochastic Gradient Descent*. Version 1. 2018. DOI: 10.48550/ARXIV.1805.09682. URL: https://arxiv.org/abs/1805.09682 (visited on 06/21/2025). Pre-published.

[36] Hendrik P. Lopuhaa and Peter J. Rousseeuw. "Breakdown Points of Affine Equivariant Estimators of Multivariate Location and Covariance Matrices". In: *The Annals of Statistics* 19.1 (Mar. 1, 1991). ISSN: 0090-5364. DOI: 10.1214/aos/1176347978. URL: https://projecteuclid.org/journals/annals-of-statistics/volume-19/issue-1/Breakdown-Points-of-Affine-Equivariant-Estimators-of-Multivariate-Location-and/10.1214/aos/1176347978.full (visited on 06/21/2025).

[37] Simon Geisler, Daniel Zügner, and Stephan Günnemann. *Reliable Graph Neural Networks via Robust Aggregation*. Version 1. 2020. DOI: 10.48550/ARXIV.2010.15651. URL: https://arxiv.org/abs/2010.15651 (visited on 06/21/2025). Pre-published.

[38] Stephanie Gil et al. "Guaranteeing spoof-resilient multi-robot networks". In: *Autonomous Robots* 41.6 (Aug. 1, 2017), pp. 1383–1400. ISSN: 1573-7527. DOI: 10.1007/s10514-017-9621-5. URL: https://doi.org/10.1007/s10514-017-9621-5.

[39] Michal Yemini et al. *Characterizing Trust and Resilience in Distributed Consensus for Cyberphysical Systems*. 2025. arXiv: 2103.05464 [math.OC]. URL: https://arxiv.org/abs/2103.05464.

[40] Alessio Maritan et al. "Network-GIANT: Fully Distributed Newton-Type Optimization via Harmonic Hessian Consensus". In: *2023 IEEE Globecom Workshops (GC Wkshps)*. 2023 IEEE Globecom Workshops (GC Wkshps). Kuala Lumpur, Malaysia: IEEE, Dec. 4, 2023, pp. 902–907. ISBN: 979-8-3503-7021-8. DOI: 10.1109/GCWkshps58843.2023.10464472. URL: https://ieeexplore.ieee.org/document/10464472/ (visited on 06/21/2025).